БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
МЕХАНИКО-МАТЕМАТИЧЕСКИЙ ФАКУЛЬТЕТ
Кафедра веб-технологий и компьютерного моделирования

А. С. Кравчук, А. И. Кравчук, Е. В. Кремень

ЯЗЫК JAVA

Система тестов

Учебные материалы
для студентов специальности 1-31 03 08
«Математика и информационные технологии
(по направлениям)»

МИНСК
2023

УДК 004.432.045:004.738.5Java (075.8)
ББК 32.973.2-018.1я73-1
К78

Рекомендовано советом
механико-математического факультета БГУ
26 января 2023 г., протокол № 5

Рецензент
кандидат технических наук, доцент *М. Н. Садовская*

Кравчук, А. С.

К78 Язык Java. Система тестов : учеб. материалы для студентов спец. 1-31 03 08 «Математика и информационные технологии (по направлениям)» / А. С. Кравчук, А. И. Кравчук, Е. В. Кремень. – Минск : БГУ, 2023. – 147 с.

Представлены вопросы, которые можно использовать для проведения тестов по темам: «Общие сведения о Java», «Основы синтаксиса. Неименованные константы», «Операции», «Типы», «Объявления», «Операторы управления программой», «Методы», «ООП», «Дженерики», «Коллекции», «Лямбда-выражения», «Потоки ввода-вывода», «Stream API» при изучении языка Java. Используя тесты, каждый самостоятельно может не только объективно оценить свои знания по теме, но и дополнить по определенным аспектам языка. Издание ориентировано как на тех, кто не имеет опыта практического программирования на языке Java, так и на тех, кто хотел бы систематизировать и улучшить свои знания.

УДК 004.432.045:004.738.5Java (075.8)
ББК 32.973.2-018.1я73-1

© Кравчук А. С., Кравчук А. И., Кремень Е. В., 2023
© БГУ, 2023

Оглавление

Введение

Использование тестирования для контроля знаний прочно вошло в практику обучения. Тестирование является достаточно объективным способом оценивания, поскольку ставит всех учащихся в равные условия, как в процессе контроля, так и в процессе оценки. Тестирование по отдельным темам пройденного материала позволяет оценить знания по теме, исключив элемент случайности при вытаскивании билета на экзамене, когда вопросы по данной теме могут и не попасть в конкретный билет. Каждый самостоятельно может не только объективно оценить свои знания по различным темам, но и сконцентрировать своё внимание на определенных аспектах языка, по которым есть вопросы в тесте, но по каким-либо причинам они ускользнули от внимания тестируемого.

Данная издание представляет собой банк вопросов, который можно использовать для создания и проведения тестов по темам «Общие сведения о Java», «Основы синтаксиса. Неименованные константы», «Операции», «Типы», «Объявления», «Операторы управления программой», «Методы», «ООП», «Дженерики», «Коллекции», «Лямбда-выражения», «Потоки ввода-вывода», «Stream API» при изучении языка Java.

Общие сведения о Java

1. **Что означает портируемость (portability) программного обеспечения написанного на определенном языке программирования?**
    - возможность адаптации некоторой программы или ее части, написанных на определенном языке программирования, с тем чтобы она работала в другой среде, отличающейся от той среды, под которую она была изначально написана с максимальным сохранением ее пользовательских свойств;
    - способность программы (или её частей), быть запущенными в другой среде/окружении. Можно подразумевать как способность работы на разных ОС, так и просто перенос на разные компьютеры с одной ОС, но их разными настройками;
    - способность программного обеспечения работать с несколькими аппаратными платформами или операционными системами.

2. **Общие сведения о Java. Что означает переносимость программного обеспечения написанного на определенном языке программирования?**
    - возможность адаптации некоторой программы или ее части, написанных на определенном языке программирования, с тем чтобы она работала в другой среде, отличающейся от той среды, под которую она была изначально написана с максимальным сохранением ее пользовательских свойств;

- способность программы (или её частей), быть запущенными в другой среде/окружении. Можно подразумевать как способность работы на разных ОС, так и просто перенос на разные компьютеры с одной ОС, но их разными настройками;
- способность программного обеспечения работать с несколькими аппаратными платформами или операционными системами.

3. **Общие сведения о Java. Что означает кроссплатформенность программного обеспечения написанного на определенном языке программирования?**
    - возможность адаптации некоторой программы или ее части, написанных на определенном языке программирования, с тем чтобы она работала в другой среде, отличающейся от той среды, под которую она была изначально написана с максимальным сохранением ее пользовательских свойств;
    - способность программы (или её частей), быть запущенными в другой среде/окружении. Можно подразумевать как способность работы на разных ОС, так и просто перенос на разные компьютеры с одной ОС, но их разными настройками;
    - способность программного обеспечения работать с несколькими аппаратными платформами или операционными системами.

4. **Общие сведения о Java. В разработке каких приложений обычно используется язык Java?**
    - веб-приложений;
    - некоторых серверных приложений;
    - трейдинговых приложений;
    - приложения, работающие на смарт-картах;
    - разработка кроссплатформенных приложений.

5. **Общие сведения о Java. Является ли Java компилируемым и/или интерпретируемым языком?**
    - только компилируемым;
    - только интерпретируемым;
    - и компилируемым, и интерпретируемым.

6. **Общие сведения о Java. Что означает термин «статическая типизация»?**
    - переменная связывается с типом в момент присваивания значения, а не в момент объявления переменной;
    - переменная связывается с типом в момент объявления и тип не может быть изменён позже;
    - тип переменной может изменяться в процессе работы программы;
    - переменная имеет тип, известный на момент компиляции.

7. **Общие сведения о Java. Какие из следующих утверждений верны?**
    - Java существует базовые типы данных, которые не являются объектами;
    - в Java все типы данных являются объектами;
    - базовые типы являются объектами;
    - ссылочные типы являются объектами.

8. **Общие сведения о Java. Какое из следующих утверждений верно?**
    - в Java можно использовать множественное наследование реализаций;
    - в Java можно использовать только простое наследование;
    - в Java можно создавать многопоточные приложения;
    - в Java можно вручную освобождать память, выделенную под объекты.

9. **Общие сведения о Java. Назначение Java Virtual Machine (JVM)?**
    - исполнение Java-программ, а также контроль за безопасностью исполнения приложений;
    - компилирует и запускает Java-программы на исполнение;
    - позволяет разработчикам создавать программы, которые могут выполняться и запускаться.

10. **Общие сведения о Java. Назначение Java Runtime Environment (JRE)?**
    - исполнение Java-программ;
    - компилирует и запускает Java-программы на исполнение;
    - позволяет разработчикам создавать программы, которые могут выполняться и запускаться.

11. **Общие сведения о Java. Назначение Java Development Kit (JDK)?**
    - исполнение Java-программ;
    - компилирует и запускает Java-программы на исполнение;
    - позволяет разработчикам создавать программы, компилировать их и запускать на исполнение.

12. **Общие сведения о Java. Какие утверждения относительно виртуальной машины Java Virtual Machine (JDM) верны?**
    - позволяет отсекать опасный код на каждом этапе работы;
    - периодически запускает сборщик мусора;
    - для исполнения приложения на языке Java на какой-либо операционной системе должна быть создана своя виртуальная машина;
    - включает ряд программ и утилит, которые позволяют компилировать, запускать программы на Java, а также выполнять ряд других функций;
    - поддерживает запуск приложений из файлов с расширением .txt.

13. Общие сведения о Java. Виртуальная машина Java Virtual Machine (JDM) это?

- специальная операционная система;
- среда исполнения Java-приложений, которая пишется специально для каждой платформы (операционной системы);
- библиотека, предоставляющая единый API для разных платформ;
- ничего из перечисленного.

14. Общие сведения о Java. Какое из следующих утверждений верно?

- две наиболее популярных реализации JDK – это Oracle JDK и OpenJDK;
- Oracle JDK всецело развивается компанией Oracle;
- OpenJDK развивается как компанией Oracle, так и еще рядом компаний совместно;
- Oracle JDK можно использовать бесплатно для персональных нужд, а также для разработки, тестирования и демонстрации приложений;
- в случае Oracle JDK для получения поддержки необходима коммерческая лицензия в виде подписки;
- функционально Oracle JDK и OpenJDK практически не отличаются.

15. Общие сведения о Java. Какие из перечисленных свойств являются общими для программ, написанных на C++ и Java?

- поддерживают обращения к физической памяти;
- компилируются в машинные коды;
- имеют схожий синтаксис;
- являются процедурными.

16. Общие сведения о Java. Динамическая компиляция в Java?

- динамическая (Just-In-Time) компиляция повышает производительность за счет трансляции Java байт-кода в машинный код в процессе работы приложения;
- повышает скорость выполнения приложений;
- при динамической компиляции достигается высокий уровень оптимизации программного кода компиляция.

17. Общие сведения о Java. Какие меры позволяют Java обеспечивать безопасность?

- правила работы с памятью;
- наличие виртуальной машины-интерпретатора;
- наличие Just In Time-компилятора;
- сертификаты для приложений, загружаемых по сети;
- политики (policy) и разрешения (permission);

- встроенные ограничения в стандартных библиотеках;
- наличие менеджера безопасности, загрузчика классов и верификатора.

18. Общие сведения о Java. Какое из следующих утверждений верно?

- адаптивная оптимизация — технология, которая производит динамическую перекомпиляцию фрагментов программы, основываясь на текущем профиле исполнения;
- адаптивный оптимизатор может просто выбирать между компиляцией «на лету» и интерпретацией инструкций;
- адаптивная оптимизация может использовать оптимизации ветвлений, программных блоков и др., чтобы уменьшить количество переключений контекста.

19. Общие сведения о Java. Выбрать пункты соответствующие инкрементной модели разработки программного обеспечения?

- тестирование выполняется в конце проекта, когда программное обеспечение удовлетворяет всем требованиям заказчика;
- это метод разработки, в котором проект проектируется, реализуется и тестируется последовательно каждый раз с небольшими добавлениями до самого окончания разработки;
- данный подход позволяет бороться с неопределенностью, снимая ее этап за этапом, и проверять правильность технического, маркетингового или любого другого решения на ранних стадиях выполнения проекта;
- продукт считается законченным в то время, когда удовлетворяет всем требованиям заказчика;
- представляет собой пример итеративного подхода к разработке программного обеспечения, который предполагает разбиение жизненного цикла проекта на последовательность итераций, каждая из которых напоминает «мини-проект», включающий все фазы жизненного цикла в применении к созданию отдельных версий системы, обладающих меньшей функциональностью по сравнению с проектом, в целом.

20. Общие сведения о Java. Преимущества инкрементной модели разработки программного обеспечения?

- на момент создания определенного инкремента требования упрощаются, поскольку есть возможность перенести не являющиеся критически важные изменения на более поздние инкременты;
- фактически выполняемое прототипирование на каждом этапе позволяет более гибко уточнять требования заказчика;
- риск распределяется на несколько меньших по размеру инкрементов;

- в конце каждой инкрементной поставки существует возможность пересмотреть риски, связанные с затратами и соблюдением установленного графика;
- в модели не предусмотрены итерации в рамках каждого инкремента;
- поскольку создание некоторых модулей будет завершено значительно раньше других, возникает необходимость в четко определенных интерфейсах;
- для модели необходимо хорошее планирование и проектирование;
- может возникнуть тенденция к оттягиванию решений трудных проблем на будущее с целью продемонстрировать руководству успех, достигнутый на ранних этапах разработки.

21.Общие сведения о Java. Недостатки инкрементной модели разработки программного обеспечения?

- на момент создания определенного инкремента требования упрощаются, поскольку есть возможность перенести не являющиеся критически важные изменения на более поздние инкременты;
- фактически выполняемое прототипирование на каждом этапе позволяет более гибко уточнять требования заказчика;
- риск распределяется на несколько меньших по размеру инкрементов;
- в конце каждой инкрементной поставки существует возможность пересмотреть риски, связанные с затратами и соблюдением установленного графика;
- в модели не предусмотрены итерации в рамках каждого инкремента;
- поскольку создание некоторых модулей будет завершено значительно раньше других, возникает необходимость в четко определенных интерфейсах;
- для модели необходимо хорошее планирование и проектирование;
- может возникнуть тенденция к оттягиванию решений трудных проблем на будущее с целью продемонстрировать руководству успех, достигнутый на ранних этапах разработки.

22.Общие сведения о Java. Масштабируемость языка программирования?

- это показатель того, на сколько можно изменить возможности программы, добавив не стандартный, а написанный разработчиком модуль, метод или функцию, не потеряв при этом в производительности;
- может трактоваться как способность написанных на этом языке web-приложений при той же функциональности с небольшой потерей эффективности обработать значительно возросшее количество одновременных интернет-запросов;
- может пониматься, как возможность одновременной работы многих человек над одним большим программным продуктом, путем

разбиения проекта на более мелкие подзадачи для каждого участника;
- увеличение размеров памяти, требуемой для стабильной работы программного продукта в зависимости от решаемых задач.

23. Общие сведения о Java. Что такое объектно-ориентированное программирование?

- методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования;
- так называют любой тип программирования, в котором не работают напрямую с ячейками памяти ПК;
- просто красивое понятие. Если вдуматься, оно не несет дополнительной смысловой нагрузки.

24. Общие сведения о Java. Основные принципы ООП?

- абстракция;
- инкапсуляция;
- наследование;
- полиморфизм;
- шаблон;
- перегрузка методов.

25. Общие сведения о Java. Что такое объект?

- именованная модель реальной сущности, обладающая конкретными значениями свойств и проявляющая свое поведение;
- модель информационной сущности, представляющая универсальный тип данных, состоящая из набора полей данных и методов их обработки.

26. Общие сведения о Java. Что такое класс?

- именованная модель реальной сущности, обладающая конкретными значениями свойств и проявляющая свое поведение;
- одно из возможных названий переменной;
- модель информационной сущности, представляющая универсальный тип данных, состоящая из набора полей данных и методов их обработки.

27. Общие сведения о Java. Для чего можно использовать Java?

- для разработки сайтов;
- для создания программ для ПК;
- для создания игр;
- для разработки приложений.

28. Общие сведения о Java. Что такое аргумент метода?

- переменная, в которую записывается результат работы метода;
- значение или переменная, передаваемая в метод через систему параметров для использования в его работе;
- значение, указываемое после ключевого слова "return";
- любая переменная, участвующая в работе метода;
- переменная, объявленная в теле метода.

29. Общие сведения о Java. Отметьте верное утверждение относительно языков Java и JavaScript?

- JavaScript является синонимом Java;
- их спецификации являются закрытыми;
- оба языка являются кроссплатформенными.

30. Общие сведения о Java. В каком виде распространяются скомпилированные классы Java?

- в виде байт-кода;
- бинарном;
- унарном;
- тернарном.

31. Общие сведения о Java. Сильная типизация языка программирования?

- язык не позволяет выполнять автоматические неявные преобразования с потерей точности, а также выполнять автоматические преобразования объектов, например, нельзя вычесть из строки множество;
- язык допускает выполнение множества неявных преобразований автоматически, даже если может произойти потеря точности или неконтролируемое (опасное) изменение положения области памяти, на которую ссылается переменная.

32. Общие сведения о Java. Что такое слабая типизация в языках программирования?

- язык не позволяет выполнять автоматические неявные преобразования с потерей точности, а также автоматические преобразования ссылок, например, нельзя вычесть из строки множество;
- язык допускает выполнение множества неявных преобразований автоматически, даже если может произойти потеря точности или неконтролируемое (опасное) изменение положения области памяти, на которую ссылается переменная.

33.Общие сведения о Java. Какие языки программирования являются динамически типизированными?

- конечные типы переменных и функций устанавливаются на этапе компиляции;
- все типы определяются уже во время выполнения программы.

34.Общие сведения о Java. Что такое раннее связывание метода с его определением?

- вызываемые методы определяются во время компиляции;
- метод, вызываемый для объекта, определяется во время выполнения программы.

35.Общие сведения о Java. Что такое позднее связывание метода с его определением?

- вызываемые методы определяются во время компиляции;
- метод, вызываемый для объекта, определяется во время выполнения программы.

36.Общие сведения о Java. Существуют ли указатели в Java?

- да;
- нет.

37.Общие сведения о Java. Что заменяет в Java работу с указателями?

- механизм ссылок;
- адресная арифметика.

38.Общие сведения о Java. Какими являются массивы в Java?

- динамическими;
- их размер можно определять интерактивно во время работы программы;
- статическими.

39.Общие сведения о Java. Что такое сборка мусора в Java?

- процесс восстановления заполненной памяти среды выполнения путем уничтожения неиспользуемых объектов;
- процесс, с помощью которого программы Java автоматически управляют памятью;
- процесс осуществляется с помощью ключевого слова delete.

40. Общие сведения о Java. Что такое сигнатура метода?

- это имя метода плюс параметры (причем порядок параметров имеет значение);
- в сигнатуру метода не входит возвращаемое значение, а также бросаемые им исключения;
- эта запись аналогична прототипу в C++.

41. Общие сведения о Java. Что составляет контракт метода?

- сигнатура метода в сочетании с типом возвращаемого значения;
- указание на генерируемые методом исключения;
- объявление метода.

Основы синтаксиса. Неименованные константы

42. Признак однострочного комментария?
43. Признак начала многострочного комментария?
44. Манипулятор перевода курсора при выводе на экран на новую строку?
45. Манипулятор, позволяющий выводить на экран логический (булев) тип?

46. В языке Java целые десятичные литеральные константы представляют собой:

- последовательность цифр от 0 до 7, начинающаяся с 0, например 011 или 0147;
- последовательность шестнадцатеричных цифр 0-9 и A-F, перед которой стоит 0X или 0x, например 0xffff;
- последовательность цифр от 0 до 9, не начинающуюся с 0, например 23, 2003.

47. В языке Java целые восьмеричные литеральные константы представляют собой:

- последовательность цифр от 0 до 7, начинающаяся с 0, например 011 или 0147;
- последовательность шестнадцатеричных цифр 0-9 и A-F, перед которой стоит 0X или 0x, например 0xffff;
- последовательность цифр от 0 до 9, не начинающуюся с 0, например 23, 2003.

48. В языке Java целые шестнадцатиричные литеральные константы представляют собой:

- последовательность цифр от 0 до 7, начинающаяся с 0, например 011 или 0147;
- последовательность шестнадцатеричных цифр 0-9 и A-F, перед которой стоит 0X или 0x, например 0xffff;
- последовательность цифр от 0 до 9, не начинающуюся с 0, например 23, 2003.

49. В языке Java вещественные десятичные литеральные константы представляют собой:

- последовательность цифр, разделенных точкой, не начинающуюся с 0, например 23.0 или 3.14;
- последовательность цифр, раздельных запятой, не начинающуюся с 0, например 23,0 или 3,14;
- последовательность цифр, содержащих мантиссу и показатель степени числа 10, например, 3e10, 5.12E-6.

50. Записать литеральную константу 72e02 с использованием точки для разделения целой и дробной частей.

51. Записать литеральную константу 72e-02 с использованием точки для разделения целой и дробной частей.

52. Обычно идентификаторы Index, INDEX и index обозначают:

- одну и ту же переменную;
- две различных переменных;
- три различных переменных.

53. В языке Java основное отличие переменных от именованных констант состоит в том, что:

- именованной константе присвоить значение можно только при ее объявлении, переменной – в любом месте программы;
- переменная существует в памяти машины, именованная константа не существует;
- именованную константу можно инициализировать значением при создании, переменную нельзя.

54. Массивы в языке Java могут быть:

- только одномерными;
- одномерными или двумерными;
- с любым количеством измерений.

55. Индексация массивов в языке Java начинается:

- с единицы;

- с нуля;
- с любого индекса, определяемого программистом.

56. Если одномерный массив в языке Java состоит из n элементов, то его индекс может принимать значения:

- от 0 до n-1;
- от 1 до n;
- от 0 до n.

57. С точки зрения языка Java выражение является истинным, если:

- это выражение равно 0;
- это выражение не равно 0;
- это выражение равно 1.

58. С точки зрения языка Java выражение является ложным, если:

- это выражение равно 0;
- это выражение не равно 0;
- это выражение равно 1.

59. Наименование класса, содержащего библиотеку математических функций?

60. В языке Java символьные литеральные константы (символьные литералы) представляют собой:

- одиночный символ, заключенный в апострофы, например 'w', 'g' или '7';
- последовательность символов, заключенную в двойные кавычки, например "Это строка";
- последовательность цифр, не начинающуюся с 0, например 23, 2003.

61. В языке Java строковые литеральные константы (строковые литералы) представляют собой:

- одиночный символ, заключенный в апострофы, например 'w', 'g' или '7';
- последовательность символов, заключенную в двойные кавычки, например "Это строка";
- последовательность цифр, не начинающуюся с 0, например 23, 2003.

Операции

62. Обозначение (знак) операции декремент

63.Обозначение (знак) операции инкремент
64.Знак составного присваивания (присваивание со сложением)
65.Знак составного присваивания (присваивание с умножением)
66.Знак составного присваивания (присваивание с вычитанием)
67.Знак составного присваивания (присваивание с делением)
68.Знак унарной операции отрицания
69.Знак операции сравнения (равно ли)
70.Знак операции "не равно"
71.Знак операции сравнения "больше"
72.Знак операции "меньше или равно"
73.Знак операции "меньше или равно"
74.Результат на экране

```
…
boolean n = false;
System.out.println(!n);…
```

75.Результат на экране

```
…
System.out.println(!true);
…
```

76.Результат на экране

```
…
System.out.println(3 % 2);
…
```

77.Результат на экране

```
…
System.out.println(3.5 % 2);
…
```

78.Результат на экране

```
…
System.out.println( 3.5 % (int) 2.3);
…
```

79.Результат на экране

```
…
System.out.println(1 < 2 ? true : false);
…
```

80.С помощью тернарной операции вычислить модуль переменной x

81.Результат на экране:

```
…
cout << 5 % 2 ? 0: 1;
…
```

82.Результат на экране?

```
…
System.out.println(5 % 2 == 0 ? 0: 1);
…
```

83.Результат на экране?

```
…
System.out.println(5 / 2 == 1 ? 0: 1);
…
```

84.Результат на экране?

```
…
System.out.println(5 >> 2 == 1 ? 0: 1);
…
```

85.Результат на экране?

```
…
System.out.println(5 << 2);
…
```

86.Результат на экране?

```
…
System.out.println(2 >> 2);
…
```

87.Результат на экране?

```
…
int n = 2;
System.out.println(n * (n % 2 == 1 ? 1: 0));
…
```

88.Результат на экране?

```
…
int n = 3;
System.out.println(n * (n << 2 == 1 ? 1: 0));
…
```

89.Результат на экране?

```
…
double x = 2.2;
System.out.println(x - (int) x);
…
```

90.Результат на экране?

```
…
int i = 1;
System.out.println(i++);
…
```

91.Результат на экране?

```
…
int i = 1;
System.out.println(i--);
…
```

92.Результат на экране?

```
…
int i = 1;
System.out.println(++i);
…
```

93.Результат на экране?

```
…
int i = 1;
System.out.println(--i);
…
```

94.Результат на экране?

```
…
System.out.println(7/4);
…
```

95.К каким типам операндов применима операция %?

- int
- long
- unsigned
- signed
- double
- float

96.К каким типам операндов применима операция >>?

- int
- long
- unsigned
- signed
- double
- float

97.Как называется операция, выполняющая действия над двумя переменными?

98.Как называется операция выполняющая действие только над одной переменной?

99.Результат на экране?

```
…
System.out.println(5/2);
…
```

100. Результатом выполнения операции 5 * 6 будет число

- 30;
- 30.0
- 0

101. Результатом выполнения операции 5.0 * 6 будет число

- 30;
- 30.0
- 300

102. Результатом выполнения операции 9 / 2 будет число

- 5
- 4
- 4.5

103. Результатом выполнения операции 7.0 / 2 будет число

- 3
- 4
- 3.5

104. Результатом выполнения операции 17 % 5 будет число

- 2
- 12
- 7

105. Результат на экране?

```
…
int x, y;
x = 1;
y = 1;
x = y++;
System.out.println(x);
…
```

- 0
- 1
- 2

106. Результат на экране?

```
…
int x, y;
x = 1;
y = 1;
x = y++;
System.out.println(y);
…
```

- 0
- 1
- 2

107. Для доступа к общедоступным элементам классов используется операция уточнения имени. Первым операндом операции должен быть:

- объект класса;
- ссылка на объект класса;
- переменная перечислимого типа.

Типы

108. тип целый
109. тип символьный
110. тип логический
111. тип с плавающей точкой
112. тип с плавающей точкой удвоенной точности
113. тип знаковый
114. тип беззнаковый
115. тип длинный
116. тип короткий

117. В языке Java базовый тип данных «char» предназначен для хранения:

- целых положительных чисел или символов;
- вещественных чисел;
- только символов.

118. В языке Java базовый тип данных «int» предназначен для хранения:

- символов;
- положительных и отрицательных целых чисел;
- вещественных чисел.

119. В языке Java базовый тип данных «double» предназначен для хранения:

- символов;
- вещественных чисел;
- целых чисел.

Объявления

120. Выбрать правильный формат объявления переменной:

- тип ИмяПеременной;
- ИмяПеременной.

121. Выбрать правильный формат объявления именованной константы:

- final тип ИмяПеременной;
- final ИмяПеременной;
- final модификатор тип ИмяПеременной
- final ИмяПеременной

122. Объявить целочисленный массив A размерностью n элементов.
123. Объявить строку str и проинициализировать ее словом "Java";
124. Объявить массив A чисел с плавающей точкой размерностью n;

125. Результат на экране?

```
enum Colors {
   YELLOW,
   WHITE,
   ORANGE
}
public class MyClass {
   public static void main(String args[]) {
```

```
        System.out.println(Colors.YELLOW);
    }
}
```

126. Результат на экране?

```
enum Colors {
    YELLOW,
    WHITE,
    ORANGE
}
public class MyClass {
    public static void main(String args[]) {
        System.out.println(Colors.YELLOW.ordinal());
    }
}
```

127. Результат на экране?

```
enum Colors {
    YELLOW,
    WHITE;
    static int get() {
        return YELLOW.ordinal() + WHITE.ordinal();
    }
}
public class MyClass {
    public static void main(String args[]) {
        System.out.println(Colors.get());
    }
}
```

128. Объявить строку str и проинициализировать ее массивом символов, которые составляют литерал «Hello».

129. Символ, который из выражения создает инструкцию?

130. Пустая инструкция?

Операторы управления программой

131. Результат на экране?

```
…
int i = 1;
switch(i) {
    case 1: System.out.print("один"); break;
    case 2: System.out.print("два"); break;
```

```
   case 3: System.out.print("три"); break;
}
…
```

132. Результат на экране?

```
…
int i = 1;
switch(i) {
   case 1: System.out.print("один");
   case 2: System.out.print("два"); break;
   case 3: System.out.print("три"); break;
}
…
```

133. Результат на экране?

```
…
switch(i) {
   case 1: System.out.print("один"); break;
   case 2: System.out.print("два");
   case 3: System.out.print("три"); break;
}
…
```

134. Результат на экране?

```
…
int i = 9;
switch(i) {
   case 9: System.out.print("сентябрь");
   case 10: System.out.print("октябрь");
   case 11: System.out.print("ноябрь");
   default: System.out.print(false);
}
…
```

135. Результат на экране?

```
…
int i = 10;
switch(i) {
   case 9: System.out.print("сентябрь");
   case 10: System.out.print("октябрь"); break;
   case 11: System.out.print("ноябрь");
   default: System.out.print(false);
}
…
```

136. Результат на экране?

```
…
int a, b = 5;
if (b % 2 == 1) a = b / 2;
else a = 3 * b;
System.out.print(a);
…
```

137. Результат на экране?

```
…
int a = 6, b = 2;
if (!(a % b == 1)) System.out.print(a * b);
System.out.print("Ничего");
…
```

138. Результат на экране?

```
…
int a = 5, b = 2;
if (((float) a) / b – a / b) > 0.7) System.out.print(a / 2);
else System.out.print(2 * b);
…
```

139. Результат на экране?

```
…
int a = 5, b = 2;
if ((a / b) % 2 == 0) System.out.print(a * a);
else System.out.print(3 * b);
…
```

140. Результат на экране?

```
…
int a = 5, b = 2, c = 3;
if (a > b & a > c) System.out.print(a);
else if (b > a & b > c) System.out.print(b);
else System.out.print(c);
…
```

141. Результат на экране?

```
…
int a = 1, b = 5, c = -3;
if (a < b & a < c) System.out.print(a);
else if (b < a & b < c) System.out.print(b);
else System.out.print(c);
…
```

142. Результат на экране?

```
…
int a = 1, b = 5, c = -3;
System.out.print( (a < b & a < c) ? a : (b < a & b < c) ? b: c);
…
```

143. Результат на экране?

```
…
int a = 1, b = 5, c = -3;
System.out.print( ((a > b & a > c) ? a : ((b > a & b > c) ? b: c)) );
…
```

144. Выбрать правильный формат оператора for()

145. Выбрать правильный формат оператора while()

146. Выбрать правильный формат оператора do while()

147. Указать конструкцию «бесконечный цикл»

148. Какое значение получит переменная y в результате выполнения участка программы

```
int x, y;
x = 1;
y = 1;
while (x < 1) {
x = x + 1;
y = y + 1;
}
```

149. Какое значение получит переменная y в результате выполнения участка программы

```
int x, y;
x = 1; y = 1;
while (x < 2) { x = x + 1; y = y + 1; }
```

150. Какое значение получит переменная y в результате выполнения участка программы

```
int x, y; x = 1; y = 1;
do { x = x + 1; y = y + 1; } while (x < 1);
```

151. Какое значение получит переменная j в результате выполнения участка программы

```
int i, j; i = 1; j = 1;
do {
   i++; j+=1;
} while (i <= 2);
```

152. Какое значение получит переменная b в результате выполнения участка программы

```
int a, b;
for(a = 0, b = 1; a < 3; a++) {
   b = b * 2;
}
```

153. Какое значение получит переменная a в результате выполнения участка программы

```
int a, b;
for(a = 0, b = 1; a < 3; a++) {
   b = b * 2;
}
```

154. Какое значение получит переменная b в результате выполнения участка программы

```
int a, b;
for(a = 0, b = 0; a < 3; a++) {
   b = b + 2;
}
```

155. Какое значение получит переменная b в результате выполнения участка программы

```
int a, b;
for(a = 0, b = 0; a < 3; b+=1, a++);
```

156. Какое значение получит переменная b в результате выполнения участка программы

```
int a, b;
for(a = 0, b = 1; a < 3; b *= 5, a++);
```

157. Выбрать правильную последовательность работы оператора

for(секция1; секция2; секция3) инструкция;

158. Оператор break служит для того, чтобы:

- прекратить выполнение содержащего его ближайшего цикла while, do … while или for;
- прекратить выполнение содержащего его условного оператора switch;
- передать управление на следующий за прерванным оператор;
- немедленно завершить программу.

Методы

159. Назначение метода

```
int function(int b) {
   return b % 2;
}
```

- определяет четность/нечетность числа b;
- определяет остаток от целочисленного деления b на 2;
- возвращает в качестве значения половину b.

160. Назначение метода

```
bool function(int b) {
   return b % 2;
}
```

- определяет является ли число b четным;
- определяет является ли число b нечетным;
- возвращает остаток от целочисленного деления b на 2.

161. Назначение метода

```
int function(int a) {
   return ! (a % 5);
}
```

- определяет является ли число a кратным 5;
- определяет является ли число a нечетным;
- возвращает отрицательную величину от целочисленного деления a на 5.

162. Назначение метода

```
int function(int a) {
   return (a >> 2);
}
```

- делит число a на 2;
- делит число a на 4;
- выполняет побитовый сдвиг в двоичном представлении a вправо на 2 бита;
- выполняет побитовый сдвиг в двоичном представлении a вправо на 4 бита.

163. Назначение метода

```
int function(int a) {
   return (a << 2);
}
```

- умножает число a на 2;
- умножает число a на 4;
- выполняет побитовый сдвиг в двоичном представлении a влево на 2 бита;
- выполняет побитовый сдвиг в двоичном представлении a влево на 4 бита.

164. Результат на экране

```
public class MyClass {
   static boolean function(double x, double y) {
      return 0 < x & 0 < y;
   }
   public static void main(String[] args) {
      System.out.print( function(1, 1) );
   }
}
```

165. Результат на экране

```
public class MyClass {
   static boolean function(double x, double y) {
      return x < 0 & 0 < y;
   }
   public static void main(String[] args) {
      System.out.print( function(1, 1) );
   }
}
```

166. Результат на экране

```
public class MyClass {
   static boolean function(double x, double y) {
      return x < -3 & y < 0;
   }
   public static void main(String[] args) {
      System.out.print( function(-5, -1) );
   }
}
```

167. Результат на экране

```
public class MyClass {
   static int function(int a, int b) {
      int result = 0;
      for(int i = a; i <= b; result += i, i++);
      return result;
   }
   public static void main(String[] args) {
```

```
        System.out.print( function(1, 3) );
    }
}
```

168. Результат на экране

```
public class MyClass {
    static int function(int a, int b) {
        int result = 1;
        for(int i = a; i <= b; i++)
            result *= i;
        return result;
    }
    public static void main(String args[]) {
        System.out.println(function(1, 3));
    }
}
```

169. Результат на экране

```
public class MyClass {
    static int function(int a, int b) {
        int result = 0;
        for(int i = a; i <= b; result += i * i, i++);
        return result;
    }
    public static void main(String args[]) {
        System.out.println(function(3, 4));
    }
}
```

170. Результат на экране

```
public class MyClass {
    static int function(int a, int b) {
        int result = 1;
        for(int i = a; i <= b; i++)
            result *= i * i;
        return result;
    }
    public static void main(String args[]) {
        System.out.println(function(2, 3));
    }
}
```

171. Результат на экране

```
public class Main {
    double f(double x) {return x;}
```

```
   boolean function(double a, double b) {
      double delta = (b - a) / 10;
      Main obj = new Main();
      for(double x = a; x < b; x += delta)
         if (obj.f(x) > obj.f(x+delta)) return false;
      return true;
   }
   public static void main(String args[]) {
      Main obj = new Main();
      System.out.println(obj.function(2, 3));
   }
}
```

172. Результат на экране

```
public class Main {
   double f(double x) {return x*x*x;}
   boolean function(double a, double b) {
      double delta = (b - a) / 10;
      Main obj = new Main();
      for(double x = a; x < b; x += delta)
         if (obj.f(x) > obj.f(x+delta)) return false;
      return true;
   }
   public static void main(String args[]) {
      Main obj = new Main();
      System.out.println(obj.function(2, 3));
   }
}
```

173. Результат на экране

```
public class Main {
   double f(double x) {return x*x;}
   boolean function(double a, double b) {
      double delta = (b - a) / 10;
      Main obj = new Main();
      for(double x = a; x < b; x += delta)
         if (obj.f(x) > obj.f(x+delta)) return false;
      return true;
   }
   public static void main(String args[]) {
      Main obj = new Main();
      System.out.println(obj.function(-2, -1));
   }
}
```

174. Указать назначение метода

```
…
static boolean function(double x) {
   return x > 0;
}
…
```

- определяет, является ли x положительным;
- определяет, является ли x отрицательным;
- ничего не делает.

175. Указать назначение метода

```
…
static boolean function(double x) {
   return !(x >= 0);
}
…
```

- определяет, является ли x положительным;
- определяет, является ли x отрицательным;
- ничего не делает.

176. Указать назначение метода

```
…
static boolean function(double a, double x) {
   return –a < x & x < a;
}
…
```

- определяет, принадлежит ли x интервалу (-a,a) (]-a,a[);
- определяет, принадлежит ли x внешности (дополнению) интервала (-a,a) (]-a,a[);
- ничего не делает.

177. Указать назначение метода

```
…
static boolean function(double a, double x) {
   return !(–a < x & x < a);
}
…
```

- определяет, принадлежит ли x интервалу (-a,a) (]-a,a[);
- определяет, принадлежит ли x внешности (дополнению) интервала (-a,a) (]-a,a[);
- ничего не делает.

178. Результат на экране

```
public class Main {
```

```
    boolean function(boolean a, boolean b) {
        return a | b;
    }
    public static void main(String args[]) {
        Main obj = new Main();
        System.out.println(obj.function(false, true));
    }
}
```

179. Результат на экране

```
public class Main {
    char function(char a, char b) {
        return a > b ? a : b;
    }
    public static void main(String args[]) {
        Main obj = new Main();
        System.out.println(obj.function('a', 'b'));
    }
}
```

180. Результат на экране

```
public class Main {
    char function(char a, char b) {
        return a > b ? a : b;
    }
    public static void main(String args[]) {
        Main obj = new Main();
        System.out.println('a' - obj.function('a', 'b'));
    }
}
```

181. Результат на экране

```
public class Main {
    int function(char a, char b) {
        return a - b;
    }
    public static void main(String args[]) {
        Main obj = new Main();
        System.out.println(obj.function((char) 5, (char) 0));
    }
}
```

182. Результат на экране

```
public class Main {
    int function(double a, float b, int c) {
```

```
        return (int) (a + b + c);
    }
    public static void main(String args[]) {
        Main obj = new Main();
        System.out.println(obj.function(2.5, 1.7f, 1));
    }
}
```

183. Результат на экране

```
public class Main {
    int function(int a, float b, int c) {
        return (int) (a + b + c);
    }
    public static void main(String args[]) {
        Main obj = new Main();
        System.out.println(obj.function((int) 2.5, 1.7f, 1));
    }
}
```

184. Результат на экране

```
public class Main {
    static double function(int a, float b, int c) {
        return (int) (a + b + c);
    }
    public static void main(String args[]) {
        System.out.println(function((int) 2.5, 1.7f, 1));
    }
}
```

185. Результат на экране

```
public class Main {
    static double function(int a, float b, int c) {
        return a * b * c;
    }
    public static void main(String args[]) {
        System.out.println(function((int) 2.3, 1.7f, 1));
    }
}
```

186. Какое значение должен возвращать метод main()?

187. Результат на экране

```
class User {
    static double function(double x) {
        if (x < 0) x = -x;
```

```
        return x;
    }
}
public class Main {
    public static void main(String args[]) {
        System.out.println(User.function(-2.5));
    }
}
```

188. Результат на экране

```
class User {
    static double function(double x) {
        return x < 0 ? -x : x;
    }
}
public class Main {
    public static void main(String args[]) {
        System.out.println(User.function(-2.5));
    }
}
```

189. Результат на экране

```
class User {
    static double function(double x) {
        return x - (int) x;
    }
}
public class Main {
    public static void main(String args[]) {
        System.out.printf("%5.1f", User.function(2.5));
    }
}
```

190. Результат на экране

```
class User {
    static double function(double x) {
        return (int) x;
    }
}
public class Main {
    public static void main(String args[]) {
        System.out.println(User.function(2.5));
    }
}
```

191. Результат на экране

```
class User {
   static String function(int x) {
      return (new Double(x)).toString();
   }
}
public class Main {
   public static void main(String args[]) {
      System.out.println(User.function((int) 2.5));
   }
}
```

192. Результат на экране

```
class User {
   static int function(double x) {
      return (int) x;
   }
}
public class Main {
   public static void main(String args[]) {
      System.out.println(User.function(2.5));
   }
}
```

193. Результат на экране

```
class User {
   static double function(int x) {
      return x;
   }
   static String function(double x) {
      return String.format("%5.1f", x - (int) x);
   }
}
public class Main {
   public static void main(String args[]) {
      System.out.println(User.function(2.5));
   }
}
```

194. Результат на экране

```
class User {
   static double function(int x) {
      return x;
   }
   static String function(double x) {
```

```
        return String.format("%5.1f", x - (int) x);
    }
}
public class Main {
    public static void main(String args[]) {
        System.out.println(User.function(2));
    }
}
```

195. Результат на экране

```
class User {
    static int function(int x) {
        return 1;
    }
    static double function(double x) {
        if (x < 0) x = -x;
        return x;
    }
}
public class Main {
    public static void main(String args[]) {
        System.out.println(User.function(2));
    }
}
```

196. Результат на экране

```
class User {
    static String function(double a, float b, int c) {
        return String.format("%d", (int) (a + b + c));
    }
    static String function(int a, float b, int c) {
        return String.format("%5.1f", a + b + c);
    }
}
public class Main {
    public static void main(String args[]) {
        System.out.printf(User.function(2, 1.7f, 1));
    }
}
```

197. Результат на экране

```
import java.util.*;
class User {
    static boolean function(double a, double x) {
        return !(-a < x & x < a);
```

```
    }
    static int function(boolean a, boolean b) {
        return Boolean.compare(new Boolean(a), new Boolean(b));
    }
}
public class Main {
    public static void main(String args[]) {
        System.out.print(User.function(false, true));
    }
}
```

198. Результат на экране

```
import java.util.*;
class User {
    static boolean function(double a, double x) {
        return !(-a < x & x < a);
    }
    static int function(boolean a, boolean b) {
        return (new Boolean(a)).compareTo(new Boolean(b));
    }
}
public class Main {
    public static void main(String args[]) {
        System.out.print(User.function(false, true));
    }
}
```

199. Результат на экране

```
class User {
    static double function(int n) {
        int result = 0;
        for(int i =0; i < n; i++)
            result = result + i;
        return result;
    }
}
public class Main {
    public static void main(String args[]) {
        System.out.print(User.function(3));
    }
}
```

200. Результат на экране

```
class User {
    static double function(int n) {
```

```
        int result = 0;
        for(int i = 0; i < n; i += 2)
            result = result + i;
        return result;
    }
}
public class Main {
    public static void main(String args[]) {
        System.out.print(User.function(5));
    }
}
```

201. Результат на экране

```
class User {
    static double function(int n) {
        int result = 1;
        for(int i =0; i < n; i+=1, result = result * i);
        return result;
    }
}
public class Main {
    public static void main(String args[]) {
        System.out.print(User.function(3));
    }
}
```

202. Результат на экране

```
class User {
    static double function(int n) {
        int result = 1;
        for(int i = 1; i < n; result = result * i, i += 2);
        return result;
    }
}
public class Main {
    public static void main(String args[]) {
        System.out.print(User.function(5));
    }
}
```

203. Результат на экране

```
class User {
    static double function(int n) {
        int result = 1;
        for(int i = 1; i < n; result = result + i, i += 2);
```

```
        return result;
    }
}
public class Main {
    public static void main(String args[]) {
        System.out.print(User.function(5));
    }
}
```

204. Результат на экране

```
class User {
    static double function(double x, int n) {
        double result = 5;
        for(int i =1; i < n; result = result * x, i++);
        return result;
    }
}
public class Main {
    public static void main(String args[]) {
        System.out.print(User.function(5, 2));
    }
}
```

205. Результат на экране

```
class User {
    static long function(long n) {
        return n == 0 ? 1 : n * User.function(n - 1);
    }
}
public class Main {
    public static void main(String args[]) {
        System.out.print(User.function(3));
    }
}
```

206. Результат на экране

```
class User {
    static long function(long n) {
        return n == 0 ? 0 : n + User.function(n - 1);
    }
}
public class MyClass {
    public static void main(String args[]) {
        System.out.print(User.function(3));
    }
```

```
}
```

207. Результат на экране

```
class User {
   static long function(long n) {
      return (n == 0 || n==1) ? 1 : n * function(n - 2);
   }
}
public class MyClass {
   public static void main(String args[]) {
      System.out.print(User.function(3));
   }
}
```

208. Результат на экране

```
class User {
   static long function(long n) {
      return (n == 0 || n==1) ? 1 : n * function(n - 2);
   }
}
public class MyClass {
   public static void main(String args[]) {
      System.out.print(User.function(3));
   }
}
```

209. Результат на экране

```
class User {
   static double function(double x, int n) {
      return n == 0 ? 1 : x * function(x, n - 1);
   }
}
public class MyClass {
   public static void main(String args[]) {
      System.out.print(User.function(2, 2));
   }
}
```

210. Результат на экране

```
class User {
   long function(int n) {
      return n == 0 ? 0 : 1 + this.function(n - 1);
   }
}
```

```
public class MyClass {
   public static void main(String args[]) {
      User obj = new User();
      System.out.print(obj.function(5));
   }
}
```

211. Результат на экране

```
class User {
   long function(int n) {
      return n == 0 ? 1 : 2 * function(n - 1);
   }
}
public class MyClass {
   public static void main(String args[]) {
      User obj = new User();
      System.out.print(obj.function(3));
   }
}
```

212. Результат на экране

```
class User {
   int function(int[] a, int n) {
      return n == 0 ? a[0] : a[n] + this.function(a, n - 1);
   }
}
public class MyClass {
   public static void main(String args[]) {
      int a[] = {1, 2, 3};
      User obj = new User();
      System.out.print(obj.function(a, a.length - 1));
   }
}
```

213. Результат на экране

```
class User {
   int function(int[] a, int n) {
      return n == 0 ? a[0] : a[n] * function(a, n - 1);
   }
}
public class MyClass {
   public static void main(String args[]) {
      int a[] = {1, 2, 3};
      User obj = new User();
```

```
        System.out.print(obj.function(a, a.length - 1));
    }
}
```

214. Результат на экране

```
class User {
    long function(long n) {
        return (n == 0 || n==1) ? 1 : n + function(n - 2);
    }
}
public class MyClass {
    public static void main(String args[]) {
        User obj = new User();
        System.out.print(obj.function(5));
    }
}
```

215. Результат на экране

```
class User {
    long function(long n) {
        return (n == 0 || n==1) ? 0 : n + function(n - 2);
    }
}
public class MyClass {
    public static void main(String args[]) {
        User obj = new User();
        System.out.print(obj.function(5));
    }
}
```

216. Указать назначение метода:

```
…
static void function(double[] a, int n) {
    for(int i = 0; i < n; i++) {
        a[i] = Math.random() * 10;
    }
}
…
```

- заполняет массив случайными целыми числами в диапазоне от 0 до 10 включительно;
- заполняет массив случайными положительными целыми числами;
- заполняет массив как положительными, так и отрицательными целыми числами.

217. Указать назначение метода:

```
…
static void function(double[] a, int n) {
   srand(time(0));
   for(int i = 0; i < n; i++) {
      a[i] = 5 - Math.random() * 10;
   }
}
…
```

- заполняет массив случайными целыми числами в диапазоне от -5 до 5 включительно;
- заполняет массив случайными только положительными целыми числами;
- заполняет массив как положительными, так и отрицательными целыми числами.

218. Результат на экране:

```
class User {
   void function(int[] a) {
      for(int i = a.length - 1; i >= 0; i--) {
         System.out.print(a[i] + " ");
      }
   }
}
public class MyClass {
   public static void main(String args[]) {
      int[] a = {1, 2, 3};
      User obj = new User();
      obj.function(a);
   }
}
```

219. Результат на экране:

```
class User {
   int function(int[] a) {
      int result = a[0];
      for(int i = 1; i < a.length; i++) {
         if (result > a[i]) result = a[i];
      }
      return result;
   }
}
public class MyClass {
   public static void main(String args[]) {
```

```
        int[] a = {1, 2, 3};
        User obj = new User();
        System.out.print(obj.function(a));
    }
}
```

220. Результат на экране:

```
class User {
    int function(int[] a) {
        int j = 0;
        for(int i = 1; i < a.length; i++) {
            if (a[j] > a[i]) j = i;
        }
        return j;
    }
}
public class MyClass {
    public static void main(String args[]) {
        int[] a = {1, 2, 3};
        User obj = new User();
        System.out.print(obj.function(a));
    }
}
```

221. Результат на экране:

```
class User {
    int function(int[] a) {
        int result = a[0];
        for(int i = 1; i < a.length; i++) {
        if (result < a[i]) result = a[i];
        }
        return result;
    }
}
public class MyClass {
    public static void main(String args[]) {
        int[] a = {1, 2, 3};
        User obj = new User();
        System.out.print(obj.function(a));
    }
}
```

222. Результат на экране:

```
class User {
    static boolean then(int a, int b) {
```

```
        return a < b;
    }
    int function(int[] a) {
        int result = a[0];
        for(int i = 1; i < a.length; i++) {
        if (then(result, a[i])) result = a[i];
        }
        return result;
    }
}
public class MyClass {
    public static void main(String args[]) {
        int[] a = {1, 2, 3};
        User obj = new User();
        System.out.print(obj.function(a));
    }
}
```

223. Результат на экране:

```
class User {
    boolean then(int a, int b) {
        return a > b;
    }
    int function(int[] a) {
        int result = a[0];
        for(int i = 1; i < a.length; i += 2) {
        if (this.then(result, a[i])) result = a[i];
        }
        return result;
    }
}
public class MyClass {
    public static void main(String args[]) {
        int[] a = {1, 2, 3};
        User obj = new User();
        System.out.print(obj.function(a));
    }
}
```

224. Результат на экране:

```
class User {
    int function(int[] a) {
        int j = 0;
        for(int i = 2; i < a.length; i+=2) {
            if (a[j] < a[i]) j = i;
```

```
        }
        return j;
    }
}
public class MyClass {
    public static void main(String args[]) {
        int[] a = {1, 4, 2, 3};
        User obj = new User();
        System.out.print(obj.function(a));
    }
}
```

225. Результат на экране:

```
class User {
    int function(int[] a) {
        int i;
        for(i = 0; (i < a.length) & (a[i] > 0); i++);
            for(int j = i; j < a.length; j += 1) {
                if ((a[j] < a[i]) & (a[j] < 0)) i = j;
        }
        return i;
    }
}
public class MyClass {
    public static void main(String args[]) {
        int a[] = {1, 4, -1, 2, -5};
        User obj = new User();
        System.out.print(obj.function(a));
    }
}
```

226. Результат на экране:

```
class User {
    int function(int[] a) {
        int i;
        for(i = 0; (i < a.length) & (a[i] < 0); i++);
            for(int j = i; j < a.length; j += 1) {
                if ((a[j] > a[i]) & (a[j] > 0)) i = j;
        }
        return i;
    }
}
public class MyClass {
    public static void main(String args[]) {
        int a[] = {1, 4, -1, 2, -5};
```

```
        User obj = new User();
        System.out.print(obj.function(a));
    }
}
```

227. Результат на экране:

```
class User {
    int function(int[] a) {
        int i;
        for(i = 0; (i < a.length) & (a[i] != 0); i++);
        return i;
    }
}
public class MyClass {
    public static void main(String args[]) {
        int a[] = {1, 0, -1, 0, -5};
        User obj = new User();
        System.out.print(obj.function(a));
    }
}
```

228. Результат на экране:

```
class User {
    int function(int[] a) {
        int i;
        for(i = a.length - 1; (i >= 0) & (a[i] != 0); i--);
        return i;
    }
}
public class MyClass {
    public static void main(String args[]) {
        int a[] = {1, 0, -1, 0, -5};
        User obj = new User();
        System.out.print(obj.function(a));
    }
}
```

229. Результат на экране:

```
class User {
    int function(int[] a) {
        int i;
        for(i = 0; (i < a.length) && (a[i] % 4 != 1); i++);
            for(int j = i; j < a.length; j+=1) {
                if ((a[j] > a[i]) && (a[j] % 4 == 1)) i = j;
        }
```

```
        return i;
    }
}
public class MyClass {
    public static void main(String args[]) {
        int a[] = {0, 9, -1, 2, 5};
        User obj = new User();
        System.out.print(obj.function(a));
    }
}
```

230. Результат на экране:

```
class User {
    int function(int[] a) {
        int i;
        for(i = a.length - 1; (i >= 0) & (a[i] >= 0); i--);
        return i;
    }
}
public class MyClass {
    public static void main(String args[]) {
        int a[] = {1, 0, -1, 0, -5};
        User obj = new User();
        System.out.print(obj.function(a));
    }
}
```

231. Результат на экране:

```
class User {
    int function(int[] a) {
        int i;
        for(i = a.length - 1; (i >= 0) & (a[i] % 2 == 0); i--);
        return i;
    }
}
public class MyClass {
    public static void main(String args[]) {
        int a[] = {1, 0, 1, 3, 2};
        User obj = new User();
        System.out.print(obj.function(a));
    }
}
```

232. Результат на экране:

```
class User {
   int function(int[] a) {
      int i;
      for(i = 0; (i < a.length) & (a[i] % 2 != 0); i++);
      return i;
   }
}
public class MyClass {
   public static void main(String args[]) {
      int a[] = {1, 0, 2, 0, 3};
      User obj = new User();
      System.out.print(obj.function(a));
   }
}
```

233. Результат на экране:

```
public class MyClass {
   int function(int[] a) {
      int s, i = 0;
      for(s = 0; i < a.length; s += a[i], i++);
      return s;
   }
   public static void main(String args[]) {
      int a[] = {1, 0, 2, 0, 3};
      MyClass obj = new MyClass();
      System.out.print(obj.function(a));
   }
}
```

234. Результат на экране:

```
public class MyClass {
   int function(int[] a) {
      int s, i = 0;
      for(s = 0; i < a.length; s *= a[i], i++);
      return s;
   }
   public static void main(String args[]) {
      int a[] = {1, 0, 2, 0, 3};
      MyClass obj = new MyClass();
      System.out.print(obj.function(a));
   }
}
```

235. Результат на экране:

```
public class MyClass {
   boolean function(int[] a) {
      int i, n = a.length;
      for(i = 0; i < n & a[i] >= 0; i++);
      return (n - 1 - i) == 0 ? false: true;
   }
   public static void main(String args[]) {
      int a[] = {1, 0, 2, 0, 3};
      MyClass obj = new MyClass();
      System.out.print(obj.function(a));
   }
}
```

236. Результат на экране:

```
public class MyClass {
   boolean function(int[] a) {
      int i, n = a.length;
      for(i = 0; i < n & a[i] % 2 != 0; i++);
      return (n - 1 - i) == 0 ? false: true;
   }
   public static void main(String args[]) {
      int a[] = {1, 0, 2, 0, 3};
      MyClass obj = new MyClass();
      System.out.print(obj.function(a));
   }
}
```

237. Результат на экране:

```
public class MyClass {
   int function(int[] a) {
      int n = a.length;
      int s = 0;
      for(int i = 0; i < n; i++)
         if (a[i] % 2 == 1) s += 1;
      return s;
   }
   public static void main(String args[]) {
      int a[] = {1, 1, 2, 6, 3};
      MyClass obj = new MyClass();
      System.out.print(obj.function(a));
   }
}
```

238. Результат на экране:

```
public class MyClass {
   int function(int[] a) {
      int n = a.length;
      int s = 0;
      for(int i = 0; i < n; i++)
         if (a[i] > 0) s += 1;
      return s;
   }
   public static void main(String args[]) {
      int a[] = {1, -1, 2, -6, 3};
      MyClass obj = new MyClass();
      System.out.print(obj.function(a));
   }
}
```

239. Результат на экране:

```
import java.util.*;
public class MyClass {
   void function(int[] a) {
      int n = a.length;
      int temp = a[0];
      for(int i = 0; i < n - 1; i++) {
         a[i] = a[i + 1];
      }
      a[n - 1] = temp;
   }
   public static void main(String args[]) {
      int a[] = {1, -1, 2, -6, 3};
      (new MyClass()).function(a);
      System.out.print(Arrays.toString(a));
   }
}
```

240. Результат на экране:

```
import java.util.*;
public class MyClass {
   void function(int[] a) {
      int temp = a[a.length - 1];
      for(int i = a.length - 1; i > 0; i--) {
         a[i] = a[i - 1];
      }
      a[0] = temp;
   }
```

```
    public static void main(String args[]) {
        int a[] = {1, -1, 2, -6, 3};
        (new MyClass()).function(a);
        System.out.print(Arrays.toString(a));
    }
}
```

241. Результат на экране:

```
import java.util.*;
public class MyClass {
    void function(int[] a) {
        int n = a.length;
        for(int i = 0; i < n/2; i++) {
            int b = a[i]; a[i] = a[n - 1 - i]; a[n - 1 - i] = b;
        }
    }
    public static void main(String args[]) {
        int a[] = {1, -1, 2, -6, 3};
        (new MyClass()).function(a);
        System.out.print(Arrays.toString(a));
    }
}
```

242. Результат на экране:

```
import java.util.*;
public class MyClass {
    void function(int[] a) {
        int n = a.length;
        for(int i = 0; i < n / 4; i++) {
            int b = a[i];
            a[i] = a[n / 2 - 1- i];
            a[n / 2 - 1- i] = b;
        }
    }
    public static void main(String args[]) {
        int a[] = {1, -1, 2, -6, 3};
        (new MyClass()).function(a);
        System.out.print(Arrays.toString(a));
    }
}
```

243. Результат на экране:

```
import java.util.*;
public class MyClass {
    static void function(int[] a) {
```

```
        int n = a.length;
        for(int i = n / 2; i < 3*n / 4; i++) {
            int b = a[i];
            a[i] = a[n - 1- (i - n / 2)];
            a[n - 1- (i - n / 2)] = b;
        }
    }
    public static void main(String args[]) {
        int a[] = {1, -1, 2, -6, 3};
        function(a);
        System.out.print(Arrays.toString(a));
    }
}
```

244. Результат на экране:

```
import java.util.*;
public class MyClass {
    static void function(int[] a) {
        for(int i = 0; i < a.length; i++)
            a[i] = a[i] * a[i];
    }
    public static void main(String args[]) {
        int a[] = {1, -1, 2};
        function(a);
        System.out.print(Arrays.toString(a));
    }
}
```

245. Результат на экране:

```
import java.util.*;
public class MyClass {
    static void function(int[] a) {
        for(int i = 0; i < a.length; i+=2)
            a[i] = 2;
    }
    public static void main(String args[]) {
        int a[] = {1, -1, 2, -6, 3};
        function(a);
        System.out.print(Arrays.toString(a));
    }
}
```

246. Результат на экране:

```
import java.util.*;
public class MyClass {
```

```
    static void function(int[] a) {
        for(int i = 1; i < a.length; i+=2)
            a[i] = 1;
    }
    public static void main(String args[]) {
        int a[] = {1, -1, 2, -6, 3};
        function(a);
        System.out.print(Arrays.toString(a));
    }
}
```

247. Результат на экране:

```
public class MyClass {
    int function(String str) {
        int n = 0;
        char[] a = str.toCharArray();
        for(int i = 0; i < a.length; i++) {
            if(a[i]>= 'a' & a[i]<= 'z') n++;
        }
        return n;
    }
    public static void main(String args[]) {
        String str = "WorLd";
        System.out.print((new MyClass()).function(str));
    }
}
```

248. Результат на экране:

```
public class MyClass {
    int function(String str) {
        int n = 0;
        char[] a = str.toCharArray();
        for(int i = 0; i < a.length; i++) {
            if(a[i]>= 'A' & a[i]<= 'Z') n++;
        }
        return n;
    }
    public static void main(String args[]) {
        String str = "WorLd";
        System.out.print((new MyClass()).function(str));
    }
}
```

249. Результат на экране:

```
public class MyClass {
   int function(String str) {
      int n = 0;
      char[] a = str.toCharArray();
      for(int i = 0; i < a.length; i++) {
         if(a[i]>= '0' & a[i]<= '9') n++;
      }
      return n;
   }
   public static void main(String args[]) {
      String str = "1, 2, 3, 4, 5";
      System.out.print((new MyClass()).function(str));
   }
}
```

250. Результат на экране:

```
public class MyClass {
   int function(String str) {
      int n = 0;
      char[] a = str.toCharArray();
      for(int i = 0; i < a.length; i++) {
         if(a[i] == ' ') n++;
      }
      return n;
   }
   public static void main(String args[]) {
      String str = "1, 2, 3, 4, 5";
      System.out.print((new MyClass()).function(str));
   }
}
```

251. Результат на экране:

```
public class MyClass {
   int function(String str) {
      int n = 0;
      char[] a = str.toCharArray();
      for(int i = 0; i < a.length; i++) {
         if(a[i] == ',') n++;
      }
      return n;
   }
   public static void main(String args[]) {
      String str = "1, 2, 3, 4, 5";
```

```
        System.out.print((new MyClass()).function(str));
    }
}
```

252. Результат на экране:

```
public class MyClass {
    int function(String str) {
        return str.indexOf(' ');
    }
    public static void main(String args[]) {
        String str = "1, 2, 3, 4, 5";
        System.out.print((new MyClass()).function(str));
    }
}
```

253. Вид результирующей матрицы в памяти после вызова метода:

```
import java.util.*;
class User {
    void function(int[][] b) {
        for(int i = 0; i < b.length; i += 1)
            b[i][i] = 1;
    }
    static void f(int[][] b) {
        for(int i = 0; i < b.length; i += 1)
            System.out.println(Arrays.toString(b[i])
                                .replaceAll("\\[|\\]|,", ""));
    }
}
public class MyClass {
    public static void main(String args[]) {
        int[][] a = new int[2][2];
        (new User()).function(a);
        User.f(a);
    }
}
```

- 0 1
  1 0
- 1 0
  0 1
- 1 1
  0 1

254. Вид результирующей матрицы в памяти после вызова метода:

```
import java.util.*;
class User {
```

```
    void function(int[][] b) {
        for(int i = 0; i < b.length; i+=1)
            b[i][b.length - 1 - i] = 1;
    }
    static void f(int[][] b) {
        for(int i = 0; i < b.length; i += 1)
            System.out.println(Arrays.toString(b[i])
                                .replaceAll("\\[|\\]|,", ""));
    }
}
public class MyClass {
    public static void main(String args[]) {
        int[][] a = new int[2][2];
        (new User()).function(a);
        User.f(a);
    }
}
```

- 1 1
  0 1
- 1 0
  0 1
- 0 1
  1 0

255. Результат на экране:

```
import java.util.*;
class User {
    int function(int[][] b) {
        int r = b[0][0];
        for(int i = 0; i < b.length; i++) {
            for(int j = 0; j < b[i].length; j++) {
                if (r < b[i][j]) r = b[i][j];
            }
        }
        return r;
    }
}
public class MyClass {
    public static void main(String args[]) {
        int[][] a = {{0, 1}, {2, 3}};
        System.out.println((new User()).function(a));
    }
}
```

256. Вид результирующей матрицы в памяти после вызова метода:

```
import java.util.*;
class User {
   void function(int[][] b) {
      for(int i = 0; i < b.length; i+=1)
         for(int j = i  + 1; j < b[i].length; j+=1)  {
            int temp = b[i][j]; b[i][j] = b[j][i]; b[j][i] = temp;
         }
   }
   static void f(int[][] b) {
      for(int i = 0; i < b.length; i += 1)
         System.out.println(Arrays.toString(b[i])
                           .replaceAll("\\[|\\]|,", ""));
   }
}
public class MyClass {
   public static void main(String args[]) {
      int[][] a = {{1, 2}, {3, 4}};
      (new User()).function(a);
      User.f(a);
   }
}
```

- 1 3
  2 4
- 1 2
  3 4
- 4 2
  3 1

257. Результат на экране:

```
import java.util.*;
class User {
   static void f(int[][] b) {
      CharSequence str = "[,]";
      for(int i = 0; i < b.length; i += 1)
         System.out.println(Arrays.toString(b[i])
                           .replaceAll("\\[|\\]|,", ""));
   }
}
public class MyClass {
   public static void main(String args[]) {
      int[][] a = {{1, 2}, {3, 4}};
      User.f(a);
   }
}
```

- 1 3
  2 4
- 1 2
  3 4
- 4 2
  3 1

258. Вид результирующей матрицы в памяти после вызова метода:

```
import java.util.*;
class User {
   void function(int[][] b) {
      for(int i = 0; i < b.length; i += 1)
         b[i][i] = b[i][i] * b[i][i];
   }
   static void f(int[][] b) {
      CharSequence str = "[,]";
      for(int i = 0; i < b.length; i += 1)
         System.out.println(Arrays.toString(b[i])
                           .replaceAll("\\[|\\]|,", ""));
   }
}
public class MyClass {
   public static void main(String args[]) {
      int[][] a = {{1, 2}, {3, 4}};
      (new User()).function(a);
      User.f(a);
   }
}
```

- 1 3
  2 4
- 1 2
  3 16
- 1 4
  9 4

259. Вид результирующей матрицы в памяти после вызова метода:

```
import java.util.*;
class User {
   void function(int[][] b) {
      int n = b.length;
      for(int i = 0; i < n; i += 1)
         b[i][n - 1 - i] = b[i][n - 1 - i] * b[i][n - 1 - i];
   }
   static void f(int[][] b) {
```

```
        CharSequence str = "[,]";
        for(int i = 0; i < b.length; i += 1)
           System.out.println(Arrays.toString(b[i])
                              .replaceAll("\\[|\\]|,", ""));
    }
}
public class MyClass {
    public static void main(String args[]) {
        int[][] a = {{1, 2}, {3, 4}};
        (new User()).function(a);
        User.f(a);
    }
}
```

- 1 3
  2 4
- 1 2
  3 16
- 1 4
  9 4

260. Основным типом подпрограмм в языке JAVA является:

- процедура;
- метод;
- оператор повторений.

261. Оператор return служит для:

- возвращения методом значения и/или прекращения ее работы;
- прекращения методом работы без возвращения значения;
- возвращения методом значения без прекращения работы.

262. Если в пользовательском методе с указанным типом возвращаемого значения в ее теле отсутствует оператор return, то такой метод:

- не будет возвращать значения;
- будет возвращать значение 1;
- будет возвращать значение 0;
- компилятор сообщит об ошибке.

263. Контрактом метода называется:

- словесное описание действий метода;
- перечень переменных, объявленных в методе;
- заголовок метода, заканчивающийся перечислением выбрасываемых исключений.

264. В языке Java:

- имеется понятие «вложенного» метода;
- отсутствует понятие «вложенного» метода;
- можно описывать вложенные методы при установке соответствующих директив компилятора.

265. Если методы одного класса имеют одинаковое имя, но разное количество или тип параметров, то такие методы называются:

- вложенными;
- глобальными;
- перегруженными.

266. Перегруженные методы применяются тогда, когда:

- необходимо смоделировать вложенность методов;
- метод должен выполнять различные действия в зависимости от количества параметров и их типов;
- нужно объявить глобальный метод.

267. Если методы отличаются количеством параметров, то:

- их можно перегружать;
- их нельзя перегружать;
- возможность их перегрузки зависит от настроек компилятора.

268. Если методы отличаются только типом возвращаемого значения, то:

- их можно перегружать;
- их нельзя перегружать;
- возможность их перегрузки зависит от настроек компилятора.

269. Областью действия локальной переменной по умолчанию является:

- вся программа;
- тот блок программы, в котором она объявлена;
- тот метод, в котором она объявлена.

270. Если в метод в качестве параметра передана переменная по значению, то изменение этой переменной внутри тела метода:

- приведет к ее изменению и вне тела метода;
- не приведет к ее изменению вне тела метода;
- приведет к ее изменению вне тела метода, но только при соответствующих настройках компилятора.

271. Если в метод в качестве параметра передана ссылка на переменную, то изменение значения этой переменной внутри тела метода:

- приведет к ее изменению и вне тела метода;
- не приведет к ее изменению вне тела метода;
- приведет к ее изменению вне тела метода, но только при соответствующих настройках компилятора.

272. В метод в качестве параметров:

- нельзя передавать ссылки на объекты;
- можно передавать ссылки на объекты;
- можно передавать ссылки, только если это ссылки на массивы.

273. Возвращаемое значение метода:

- может быть ссылкой;
- не может быть ссылкой;
- может быть ссылкой, только если эта ссылка указывает на массив.

274. Результат на экране

```
class User {
   boolean function(final Double x, final Double y) {
      return 0 < x & 0 < y;
   }
}
public class MyClass {
   public static void main(String args[]) {
      System.out.print((new User()).function(3., 1.));
   }
}
```

275. Результат на экране

```
class User {
   boolean function(final Double x, final Double y) {
         return x < -3 && y < 0;
   }
}
public class MyClass {
   public static void main(String args[]) {
      System.out.print((new User()).function(-2., -10.));
   }
}
```

ООП

276. Класс в Java - это:

- тип данных, содержащий в своем заголовке служебное слово class;
- любой тип данных, используемый программистом;
- базовые типы данных.

277. Какие бывают типы вложенных (nested) классов в Java?

- статические вложенные классы;
- внутренние классы;
- локальные классы;
- анонимные классы;
- формальные классы.

278. Выберите правильные утверждения?

- внутренние (inner) классы можно использовать для многократного переопределения одних и тех же методов внутри объемлющего класса;
- локальные (local) классы используются для определения классов внутри любых блоков (в том числе телах методов);
- анонимные классы используются для однократного переопределения внешнего класса при создании его объекта.

279. Зачем применяются интерфейсы в Java?

- интерфейсы позволяют задавать требования к классам, то есть определяют какие методы должны присутствовать в классе;
- интерфейсы используются для стандартизации «поведения» объектов класса.

280. Какие методы может содержать интерфейс?

- абстрактные;
- статические;
- дефолтные;
- простейшие.

281. Какие методы интерфейса должны иметь тело?

- статические;
- дефолтные;
- абстрактные;
- формальные.

282. Выберите правильные утверждения?

- класс реализует интерфейс;
- класс имплементирует интерфейс;
- класс расширяет базовый класс;
- класс расширяет интерфейс.

283. Класс-наследник называется?

- производным классом;
- дочерним классом;
- подкласс;
- суперклассом.

284. Класс-предок называется?

- базовый класс;
- родительский класс;
- суперкласс;
- подкласс.

285. Выберите правильные утверждения?

- иммутабельными объектами являются объекты, состояние которых не изменяется во время выполнения программы;
- иммутабельными объектами являются объекты, при попытке изменить значение которых создается новый объект с другим значением;
- финальными объектами называются объекты, определенные со спецификатором final;
- состояние (значения полей) финального объекта можно изменить;
- иммутабельнотсь эквивалентна финальности.

286. Членами (компонентами) класса могут быть?

- как поля, так и методы, объявленные как private, protected или public;
- только поля, объявленные как private;
- только методы, объявленные как private;
- только поля и методы, объявленные как private;
- только поля и методы, объявленные как public.

287. Что называется конструктором?

- метод, имя которого совпадает с именем класса, вызывающийся при инициализации объекта класса;
- метод, имя которого совпадает с именем класса и который вызывается автоматически до создания объекта класса;
- метод, имя которого необязательно совпадает с именем класса и который вызывается для создания объекта класса.

288. Объект – это:

- участок памяти, физически хранящий состояние объекта;
- экземпляр класса;
- переменная, содержащая ссылку на участок памяти, физически хранящий состояние объекта;
- класс, который содержит в себе свойства и методы их обработки.

289. Существует ли деструктор в JAVA?

290. Что называется наследованием?

- это механизм, посредством которого производный класс получает элементы родительского и может дополнять либо изменять его свойства и методы;
- это механизм, предназначенный исключительно для переопределения методов базового класса;
- это механизм, посредством которого производный класс получает только поля базового класса;
- это механизм, посредством которого производный класс получает элементы родительского, может их дополнить, но не может переопределить.

291. Выберите правильное объявление класса MoreDetails производного от Details?

- class MoreDetails extends Details {};
- class MoreDetails public class Details;
- class MoreDetails: class(Details).

292. Выберите правильное утверждение.

- если элементы класса объявлены как private, то они недоступны ни наследникам класса, ни методам других классов;
- если элементы класса объявлены как private, то они доступны только наследникам класса и методам других классов;
- если элементы объявлены как public, то они доступны наследникам класса, но не методам других классов.

293. Возможность и способ обращения производного класса к элементам базового определяется …

- спецификаторами доступа: private, public, protected в теле базового класса;
- спецификаторами доступа: private, public, protected в теле производного класса;
- только спецификатором доступа protected в заголовке объявления производного класса;

- спецификаторами доступа: private, public, protected в заголовке объявления производного класса.

294. Для доступа к элементам объекта используются:

- при обращении через имя объекта (или ссылку) – точка;
- при обращении через имя объекта (или ссылку) – два двоеточия;
- при обращении через имя объекта – точка, при обращении через ссылку – два двоеточия.

295. Полиморфизм – это:

- средство, позволяющее использовать имя для методов определяющих однотипные действия для объектов родственных классов (связанных иерархией наследования);
- средство, позволяющее в одном классе использовать методы с одинаковыми именами;
- средство, позволяющее перегружать методы для работы с разными типами аргументов или их разным количеством;
- возможность использования параметризации (generics);
- средство, позволяющее в одном классе использовать методы с разными именами для выполнения одинаковых действий.

296. Выберите наиболее полный ответ.

Полиморфизм в объектно-ориентированном программировании реализуется через:

- механизмы перегрузки и переопределения методов, а также параметризацию (создание generics);
- механизмы перегрузки методов;
- создания абстрактных методов.

297. Выберите правильные утверждения:

- у конструктора могут быть параметры;
- конструктор вызывается автоматически при инициализации объекта;
- если конструктор не создан, компилятор создаст его автоматически;
- конструктор наследуется;
- конструктор должен явно вызываться всегда перед объявлением объекта;
- объявление каждого класса должно содержать явно прописанный конструктор.

298. Отметьте правильные утверждения:

- конструкторов класса может быть несколько, но их синтаксис должен подчиняться правилам перегрузки методов;
- конструктор не возвращает значение;

- синтаксис конструкторов класса никак не регламентируется;
- конструктор возвращает ссылку на объект.

299. Дано определение класса

```
class monstr {
    private int health, armo;
    public int color;
    public monstr(int he=50, int arm=10);
}
```

Укажите свойства и методы, доступные методам других классов

- `color; monstr();`
- `health, armo, monstr();`
- `monstr();`
- `health, armo, color, monstr();`

300. Стиль ООП рекомендует объявлять свойства классов:

- со спецификатором доступа private;
- со спецификатором доступа public;
- без спецификаторов доступа;
- со спецификатором local;
- со спецификатором global.

301. Могут ли совпадать имена параметров метода и имена полей (свойств) класса?

- нет;
- да;
- только в записях (record).

302. При описании метода перед его именем указывается:

- модификатор доступа и тип возвращаемого значения;
- имя экземпляра объекта;
- имя поля объекта;
- имя объекта-предка;
- ничего не указывается.

303. Выберите правильные утверждения. Действие спецификатора доступа private распространяется …

- только на поле или метод, перед которым указан;
- до другого спецификатора доступа;
- до спецификатора доступа public;
- до спецификатора доступа protected;
- до конца файла.

304. Какая операция используется для доступа к открытым (публичным) полям (свойствам) объекта?

- операция «.»;
- операция «->»;
- операция «,»;
- операция «::»;
- операция «*».

305. Какие из следующих утверждений правильные?

- поля класса могут быть описаны с использованием ключевого слова static;
- поля класса могут быть описаны с использованием ключевого слова final;
- поля класса могут иметь тип самого класса.

306. Возможность иметь в одном классе несколько методов с одним именем это…

- Overloading (перегрузка);
- Overriding (переопределение);
- Inheritance (наследование);
- Encapsulation (инкапсуляция).

307. Возможность иметь в классах, связанных наследованием методы с одним именем это…

- Overriding (переопределение);
- Overloading (перегрузка);
- Inheritance (наследование);
- Encapsulation (инкапсуляция).

Дженерики

308. Чем механизм generic-ов (шаблонов) отличается от механизма перегрузки?

- перегрузка не требует единообразия алгоритмов перегружаемых методов;
- ничем не отличается, это просто разные наименования одного и того же;
- дженерик создается для методов, различающихся типами данных.

309. Какие виды дженериков (шаблонов) существуют в языке Java?

- шаблоны методов и классов;
- шаблоны конструкторов;
- шаблоны классов и структур.

310. В классе определен дженерик

```
public class MyClass {
    static <T extends Number> boolean cmp(T a, T b) {
        return a.doubleValue() > b.doubleValue() ?
               true : false;
    }
    public static void main(String args[]) {
        double b = 1;
        int a = 2;
        System.out.println(… …);
    }
}
```

Выберите правильный вызов метода `cmp()` для параметров `int a`, `double b` в инструкции `System.out.println(… …);`?

- `cmp((double) a, b)`
- `cmp(a, int(b))`
- `cmp(a, b)`

311. Какое заголовок метода-дженерика `func()` соответствует синтаксису Java?

- `<T> void func(T a, T b)`
- `void func(a, b)`
- `void func<T>(T a, T b)`

312. Выберите правильный (синтаксически верный) заголовок метода-дженерика `func()`, принимающей два параметра типа `T`?

- `<T> void func(T a, T b)`
- `generic <T> void func(T a, T b)`
- `<class T> void func(T a, T b)`

313. Какой правильный вариант описания класса-дженерика?

- `class Array<T> { … }`
- `template <T> class Array { … }`
- `generic <T> class Array { … }`

314. Имеется программа

```
public class MyClass {
```

```
    static <T extends Number> double func1(T a, T b) {
        if(a.intValue() ==0? false : true ) {
            return a.doubleValue() % b.doubleValue();
        }
        return a.doubleValue();
    }
    public static void main(String args[]) {
        double a = 5, b = 6;
        System.out.println(func1(a, b));
    }
}
```

Верен ли код?

```
double a = 5, b = 6;
System.out.println(func1(a, b));
```

- да, все верно;
- нет, ошибка связана с типом операндов в операторе `if`;
- нет, ошибка связана с типом операндов в операторе `%`.

315. Существует ли в Jav-е готовый набор дженериков?

316. Можно ли в шаблоне класса определить статический метод?

- да;
- нет, будет ошибка компиляции;
- да, но результат работы программы непредсказуем.

317. Отметьте все верные утверждения:

- память, занимаемая объектом класса, сгенерированного из шаблона, освобождается автоматически;
- шаблон не является классом;
- все классы-шаблоны наследуют класс `Generic`.

318. Сколько параметров может быть у шаблона при определении шаблона метода?

- один;
- столько, сколько аргументов у метода;
- столько, сколько типов используется для параметризации.

319. Создаваемый класс-дженерик:

- должен быть унаследован от класса `Generic`;
- должен быть отмечен ключевым словом `generic`;
- должен содержать в заголовке, по крайней мере, один параметризованный тип.

320. Отметьте правильный заголовок метода-дженерика.

- `<T, V> void method(T x, V y)`
- `static <T, V> T func(T x, V y)`
- `Generic <T> Sum()`
- `<class T> Sum(T x1, T x2)`

321. Выберите правильный вариант создания экземпляра объекта класса-дженерика

```
class Matrix <T> {
     Matrix(int n, int m);
}
```

- `Matrix a;`
- `Matrix<Double> a = new Matrix<>(4,5);`
- `Matrix :: <float> a(4, 5);`

322. Укажите правильное объявление шаблона метода, если в программе производится вызов `double x = zero();`

- `static <T> int zero() { return 0;}`
- `static T int zero() { return 0;}`
- `<class T> T zero { return 0;}`

Коллекции

323. Что из себя представляет Java Collection Framework?

- иерархия интерфейсов и их реализаций, которая позволяет разработчику использовать большое количество готовых структур данных;
- является частью JDK;
- набор средств работы с массивами в императивном программировании;
- библиотека программирования интерфейса пользователя.

324. Что из себя представляют коллекции/контейнеры?

- это хранилища, поддерживающие различные способы накопления и упорядочения объектов с целью обеспечения возможностей эффективного доступа к ним;
- они представляют собой реализацию абстрактных структур данных, поддерживающих три основные операции (добавление, удаление, изменение элемента в коллекции);
- участок кода с перегруженными функциями;
- совокупность классов, связанных наследованием.

325. Какие типы данных могут хранить коллекции?

- базовые;
- примитивные;
- ссылочные;
- указатели.

326. Используется ли обобщения (параметризация) в коллекциях?

- да, интерфейсы и их имплементирующие коллекции являются обобщенными (параметризованными);
- нет, параметризация не используется в коллекциях.

327. Для чего служит Java Collection framework?

- для избавления программиста от необходимости самостоятельно создавать реализации различных динамических структур данных;
- существенно расширяют возможности разработчика по манипулированию этими данными;
- стандартизируют код, обрабатывающий коллекции;
- редко используемая иерархия интерфейсов и классов.

328. Из каких частей состоит Java Collection Framework:

- интерфейсы;
- классы;
- алгоритмы;
- данные.

329. Недостатки массивов?

- массивы имеют фиксированную длину;
- хранят однотипные данные;
- не имеют недостатков.

330. Какой пакет следует импортировать при использовании коллекций?

- `java.util`;
- `java.lang`;
- `java.math`;

331. Наиболее распространенный принцип группировки (систематизации) классов коллекций?

- по реализации того или иного параметризованного интерфейса;
- по названию;
- по алфавиту.

332. По каким признакам можно осуществить группировку (систематизацию) коллекций?

- по реализации того или иного интерфейса;
- по расширению того или иного абстрактного класса;
- по алфавиту.

333. Что определяет интерфейс?

- определяет базовый функционал, реализующего его класса (коллекции);
- интерфейс практически ничего не определяет, а просто используется для систематизации.

334. Выберите базовые интерфейсы для коллекций?

- интерфейс `Iterable<E>`;
- интерфейс `Collection<E>` ;
- интерфейс `Map<K, V>`;
- интерфейс `List<E>`.

335. Выберите верные утверждение?

- интерфейс `Collection<E>` расширяет интерфейс `Iterable<E>`;
- интерфейс `Iterable<E>` расширяет интерфейс Collection<E>;
- интерфейс `Set<E>` расширяет интерфейс Collection<E>;
- интерфейс `List<E>` расширяет интерфейс Collection<E>;
- интерфейс `Queue<E>` расширяет интерфейс Collection<E>.

336. Выберите верные утверждения?

- интерфейс `SortedSet<E>` - расширяет интерфейс `Set<E>` для создания сортированных коллекций;
- интерфейс `NavigableSet<E>` – в свою очередь расширяет интерфейс `SortedSet<E>` для создания коллекций, в которых можно осуществлять поиск по соответствию;
- интерфейс `Deque<E>` - наследует интерфейс `Queue<E>` и представляет функционал для двусвязных очередей;
- интерфейс `Iterable<E>` расширяет интерфейс Collection<E> для организации последовательного доступа к элементам коллекции.

337. Что из себя представляет шаблон проектирования (паттерн, паттерн проектирования)?

- повторяемая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста;

- обычно шаблон не является законченным образцом, который может быть прямо преобразован в код;
- это образец решения задачи, который можно использовать в различных ситуациях;
- он показывает только общие отношения и взаимодействия между классами или объектами;
- программная реализация параметризованных классов и интерфейсов.

338. Выберите основные методы интерфейса Iterator?

- hasNext();
- next();
- add();
- iteratior().

339. Выберите верные утверждения?

- методы hasNext() и next() обрабатывают следующий элемент коллекции;
- метод hasNext() обрабатывает текущий элемент;
- метод next() возвращает текущий элемент коллекции и переходит к следующему.

340. Выберите основные методы интерфейса Iterable?

- hasNext();
- next();
- add();
- iteratior().

341. Выберите верные утверждения?

- интерфейс Iterable ***не*** расширяет интерфейс Iterator;
- имеет метод iterator(), возвращающий ссылку типа Iterator, для инициализации ссылки на начальное значение объекта-коллекции;
- интерфейс Iterable расширяет интерфейс Iterator.

342. Результат работы программы?

```
import java.util.*;
public class Program {
    public static void main(String[] args) {
        ArrayList s = new ArrayList();
        s.add("8"); s.add("9");
        s.add("7"); s.add("4");
        Iterator<String> iter = s.iterator();
        while(iter.hasNext()){
```

```
                System.out.print(iter.next());
            }
        }
    }
```

343. Результат работы программы?

```
import java.util.*;
public class Program {
    public static void main(String[] args) {
        ArrayList<String> a = new ArrayList<String>();
        a.add("1"); a.add("2");
        ListIterator<String> listIter =
                                a.listIterator();
        while(listIter.hasNext()){
             System.out.print(listIter.next());
        }
        listIter.set("0");
        while(listIter.hasPrevious()){
            System.out.print(listIter.previous());
        }
    }
}
```

344. Выберите верные утверждения?

- интерфейс ListIterator расширяет интерфейс Iterator;
- определяет ряд дополнительных методов к интерфейсу Iterator;
- методы интерфейса ListIterator позволяют обрабатывать список в прямом и обратном порядке;
- интерфейс ListIterator расширяет интерфейс Iterable;
- определяет ряд дополнительных методов к интерфейсу Iterable.

345. Укажите методы интерфейса List?

- add();
- clone();
- equals();
- get();
- size();
- toString();
- hasNext();
- next().

346. Результат работы программы?

```
import java.util.*;
public class Main {
```

```
        public static void main(String args[]) {
           Vector v = new Vector(3, 2);
           System.out.print(v.size());
           v.add(10); v.add(8); v.add(3); v.add(5);
           System.out.print(v.size());
         }
      }
```

347. Результат работы программы?

```
      import java.util.*;
      public class Main {
         public static void main(String args[]) {
            Vector v = new Vector();
            v.add(1); v.add(2); v.set(0, 10);
            v.add("7"); v.add(6.08);
            System.out.println(v);
          }
      }
```

348. Результат работы программы?

```
      import java.util.*;
      public class Main {
         public static void main(String args[]) {
            Vector v = new Vector();
            v.add(1); v.add(2); v.add(3);
            v.set(0, 10); v.add("7");
            System.out.println(v.get(v.size() - 1));
          }
      }
```

349. Результат работы программы?

```
import java.util.*;
public class Main {
    public static void main(String args[]) {
        Vector v = new Vector();
        v.add(1); v.add(2); v.add(3);
        v.set(0, 10); v.add("7");
        System.out.println(v.contains(new Integer(3)));
    }
}
```

350. Результат работы программы?

```
import java.util.*;
public class Main {
   public static void main(String args[]) {
      Vector v = new Vector();
```

```
        v.add(1); v.add(2); v.add(3);
        v.set(0, 10); v.add("7");
        for(var s : v) {
           System.out.print(s + " ");
        }
    }
}
```

351. Результат работы программы?

```
import java.util.*;
public class Main {
   public static void main (String args[]) {
      Vector a = new Vector();
      a.add(1); a.add(2); a.add(3);
      a.clear();
      System.out.println(a);
   }
}
```

352. Выберите верные утверждения?

- коллекция `Vector` позволяет изменять свою длину в ходе выполнения программы;
- вызов метода `clear()` для коллекции коллекции `Vector` необходим, если разработчик старой переменной хочет присвоить новый набор величин содержащий другие значения никак не связанные с предыдущими;
- коллекция `Vector` не может изменить значения элементов;
- коллекция `Vector` является `Immutable`.

353. Результат работы программы?

```
import java.util.*;
public class Main {
   public static void main (String args[]) {
      Vector a = new Vector();
      for(int i = 0; i < 5; i++) {
          a.add(i);
      }
      Vector b = new Vector();
      for(int i = 0; i < 3; i++) {
          b.add(i + 2);
      }
      a.removeAll(b);
      System.out.println(a.toString());
   }
}
```

354. Выберите верны утверждения для применения метода `removeALL()` коллекции `Vector`?

- порядок перечисления элементов в удаляемом массиве неважен;
- удаляемый массив может содержать значения элементов, не существующих в массиве, из которого выполняется удаление;
- элементы удаляемого массива с значениями несовпадающими со значениями массива, из которого происходит удаление также могут иметь произвольные индексы;
- удаляемый массив может иметь большую размерность, чем массив, из которого будет происходить удаление;
- удаляемый массив, ***не*** может содержать в себе в произвольном порядке удаляемый массив; результатом подобной операции будет генерация исключения.

355. Результат работы программы?

```
import java.util.*;
public class Program {
   static void method(Vector a) {
      var temp = a.get(0);
      a.remove(0);
      a.add(temp);
   }
   public static void main (String args[]) {
      Vector a = new Vector();
      for(int i = 0; i < 5; i++) {
          a.add(i);
      }
     method(a);
     System.out.println(a.toString());
   }
}
```

356. Результат работы программы?

```
import java.util.*;
public class Program {
   static void method(Vector a) {
      var temp = a.get(a.size() - 1 );
      a.remove(a.size() - 1);
      a.add(0, temp);
   }
   public static void main (String args[]) {
      Vector a = new Vector();
      for(int i = 0; i < 5; i++) {
          a.add(i);
      }
```

```
      method(a);
      System.out.println(a.toString());
   }
}
```

357. Результат работы программы?

```
import java.util.*;

record Person(String name, int age) { }

public class Program {
   static <T> Vector method(T[] b) {
     Vector array = new Vector();
     for(int i = 0; i < b.length; i++) {
       array.add(b[i]);
     }
     return array;
   }

   public static void main (String args[]) {
     Person[] a = {new Person("Том", 36),
                   new Person("Боб", 5),
                   new Person("Сергей", 43) };
     System.out.println(method(a).get(0));
   }
}
```

358. Результат работы программы?

```
import java.util.*;
record Person(String name, int age) { }
public class Program {
   static Vector<Vector> method(Person[] b){
     Vector<Vector> array = new Vector();
     for(int i = 0; i < b.length; i++) {
            Vector t = new Vector();
            t.add(b[i].name());;
            t.add(b[i].age());
            array.add(t);
     }
     return array;
   }
   public static void main (String args[]) {
     Person[] a = {new Person("Том", 36),
                   new Person("Боб", 36),
                   new Person("Сергей", 36) };
```

```
     System.out.println(method(a).get(0).get(1));
   }
}
```

359. Результат работы программы?

```
import java.util.*;
public class Program {
    static Vector<Vector> method(int n) {
        Vector<Vector> b = new Vector(n);
        for(int i = 0; i < n; i++) {
            Vector tempVector = new Vector();
            for(int j = 0; j < n; j++) {
              tempVector.add(0);
            }
            tempVector.set(n - 1 - i, i + 1);
            b.add(tempVector);
        }
        return b;
    }
   public static void main (String args[]) {
      int n = 2;
      Vector<Vector> a = method(n);
      System.out.println(a.get(1));
   }
}
```

360. Результат работы программы?

```
import java.util.*;
public class Program {
    static void method(Vector<Vector> c) {
      for(int i = 0; i < c.size(); i++) {
         for(int j = i + 1; j < c.get(i).size(); j++){
            var temp = c.get(i).get(j);
            c.get(i).set(j, c.get(j).get(i));
            c.get(j).set(i, temp);
         }
      }
    }

    public static void main (String args[]) {
        Vector<Vector> a = new Vector();
        a.add(new Vector()); a.add(new Vector());
        a.get(0).add(0); a.get(0).add(1);
        a.get(1).add(2); a.get(1).add(3);
        method(a);
```

```
        System.out.println(a.get(0));
    }
}
```

361. Результат работы программы?

```
import java.util.*;
class User{
   private Vector<String> n = new Vector();
   {
      n.add("M"); n.add("Б"); n.add("B");
   }
    int method(String str){
       for(int i = 0; i < n.size(); i++) {
         if(str.equals(this.n.get(i))) {
            return i;
         }
       }
       return -1;
   }
}

public class Main
{
    public static void main(String[] args) {
       User data = new User();
       System.out.print(data. method("M"));
   }
}
```

362. Результат работы программы?

```
import java.util.*;
class User{
   private Vector<Integer> c = new Vector();
   {
      c.add(2); c.add(4); c.add(1);
   }
    int method(String str){
       for(int i = 0; i < c.size(); i++) {
         if(Integer.parseInt(str) == this.c.get(i)) {
            return i;
         }
       }
       return -1;
   }
}

public class Main {
```

```
    public static void main(String[] args) {
       User data = new User();
       System.out.print(data.method("4"));
   }
}
```

363. Выберите верные утверждения?

- класс `Stack` является расширением коллекции `Vector`.
- класс `Stack` является реализацией стека `LIFO` (last-in-first-out – последний-вошёл-первый-вышел).
- класс `Stack` поддерживает только один конструктор «по умолчанию» `Stack()` (без параметров);
- является частично синхронизированной коллекцией (кроме метода добавления `push()`).
- класс `Stack` часто используется.

364. Укажите методы интерфейса List?

- peek();
- pop();
- push();
- add();
- clone();
- equals();
- hasNext();
- next().

365. Результат работы программы?

```
import java.util.*;
public class Main {
   public static void main(String args[]) {
      Stack stack = new Stack();
      stack.push(0); stack.push(1);
      stack.push(2);
      System.out.println(stack);
   }
}
```

366. Результат работы программы?

```
import java.util.*;
public class Main {
   public static void main(String args[]) {
      Stack stack = new Stack();
      stack.push(0); stack.push(1);
      stack.push(2);
```

```
      System.out.println(stack.peek());
   }
}
```

367. Результат работы программы?

```
import java.util.*;
public class Main {
   public static void main(String args[]) {
      Stack stack = new Stack();
      stack.push(0); stack.push(1);
      stack.push(2);
      System.out.println(stack.search(0));
   }
}
```

368. Результат работы программы?

```
import java.util.*;
public class Main {
   public static void main(String args[]) {
      Stack stack = new Stack();
      stack.push(0); stack.push(1);
      stack.push(2);
      System.out.println(stack.pop());
   }
}
```

369. Результат работы программы?

```
import java.util.*;
public class Main {
   public static void main(String args[]) {
      Stack s = new Stack();
      System.out.println(s.size());
      s.add(new Integer(1)); s.add(2);
      s.add(3); s.add(4);
   }
}
```

370. Результат работы программы?

```
import java.util.*;
public class Main {
   public static void main(String args[]) {
      Stack s = new Stack();
      s.add(new Integer(1)); s.add(2);
      s.add(3); s.add(4);
      for(int i = 0 ; i < s.size() ; i++) {
```

```
            System.out.print(s.get(i));
        }
    }
}
```

371. Результат работы программы?

```
import java.util.*;
public class Main {
    public static void main(String args[]) {
        Stack s = new Stack();
        s.add(new Integer(1)); s.add(2);
        s.add(3); s.add(4);
        s.set(0, new Integer(10));
        s.add(new String("abv"));
        System.out.println(s.get(0));
    }
}
```

372. Результат работы программы?

```
import java.util.*;
public class Main {
    public static void main(String args[]) {
        Stack s = new Stack();
        s.add(new Integer(1)); s.add(2);
        s.add(3); s.add(4);
        s.set(0, new Integer(10));
        s.add(new String("abv"));
        System.out.println(s.get(s.size() - 1));
    }
}
```

373. Результат работы программы?

```
import java.util.*;
public class Main {
    public static void main(String args[]) {
        Stack s = new Stack();
        s.add(new Integer(1)); s.add(2);
        s.set(0, new Integer(10));
        s.add(new String("abv"));
        for(var t : s) {
            System.out.print(t);
        }
    }
}
```

374. Результат работы программы?

```
import java.util.*;
public class Main {
```

```
   public static void main(String args[]) {
      Stack s = new Stack();
      s.add(new Integer(1)); s.add(2);
      s.add(3); s.add(4);
      Iterator iter = s.iterator();
      while(iter.hasNext()) {
         System.out.print(iter.next() + " ");
      }
   }
}
```

375. Результат работы программы?

```
import java.util.*;
public class Main {
   public static void main(String args[]) {
      Stack s = new Stack();
      s.add(3); s.add(4);
      s.set(0, new Integer(10));
      s.add(new String("abv"));
      Iterator iter = s.iterator();
      while(iter.hasNext()) {
         System.out.print(s.pop() + " ");
      }
   }
}
```

376. Результат работы программы?

```
import java.util.*;
public class Main {
   public static void main (String args[]) {
      Stack s = new Stack();
      for(int i = 0; i < 5; i++) {
          s.push(i);
      }
      Stack d = new Stack();
      for(int i = 0; i < 3; i++) {
          d.add(i + 3);
      }
      s.removeAll(d);
      System.out.println(s);
   }
}
```

377. Результат работы программы?

```
import java.util.*;
public class Program {
   static Stack method1(int n) {
     Stack s = new Stack();
```

```
      for(int i = 0; i < n; i++) {
        s.add(i + 1);
      }
      s.push(12.56); s.push(new Boolean(false));
      return s;
   }
   static void method2(Stack s) {
      Iterator iter = s.iterator();
      var temp = iter.next();
      iter.remove();
      s.push(temp);
   }

   public static void main (String args[]) {
      Stack s = method1(5);
      method2(s);
      System.out.println(s.toString());
   }
}
```

378. Результат работы программы?

```
import java.util.*;
public class Program {
   static Stack method1(int n) {
     Stack s = new Stack();
     for(int i = 0; i < n; i++) {
       s.push(i + 1);
     }
     s.add(new Boolean(true));
     return s;
   }
   static void method2(Stack s) {
      Stack temporary = (Stack) s.clone();
      s.clear();
      s.push(temporary.pop());
      Iterator iter = temporary.iterator();
      while(iter.hasNext()) {
         s.push(iter.next());
      }
   }
   public static void main (String args[]) {
     Stack s = method1(2);
     method2(s);
     System.out.println(s.toString());
   }
}
```

379. Результат работы программы?

```
import java.util.*;
public class Program {
   static Stack method1(int n) {
     Stack s = new Stack();
     for(int i = 0; i < n; i++) {
       s.push(i + 1);
     }
     s.add(12.56);
     return s;
   }
   static void method2(Stack s) {
      Stack temporary = (Stack) s.clone();
      s.clear();
      Iterator iter = temporary.iterator();
      while(iter.hasNext()) {
           s.push(temporary.pop());
      }
   }

   public static void main (String args[]) {
     Stack s = method1(2);
     method2(s);
     System.out.println(s);
   }
}
```

380. Результат работы программы?

```
import java.util.*;

record Person(String n, int a) { }

public class Program {
   static <T> Stack method(T[] a) {
     Stack s = new Stack();
     for(int i = 0; i < a.length; i++) {
       s.push(a[i]);
     }
     return s;
   }

   public static void main (String args[]) {
     Person[] a = {new Person("Иван", 28),
                   new Person("Петр", 15),
                   new Person("Сергей", 45) };
     System.out.println(method(a).get(1));
   }
}
```

381. Результат работы программы?

```
import java.util.*;

record Person(String n, int a) { }

public class Program {
   static Stack<Stack> method(Person[] a){
     Stack<Stack> s = new Stack();
     for(int i = 0; i < a.length; i++) {
            Stack temporary = new Stack();
            temporary.push(a[i].n());;
            temporary.push(a[i].a());
            s.add(temporary);
     }
     return s;
   }
   public static void main (String args[]) {
     Person[] a = {new Person("Том", 36),
                   new Person("Боб", 20),
                   new Person("Сергей", 32) };
     System.out.println(method(a).pop().get(1));
   }
}
```

382. Результат работы программы?

```
import java.util.*;
public class Program {
   static Stack<Stack> method1(int n) {
      Stack<Stack> s = new Stack();
      for(int i = 0; i < n; i++) {
         Stack t = new Stack();
         int j;
         for(j = 0; j < n - i - 1; j++) {
            t.push(j + 1);
         }
         t.push(i + 1);
         for(j = j + 1; j < n; j++) {
            t.add(0);
         }
         s.add(t);
      }
      return s;
   }
   static void method2(Stack<Stack> s) {
      Iterator iter = s.iterator();
```

```
      while(iter.hasNext()) {
         System.out.println(iter.next());
      }
   }
   public static void main (String args[]) {
      int n = 3;
      Stack<Stack> s = method1(n);
      method2(s);
   }
}
```

383. Результат работы программы?

```
import java.util.*;
class User{
    private Stack<String> n = new Stack();
    private Stack<Integer> p = new Stack();
    {
        n.push("A"); n.push("Б"); n.push("B");
        p.push(1); p.push(2); p.push(3);
    }
    int f1(String s){
       Iterator iter = n.iterator();
       for(int i = 0; iter.hasNext(); i++) {
         if(s.equals(iter.next())) {
            return i;
         }
       }
       return -1;
   }
   int f2(String s) {
      int i;
      if ((i = f1(s)) < 0) return i;
      Iterator iter = p.iterator();
       for(int j = 0; j < i; j++) {
         iter.next();
       }
      return ((Integer) iter.next()).intValue();
   }
}

public class Main {
    public static void main(String[] args) {
       User d = new User();
       System.out.print(d.f2("M"));
   }
}
```

384. Выберите верные утверждения?

- коллекция `ArrayList` реализует интерфейс `List`.
- `ArrayList` является реализацией динамического массива объектов;
- коллекцию `ArrayList` следует применять, если в процессе работы предполагается частое обращение к элементам по индексу (например, применение сортировок);
- данную коллекцию рекомендуется избегать, если требуется частое удаление/добавление элементов в середину коллекции;
- ***не*** позволяет хранить любые данные в качестве элемента.

385. Выберите свойства коллекции `ArrayList`?

- быстрый доступ к элементам коллекции *по индексу* за время `O(1)`;
- медленный доступ к элементам коллекции *по значению* за линейное время `O(n)`;
- медленный (за линейное время `O(n)`), когда вставляются и удаляются элементы из «середины» списка;
- коллекция не синхронизирована (для многопоточности, т.е. разрешен доступ из разных потоков).
- ***не*** позволяет хранить `null` в качестве элемента.

386. Результат работы программы?

```
import java.util.*;
public class Main {
   public static void main(String args[]) {
      ArrayList a =
      new ArrayList(Arrays.asList(1, "два", true, 4));
      System.out.println(a);
   }
}
```

387. Результат работы программы?

```
import java.util.*;
public class Main {
   public static void main(String args[]) {
      String[] str =
          {"раз", (new Integer(2)).toString(), "три"};
      ArrayList a =
                 new ArrayList(Arrays.asList(str));
      System.out.println(a);
   }
}
```

388. Результат работы программы?

```
import java.util.*;
public class Main {
   public static void main(String args[]) {
      ArrayList a = new ArrayList();
      a.add(1); a.add("Строка"); a.add(4.56);
      a.set(2, 10); a.set(0, 0); a.set(1, 20);
      System.out.println(a);
   }
}
```

389. Результат работы программы?

```
import java.util.*;
public class Main {
   public static void main(String args[]) {
      ArrayList a = new ArrayList();
      a.add(1); a.add("Строка"); a.add(4.56);
      a.set(2, 10); a.set(0, 0); a.set(1, 20);
      System.out.println(a.get(a.size() - 2));
   }
}
```

390. Результат работы программы?

```
import java.util.*;
public class Main {
   public static void main(String args[]) {
      ArrayList a = new ArrayList();
      a.add(1); a.add("Строка"); a.add(4.56);
      a.set(2, 10); a.set(0, 0); a.set(1, 20);
      System.out.println(a.contains(new Integer(10)));
   }
}
```

391. Результат работы программы?

```
import java.util.*;
public class Main {
   public static void main(String args[]) {
      ArrayList a = new ArrayList();
      a.add(1); a.add("Строка"); a.add(4.56);
      a.set(2, 10); a.set(0, 0); a.set(1, 20);
      Integer[] r = new Integer[a.size()];
      a.toArray(r);
      for (var e : r) {
            System.out.print(e);
      }
   }
}
```

392. Результат работы программы?

```
import java.util.*;
public class Main {
   public static void main(String args[]) {
      ArrayList a = new ArrayList();
      a.add(1); a.add("Строка"); a.add(4.56);
      a.set(2, 10); a.set(0, 0); a.set(1, 20);
      Integer[] r = new Integer[a.size()];
      Object[] obj = a.toArray(r);
      System.out.print(Arrays.toString(obj));
   }
}
```

393. Результат работы программы?

```
import java.util.*;
public class Main {
   public static void main(String args[]) {
      ArrayList a = new ArrayList();
      a.add(1); a.add("Строка"); a.add(4.56);
      a.set(2, 10); a.set(0, 0); a.set(1, 20);
      Integer[] r = new Integer[a.size()];
      r = (Integer[]) a.toArray(r);
      for (var e : r) {
            System.out.print(e + " ");
      }
   }
}
```

394. Результат работы программы?

```
import java.util.*;
public class Main extends ArrayList {
    public static void main(String[] args)
    {
        Main arr = new Main();
        arr.add(1.5); arr.add("Строка");
        arr.add(3); arr.add(new Boolean("true"));
        arr.removeRange(0, 2);
        System.out.println(arr);
    }
}
```

395. Укажите методы интерфейса ListIterator?

- peek();
- pop();
- push();
- clone();
- add();

- hasNext();
- hasPrevious();
- next();
- previous().

396. Результат работы программы?

```
import java.util.*;
public class Main {
    public static void main(String[] args) {
        ArrayList states = new ArrayList();
        states.add("G"); states.add("F");
        states.add("I"); states.add("S");
        ListIterator listIter = states.listIterator();
        while(listIter.hasNext()) {
            listIter.next();
        }
        listIter.set("Я");
        System.out.print(listIter.nextIndex());
        listIter.add("Б");
        while(listIter.hasPrevious()) {
            System.out.print(listIter.previous());
        }
    }
}
```

397. Результат работы программы?

```
import java.util.*;
public class Program {
   void method1(ArrayList a, int n) {
     for(int i = 0; i < n; i++) {
         a.add(i + 1);
     }
   }
   void method2(ArrayList a) {
      ListIterator iter = a.listIterator();
      var temp = iter.next();
      iter.remove();
      while(iter.hasNext()){
         iter.next();
      }
      iter.add(temp);
   }
   public static void main (String args[]) {
     Program ex = new Program();
     ArrayList a = new ArrayList();
```

```
      ex.method1(a, 5);
      ex.method2(a);
      System.out.println(a);
   }
}
```

398. Результат работы программы?

```
import java.util.*;
public class Program {
   void method(ArrayList a, int n) {
      for(int i = 0; i < n; i++) {
         for(int j = 0; j < n; j++) {
            if(i >= j) a.add(i * n + j, j + 1);
            else a.add(i * n + j, 0);
         }
      }
   }

   public static void main (String args[]) {
      int n = 2;
      Program ex = new Program();
      ArrayList b = new ArrayList();
      ex.method(b, n);
      System.out.println(b);
   }
}
```

399. Результат работы программы?

```
import java.util.*;
public class Program {
    void method1(ArrayList a, int n) {
        for(int i = 0; i < n; i++) {
            for(int j = 0; j < n; j++) {
                if(i >= j) a.add(i * n + j, j + 1);
                else a.add(i * n + j, 0);
            }
        }
    }
    void method2(ArrayList a) {
        int n =
           (new Double(Math.sqrt(a.size()))).intValue();
        for(int i = 0; i < n; i++) {
            for(int j = i + 1; j < n; j++) {
                var temp = a.get(i * n + j);
                a.set(i * n + j, a.get(j * n + i));
                a.set(j * n + i, temp);
            }
```

```
        }
    }
    public static void main (String args[]) {
        int n = 2;
        Program ex = new Program();
        ArrayList b = new ArrayList();
        ex.method1(b, n);
        ex.method2(b);
        System.out.println(b);
    }
}
```

400. Результат работы программы?

```
import java.util.*;

record User(String n, int a) { }

public class Program
{
   static <T> void method(ArrayList a, T[] arr){
     ListIterator it = a.listIterator();
     for(int i = 0; i < arr.length; i++) {
        it.add(arr[i]);
     }
   }
   public static void main (String args[]) {
     User[] p = {new User("T", 25),
                 new User("Б", 33),
                 new User("C", 45) };
     ArrayList b = new ArrayList();
     method(b, p);
     System.out.println(b.get(2));
   }
}
```

401. Результат работы программы?

```
import java.util.*;
class User {
   private ArrayList<String> a = new ArrayList();
   private ArrayList<Integer> b = new ArrayList();
   {
      a.add("M"); a.add("Б"); a.add("Витебск");
      b.add(1); b.add(2); b.add(3);
   }
    int method1(String c){
       for(int i = 0; i < a.size(); i++) {
         if(c.equals(this.a.get(i))) {
```

```
            return i;
          }
        }
        return -1;
    }
    int method2(String n) {
       int i = method1(n);
       if (i < 0) return i;
       return this.b.get(i);
    }
}
public class Main
{
     public static void main(String[] args) {
        User data = new User ();
        System.out.print(data.method2("M"));
    }
}
```

402. Выберите верные утверждения?

- для того чтобы объекты можно было сравнить и сортировать, они должны реализовать параметризованный интерфейс `Comparable`;
- интерфейс `Comparable` содержит один единственный метод `compareTo()`;
- метод `compareTo()` сравнивает текущий объект с объектом, переданным в качестве параметра;
- если метод `compareTo()` возвращает отрицательное число, то текущий объект располагается перед тем, который передается через параметр.
- если метод `compareTo()` возвращает положительное число, то, наоборот, после второго объекта.
- при использовании метода `compareTo()` значение `null` может быть значением участвующего в сравнении экземпляра класса;
- метод `compareTo()` статический.

403. Выберите верные утверждения?

- интерфейс `Comparator` используется если класс по какой-то причине не может реализовать интерфейс `Comparable`, или же просто нужен другой вариант правила сравнения;
- интерфейс `Comparator` содержит один из основных методов, а именно метод `compare()`;
- метод `compare()` возвращает числовое значение;
- для переопределения метода `compare()` интерфейс `Comparator` ***не*** должен быть реализован классом.

404. Выберите верные утверждения?

- интерфейс `Queue` описывает коллекции с предопределенным способом вставки и извлечения элементов, а именно — очереди;
- очередь – структура данных, в которой добавление элемента возможно лишь в конец очереди, а выборка — только из начала очереди, при этом выбранный элемент из очереди удаляется;
- в очереди реализуется принцип «первым вошел — первым вышел» (англ. first-in, first-out — `FIFO`);
- класс `Queue` является реализацией стека `LIFO` (last-in-first-out – последний-вошёл-первый-вышел).

405. Укажите методы `Queue`, дополняющие методы `Collection`?

- `next();`
- `previous();`
- `peek();`
- `poll()`.

406. Выберите верные утверждения?

- особенностью очереди `PriorityQueue` является возможность управления порядком элементов (приоритетом);
- «По умолчанию», элементы упорядочиваются (сортируются) с использованием «`natural ordering`»;
- правило упорядочивания можно переопределить при помощи объекта `Comparator`, который задается при создании очереди;
- коллекция `PriorityQueue` поддерживает `null` в качестве элементов.

407. Результат работы программы?

```
import java.util.*;
record User(String str, int a)
                        implements Comparable<User> {
    @Override
    public int compareTo(User obj) {
        return this.a() - obj.a();
    }
    @Override
    public String toString() {
        return "User {" + "Str = " + str + "}";
    }
}
public class Demo
{
    public static void main(String[] args) {
```

```
        PriorityQueue q = new PriorityQueue();
        q.add(new User("Саша", 36));
        q.add(new User("Маша", 23));
        q.add(new User("Даша", 34));
        System.out.println(q.peek());
    }
}
```

408. Результат работы программы?

```
import java.util.*;
public class QueueDemo
{
    public static void main(String[] args) {
        Vector v = new Vector();
        v.add(new Integer(36));
        v.add(new Integer(23));
        v.add(new Integer(34));
        PriorityQueue queue = new PriorityQueue(v);
        System.out.println(queue);
    }
}
```

409. Результат работы программы?

```
import java.util.*;
record User(String str, int a)
                        implements Comparable<User> {
    @Override
    public int compareTo(User obj) {
        return this.str().compareTo(obj.str());
    }
    @Override
    public String toString() {
        return "User {" + "Str = " + str + "}";
    }
}
public class Demo
{
    public static void main(String[] args) {
        PriorityQueue q = new PriorityQueue();
        q.add(new User("Саша", 36));
        q.add(new User("Маша", 23));
        q.add(new User("Даша", 34));
        System.out.println(q.peek());
    }
}
```

410. Результат работы программы?

```
import java.util.*;
record User(String str, int a) {
    @Override
    public String toString() {
        return "User {" + "Str = " + str + "}";
    }
}
class UserCmp implements Comparator<User>{
    @Override
    public int compare(User o1, User o2) {
       return o1.str().compareTo(o2.str());
    }
}
public class Demo
{
    public static void main(String[] args) {
        UserCmp cmp = new UserCmp();
        PriorityQueue q = new PriorityQueue (cmp);
        q.add(new User("Саша", 36));
        q.add(new User("Маша", 23));
        q.add(new User("Даша", 34));
        System.out.println(q.peek());
    }
}
```

411. Результат работы программы?

```
import java.util.*;
record User(String str, int a) {
    @Override
    public String toString() {
        return "User {" + "Str = " + str + "}";
    }
}
class UserCmp implements Comparator<User>{
    @Override
    public int compare(User o1, User o2) {
       return o1.a() - o2.a();
    }
}
public class Demo
{
    public static void main(String[] args) {
        UserCmp cmp = new UserCmp();
```

```
        PriorityQueue q = new PriorityQueue(cmp);
        q.add(new User("Саша", 36));
        q.add(new User("Маша", 23));
        q.add(new User("Даша", 34));
        System.out.println(q.peek());
    }
}
```

412. Результат работы программы?

```
import java.util.*;
record User(String str, int a) {
    @Override
    public String toString() {
        return "User{" +
        "str = " + str + ", a = " + a +'}';
    }
}
public class Demo
{
    public static void main(String[] args) {
        Comparator<User> cmp = new Comparator<User>()
        {
            @Override
            public int compare(User o1, User o2) {
                return o1.str().compareTo(o2.str());
            }
        };
        PriorityQueue q = new PriorityQueue(cmp);
        q.add(new User("Иванов", 36));
        q.add(new User("Иванов", 25));
        System.out.println(q.peek());
    }
}
```

413. Результат работы программы?

```
import java.util.*;
record User(String str, int a) {
    @Override
    public String toString() {
        return "User{ a = " + a + " }";
    }
}
public class Demo
{
    public static void main(String[] args) {
```

```
        Comparator<User> cmp = new Comparator<User>()
        {
            @Override
            public int compare(User o1, User o2) {
                return o1.a() - o2.a();
            }
        };
        PriorityQueue q = new PriorityQueue(cmp);
        q.add(new User("Иванов", 36));
        q.add(new User("Иванов", 25));
        System.out.println(q.peek());
    }
}
```

414. Результат работы программы?

```
import java.util.*;
record User(String str, int a) {
    public String toString() {
        return "User { " +
             "str = " + str + ", a = " + a + " }";
    }
}
class User1 implements Comparator<User> {
    public int compare(User a, User b){
       return a.str().compareTo(b.str());
    }
}
class User2 implements Comparator<User>{
   public int compare(User a, User b){
      return a.a() - b.a();
   }
}
public class ComparatorDemo
{
   public static void main(String[] args) {
      Comparator<User> object =
             new User1().thenComparing(new User2());
      PriorityQueue q =
           new PriorityQueue (10, object);
      q.add(new User("Маша", 23));
      q.add(new User("Даша", 34));
      q.add(new User("Даша", 25));
      System.out.println(q.peek());
   }
}
```

415. Результат работы программы?

```
import java.util.*;
record User(String str, int a) {
    public String toString() {
        return "User { " +
             "str = " + str + ", a = " + a + " }";
    }
}
public class ComparatorDemo
{
   public static void main(String[] args) {
      Comparator<User> object = Comparator
                           .comparing(User::a)
                           .thenComparing(User::str);
      PriorityQueue q = new PriorityQueue (object);
      q.add(new User("Саша", 21));
      q.add(new User("Даша", 34));
      q.add(new User("Даша", 25));
      System.out.println(q.peek());
   }
}
```

416. Результат работы программы?

```
import java.util.*;
record User(String str, int a) {
    public String toString() {
        return "User { " +
             "str = " + str + ", a = " + a + " }";
    }
}
public class ComparatorDemo
{
   public static void main(String[] args) {
      Comparator<User> object = Comparator
                              .comparing(User::str)
                           .thenComparing(User::a);
      PriorityQueue q = new PriorityQueue (object);
      q.add(new User("Саша", 21));
      q.add(new User("Даша", 34));
      q.add(new User("Даша", 25));
      System.out.println(q.peek());
   }
}
```

417. Результат работы программы?

```
import java.util.*;
record User(String str, int a) {
    public String toString() {
        return "User { " +
             "str = " + str + ", a = " + a + " }";
    }
}
public class ComparatorDemo
{
   public static void main(String[] args) {
      Comparator<User> object = Comparator
                                .comparing(User::str);
      PriorityQueue q = new PriorityQueue (object);
      q.add(new User("Саша", 21));
      q.add(new User("Даша", 34));
      q.add(new User("Даша", 25));
      System.out.println(q.peek());
   }
}
```

418. Результат работы программы?

```
import java.util.*;
record User(String str, int a) {
    public String toString() {
        return "User { " +
             "str = " + str + ", a = " + a + " }";
    }
}
public class ComparatorDemo
{
   public static void main(String[] args) {
      Comparator<User> object = Comparator
                                .comparing(User::a);
      PriorityQueue q = new PriorityQueue (object);
      q.add(new User("Саша", 21));
      q.add(new User("Даша", 34));
      q.add(new User("Даша", 25));
      System.out.println(q.peek());
   }
}
```

419. Результат работы программы?

```
import java.util.*;
public class QueueExample
```

```
{
    static PriorityQueue User(Collection c) {
        return new PriorityQueue(c);
    }
    public static void main(String[] args) {
        Vector v = new Vector();
        v.add("В"); v.add("Б");
        v.add("Г"); v.add("М");
        v = new Vector(User(v));
        System.out.println(v);
    }
}
```

420. Укажите методы `PriorityQueue`?

- `next();`
- `previous();`
- `comparator();`
- `iterator();`
- `toArray();`

421. Выберите верные утверждения?

- интерфейс `Deque` расширяет интерфейс `Queue` и определяет поведение двунаправленной очереди;
- объект класса `Deque` может работать как обычная однонаправленная очередь
- объект класса `Deque` может работать как стек.

422. Укажите методы `Deque`?

- `next();`
- `previous();`
- `addFirst();`
- `getLast();`
- `pop();`
- `push();`

423. Выберите верные утверждения?

- коллекция `ArrayDeque` — реализация интерфейса `Deque`;
- коллекция `ArrayDeque` расширяет интерфейс `Queue`;
- коллекция `ArrayDeque` позволяют реализовать конструкцию вида LIFO (last-in, first-out);
- коллекция `ArrayDeque` позволяют реализовать конструкцию FIFO (first-in, first-out);

- коллекция `ArrayDeque` представляет возможность обращения к элементу по индексу.

424. Выберите верные утверждения?

- В коллекции `ArrayDeque` запрещено хранение значения `null`;
- коллекция `ArrayDeque` не поддерживает многопоточный безопасный доступ;
- коллекция `ArrayDeque` работает быстрее чем `Stack`, если используется как LIFO коллекция, а также быстрее чем `LinkedList`, если используется как FIFO;
- коллекция `ArrayDeque` позволяет работать со списком с обоих концов;
- некоторые методы коллекции `ArrayDeque` могут выбрасывать исключения;
- поддержка двунаправленного обхода итераторатораторами.
- коллекция `ArrayDeque` позволяет обращаться к элементам по индексу.

425. Укажите методы `Deque`?

- `previous();`
- `next();`
- `addFirst();`
- `getLast();`
- `descendingIterator().`

426. Результат работы программы?

```
import java.util.*;
public class Examples
{
   public static void main(String[] args) {
      ArrayDeque arrDeque = new ArrayDeque();
      arrDeque.add("Г");
      arrDeque.addFirst(1);
      arrDeque.push(3.14);
      arrDeque.addLast('И');
      System.out.println(arrDeque);
   }
}
```

427. Результат работы программы?

```
import java.util.*;

record User(String name) {}
```

```
public class Example
{
   public static void main(String[] args) {
      ArrayDeque arrDeque = new ArrayDeque();
      arrDeque.add("Г");
      arrDeque.addFirst(1);
      arrDeque.addLast('И');
      arrDeque.push(new User("Я"));
      System.out.println(arrDeque.getFirst());
   }
}
```

428. Результат работы программы?

```
import java.util.*;

record User(String name) {}

public class Example
{
   public static void main(String[] args) {
      ArrayDeque arrDeque = new ArrayDeque();
      arrDeque.add("Г");
      arrDeque.addFirst(1);
      arrDeque.addLast('И');
      arrDeque.push(new User("Я"));
      System.out.println(arrDeque.getLast());
   }
}
```

429. Результат работы программы?

```
import java.util.*;
public class Examples
{
   public static void main(String[] args) {
      ArrayDeque arrDeque = new ArrayDeque();
      arrDeque.add("Г");
      arrDeque.addFirst(1);
      arrDeque.push(3.14);
      arrDeque.addLast('И');
      while(arrDeque.peek()!=null) {
         System.out.print(arrDeque.pop());
      }
   }
}
```

430. Результат работы программы?

```
import java.util.*;
public class Demo
{
   public static void main(String[] args) {
      Deque<Integer> deQue = new ArrayDeque<>();
      deQue.addFirst(5);
      deQue.addFirst(2);
      deQue.addLast(2);
      deQue.addLast(1);
      for (Iterator itr = deQue.iterator();
           itr.hasNext(); ) {
         System.out.print(itr.next());
      }
    }
}
```

431. Результат работы программы?

```
import java.util.*;
public class Demo
{
   public static void main(String[] args) {
      Deque<Integer> deQue = new ArrayDeque<>();
      deQue.addFirst(5);
      deQue.addFirst(2);
      deQue.addLast(2);
      deQue.addLast(1);
      for (Iterator dItr = deQue.descendingIterator();
           dItr.hasNext(); ) {
         System.out.print(dItr.next());
      }
    }
}
```

432. Результат работы программы?

```
import java.util.*;
public class Program {
   void user(ArrayDeque a) {
      var temp = a.removeFirst();
      a.addLast(temp);
   }
   public static void main (String args[]) {
     ArrayDeque a =
                new ArrayDeque(Arrays.asList(1,2,3));
```

```
      (new Program()).user(a);
      System.out.println(a);
   }
}
```

433. Результат работы программы?

```
import java.util.*;
public class Program {
   void user(ArrayDeque a) {
      var temp = a.removeLast();
      a.addFirst(temp);
   }
   public static void main (String args[]) {
     ArrayDeque a =
               new ArrayDeque(Arrays.asList(1,2,3));
     (new Program()).user(a);
     System.out.println(a);
   }
}
```

434. Результат работы программы?

```
import java.util.*;
public class Program {
   static void user(ArrayDeque a) {
      ArrayDeque copy = a.clone();
      a.clear();
      for(Iterator dItr = copy.descendingIterator();
          dItr.hasNext(); ) {
         a.add(dItr.next());
      }
   }
   public static void main (String args[]) {
     ArrayDeque a =
                new ArrayDeque(Arrays.asList(1,2,3));
     user(a);
     System.out.println(a);
   }
}
```

435. Выберите верные утверждения?

- `LinkedList` — коллекция, реализующая интерфейсы `Dequeue`, `Queue` и `List`;
- коллекция `LinkedList` позволяет хранить любые данные, включая `null`;

- особенностью реализации коллекции `LinkedList` является то, что в ее основе лежит двунаправленный связный список (каждый элемент имеет ссылку на предыдущий и следующий);
- коллекцию `LinkedList` ***нельзя*** использовать как стек или очередь;
- коллекция `LinkedList` имеет методы ***отличные*** от методов, определенных интерфейсами `Deque` и `List`.

436. Выберите верные утверждения для коллекции `LinkedList`?

- можно организовать стэк (`LIFO`), очередь (`FIFO`), или двустороннюю очередь (очередь с возможностью добавления в начало или конец), со временем доступа `O(1)`;
- на вставку и удаление из середины списка, получение элемента по индексу или значению потребуется линейное время `O(n)`.
- на добавление и удаление из середины списка, используя `ListIterator.add()` и `ListIterator.remove()`, потребуется `O(1)`;
- позволяет добавлять любые значения в том числе и `null`;
- для хранения примитивных типов использует соответствующие классы-обертки;
- синхронизирована.

437. Выберите верные утверждения?

- вставка/удаление в/из середин-у/ы коллекций, реализующих `Deque`, нужного/ненужного элемента требуют только переопределения ссылок на следующий и предыдущий элементы коллекции;
- при вставке в середину коллекции, реализующей только интерфейс `List` необходимо не только проверить и при необходимости переопределить `capacity` коллекции, то и сам процесс вставки/удаления элемента из середины сопровождается сдвигом всех элементов коллекции влево/право;
- формально, если в программе чаще происходят операции вставки/удаления в/из середины списка, то коллекции, реализующие `Deque`, должны работать быстрее, чем коллекции, реализующие только `List`;
- если учитывать время, затрачиваемое на обязательную предварительную операцию поиска удаляемого или места вставляемого элемента в коллекциях реализующих `Deque` (это использование итератора за время `O(n)`) и в коллекции реализующей `List` (это использование метода `get()` за время `O(1)`), то говорить о явных преимуществах одной коллекции перед другой не возможно.

438. Выберите верные утверждения?

- коллекции, реализующие интерфейс `Map<K, V>` представляют собой «соответствие» или иначе говоря «словарь», где каждый элемент представляет пару «ключ-значение»;
- коллекции, реализующие интерфейс `Map<K, V>` также называется в русскоязычной литературе «отображением» (имеется ввиду множества ключей во множество значений);
- все ключи коллекции интерфейса `Map<K, V>` должны быть уникальными в рамках имплементирующей коллекции;
- коллекции, реализующие интерфейс `Map<K, V>`, облегчают поиск элемента по ключу;
- интерфейс `Map` расширяет интерфейс `Collection`.

439. Укажите методы интерфейса `Map`?

- `containsKey();`
- `containsValue();`
- `get();`
- `put();`
- `putIfAbsent();`
- `next();`
- `addFirst();`
- `getLast();`
- `descendingIterator()`.

440. Что такое хеш-таблица?

- это специальная структура данных для хранения пар ключей и их значений;
- это ассоциативный массив, в котором ключ обрабатывается хеш-функцией;
- массив значений хеш-функции.

441. Все три операции: вставка, поиск и удаление — в среднем выполняются за время O(1), среднее время поиска по ней также равно O(1) и O(n) в худшем случае?

- да;
- нет.

442. Выберите верные утверждения?

- коллекция `Hashtable` — реализация такой структуры данных, как хэш-таблица;
- коллекция `Hashtable` не позволяет использовать `null` в качестве значения или ключа;

- Hashtable является синхронизированной (почти все методы помечены как synchronized);
- из-за синхронизации коллекция Hashtable имеет существенные проблемы с производительностью;
- в большинстве случаев рекомендуется использовать коллекцию Hashtable.

443. Результат работы программы?

```
import java.util.*;
public class Program
{
   public static void main (String args[]) {
     Hashtable ht = new Hashtable();
     ht.put("О", 6); ht.put("С", 9);
     ht.put("П", 5); ht.put("Д", 5);
     System.out.println(ht.get("П"));
   }
}
```

444. Результат работы программы?

```
import java.util.*;
public class Program
{
   public static void main (String args[]) {
     Hashtable ht = new Hashtable();
     ht.put("О", 6); ht.put("С", 9);
     ht.put("П", 5); ht.put("Д", 5);
     System.out.println(ht.containsValue(0));
   }
}
```

445. Результат работы программы?

```
import java.util.*;
public class Program
{
   public static void main (String args[]) {
     Hashtable ht = new Hashtable();
     ht.put("О", 6); ht.put("С", 9);
     ht.put("П", 5); ht.put("Д", 5);
     System.out.println(ht.containsKey(0));
   }
}
```

446. Выберите верные утверждения?

- коллекция HashMap является альтернативой HashTable.

- у HashMap выше скорость выполнения операций чем у HashTable;
- HashMap позволяет использовать null как в качестве *ключа*, так и *значения*;
- в HashMap для null-ключа hashCode() всегда равен нулю;
- в HashMap порядок итерирования не совпадает с порядком добавления записей, но какой признак лежит в основе итерирования ***не*** установлено;
- добавление элемента выполняется за константное время O(1), но время удаления, получения зависит от распределения хэш-функции - в идеале является константным, но может быть и линейным O(n);
- HashMap является упорядоченной: порядок хранения элементов зависит от хэш-функции;
- HashMap допускает дублей ключей (в коллекции остается последний по порядку добавления элемент с дублирующимся ключом).

447. Выберите верные утверждения?

- в коллекции LinkedHashMap порядок итерирования равен порядку добавления элементов, но может быть изменен в конструкторе на порядок обращения методами get() и put();
- элементы объекта данной коллекции имеют двунаправленные ссылки на следующий и предыдущий, что увеличивает размер памяти, необходимой для хранения объекта коллекции LinkedHashMap по сравнению с объектом HashMap;
- LinkedHashMap выполняет базовые операции добавления, удаления и сохранения значений в объекте коллекции за постоянное время;
- допускает null в качестве ключа, а также значения;
- реализация LinkedHashMap синхронизирована.

448. Выберите верные утверждения?

- интерфейс Set расширяет интерфейс Collection;
- представляет неупорядоченный набор *уникальных* элементов.
- метод add() добавляет элемент в коллекцию и возвращает true, *только* если в коллекции еще нет такого элемента;
- интерфейс Set является программной моделью математического понятия «множество»;
- коллекции, реализующие интерфейс Set, содержит это слово в своем названии;
- интерфейс Set добавляет к методам интерфейса Collection новые методов;

Лямбда-выражения

449. Что такое лямбда-выражения?

- лямбда представляет набор инструкций, которые можно выделить в отдельный объект и затем многократно вызвать в различных местах программы;
- лямбда-выражение – это блок кода, который можно передать для выполнения позже, один раз или несколько раз;
- выражение с переменными параметрами;
- набор инструкций, объединенных в озаглавленный метод;
- набор инструкций, объединенных в специальный класс.

450. Что является основой синтаксиса лямбда-выражения?

- лямбда-оператор, который представляет стрелку ->;
- реализация метода, определенного в функциональном интерфейсе;
- функциональный интерфейс должен содержать единственный абстрактный метод;
- лямбда-выражение разделяется на две части: левая часть содержит список параметров выражения, а правая, собственно, представляет тело лямбда-выражения, в котором выполняются все действия;
- функциональный интерфейс, на основе которого создается лямбда выражение, может содержать несколько методов;
- абстрактный метод интерфейса для создания лямбда-выражения может содержать несколько параметров.

451. Результат работы программы?

```
interface Able {
    int function(int x, int y);
}

public class Demo
{
    public static void main(String[] args) {
        Able operation;
        operation = (x, y) -> x + y;
        int result = operation.function (3, 65);
        System.out.println(result);
   }
}
```

452. Какую синтаксическую конструкцию реализует следующий код?

```
interface Able {
    int function(int x, int y);
```

```
}

public class Demo
{
    public static void main(String[] args) {
        Able op = new Able() {
             public int function(int x, int y) {
                return x + y;
            }
        };
        int z = op.function(3, 65);
        System.out.println(z);
    }
}
```

- вложенный класс;
- анонимный класс;
- лямбда выражение.

453. Результат работы программы?

```
interface Able {
    int function(int x, int y);
}

public class Demo
{
   public static void main(String[] args) {
      Able operation1 = (x, y) -> x + y;
      Able operation2 = (int x, int y) -> x - y;
      Able operation3 = (x, y) -> x * y;
      int result = operation3.function(10, 35);
      System.out.println(result);
   }
}
```

454. При написании лямбда-выражения можно:

- использовать для параметров спецификатор final;
- использовать вызовы методов;
- использовать вызов конструктора класса;
- использовать операторы управления программой;
- использовать оператор return;
- изменить тип параметров, передаваемых в выражение;
- изменить тип возвращаемого значения.

455. Типы возвращаемого значения и параметров в лямбда-выражении определяются абстрактной функцией соответствующего интерфейса?

- да;

- нет;
- только в некоторых случаях.

456. Что такое отложенное выполнение в использовании лямбд (`deferred execution`)?

- в одном месте программы определяется лямбда-выражение и затем его можно вызывать при необходимости неопределенное количество раз в различных частях программы;
- возможность выполнить блок кода не в месте определения блока, а в более поздний момент времени работы программы;
- для выполнения лямбды необходимо использовать ключевое слово run.

457. Параметры лямбда-выражения:

- должны соответствовать по типу параметрам метода из функционального интерфейса;
- при написании самого лямбда-выражения тип параметров писать необязательно, однако это можно сделать;
- если метод не принимает никаких параметров, то пишутся пустые скобки;
- это конкретные числа.

458. Результат работы программы?

```
interface Able {
    int function();
}

public class Demo
{
    public static void main(String[] args) {
        Able operation;
        operation = () -> 110 / 10;
        int result = operation.function();
        System.out.println(result);
   }
}
```

459. Результат выполнения программы?

```
interface Able {
    int function(int n);
}

public class Demo
{
    public static void main(String[] args) {
```

```
            Able operation;
            operation = n -> n * n;
            int result = operation.function(3);
            System.out.println(result);
        }
    }
```

460. Что такое «терминальное лямбда-выражение»?

- иногда используемый в литературе термин, для обозначения лямбда-выражения, не возвращающего никакого значения;
- лямбда-выражения, прекращающие выполнение программы/метода.

461. Результат выполнения программы?

```
interface Able{
    void function(String s);
}

public class Demo
{
   public static void main(String[] args) {
      Able printer = s -> System.out.println(s);
      printer.function("String!");
   }
}
```

462. Какие типы лямбда-выражений существуют?

- однострочное выражение;
- блок кода;
- финальные.

463. Результат выполнения программы?

```
interface Able{
    int f(int x, int y);
}

public class Demo
{
    static int x = 10;
    static int y = 20;
    public static void main(String[] args) {
        Able z = (int x, int y)-> {
           if (y == 0) return 0;
           else        return x/y;
        };
        System.out.println(z.f(2,11)+" "+z.f(320,0));
    }
}
```

464. Лямбда-выражение может использовать:

- и переменные, которые объявлены на уровне класса, в котором лямбда-выражение определено;
- и переменные, которые объявлены на уровне метода, в котором лямбда-выражение определено;
- собственные локальные переменные, которые объявлены на уровне лямбда-выражения;
- только переменные уровня класса или метода;
- только локальные переменные, объявленные в самом лямбда-выражении.

465. Результат выполнения программы?

```
interface Able{
    int f();
}

public class Demo
{
    static int x = 10;
    static int y = 20;
    public static void main(String[] args) {
        Able z = () -> {
            x = 3;
            return x+y;
        };
        System.out.print(z.f() + " " + x);
    }
}
```

466. Результат работы программы?

```
interface Able {
    int f();
}

public class Demo
{
    public static void main(String[] args) {
    int n=70;
    int m=30;
    Able z = () -> {
        return m + n;
    };
```

```
            System.out.println(z.f());
            }
        }
```

467. Выберете верные утверждения о лямбда выражении. Значения переменных:

- уровня метода *можно* изменять внутри лямбда-выражения;
- уровня метода *нельзя* изменять внутри лямбда-выражения;
- уровня метода, используемую в лямбда-выражении *нельзя* изменять за предалами лямбда-выражения;
- уровня класса *можно* изменять внутри лямбда-выражения;
- уровня класса *нельзя* изменять внутри лямбда-выражения.

468. Выберите верные утверждения о лямбда-выражении:

- функциональный интерфейс, используемый для лямбда-выражения *не* может быть обобщенным;
- в лямбда-выражении *допускается* использование обобщений;
- в лямбда-выражении использование обобщений *не* допускается;
- функциональный интерфейс, *может* быть обобщенным, но необходимо типизировать объект, ссылающийся на интерфейс конкретным типом, который потом будет использоваться в лямбда-выражении.

469. Результат работы программы?

```
interface Able<T> {
    T f(T x, T y);
}

public class Demo
{
   public static void main(String[] args) {
      Able<Integer> z = (x, y)-> x + y;
      Able<String> p = (x, y) -> x + y;
      System.out.print(z.f(22, 1) + " ");
      System.out.println(p.f("21","11"));
   }
}
```

470. Результат работы программы?

```
interface Able {
    boolean f (int n);
}

public class Demo
{
```

```
    private static int sum(int[] a, Able func) {
        int result = 5;
        for(int i : a) {
            if (func.f(i)) result += i;
        }
        return result;
    }

    public static void main(String[] args) {
        Able z = (n) -> n%2 == 0;
        int[] nums = {1, 2, 3, 4, 5, 6, 7, 8, 9};
        System.out.println(sum(nums, z));
    }
}
```

471. Результат работы программы?

```
interface Able {
    boolean f(int n);
}

public class Demo
{
    private static int sum(int[] a, Able func) {
        int result = 3;
        for(int i : a) {
            if (func.f(i)) result += i;
        }
        return result;
    }

    public static void main(String[] args) {
        int[] nums = {1, 2, 3, 4, 5, 6, 7, 8, 9};
        //сумма чисел, со знач. больше 5
        int x = sum(nums, (n)-> n > 5);
        System.out.println(x);
    }
}
```

472. Результат работы программы?

```
import java.util.*;
public class Demo
{
   public static void main(String[] args) {
      ArrayList evens = new ArrayList();
      evens.add(1);
      evens.add("String");
      evens.add(3);
      evens.add(true);
```

```
        evens.forEach((n)->{System.out.print(n + " ");});
    }
}
```

473. Результат работы программы?

```
interface Able {
    boolean f(int n);
}

class Library {
    static boolean isSomething(int n){
        return n % 2 == 0;
    }
    static boolean isAnything(int n){
        return n > 0;
    }
}

public class Demo
{
   private static int sum (int[] array, Able func) {
      int result = 0;
      for(int i : array) {
          if (func.f(i)) result += i;
      }
      return result;
   }

   public static void main(String[] args) {
      int[] nums={ -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5};
      System.out.print(sum(nums, Library::isSomething));

      Able expr = Library::isAnything;
      System.out.println(" " + sum(nums, expr));
   }
}
```

474. Результат работы программы?

```
    import java.util.*;
    public class Demo
    {
       public static void main(String[] args) {
          ArrayList evens = new ArrayList();
          evens.add(51);
          evens.add("Str");
          evens.add(3);
          evens.add(true);
          evens.forEach(System.out::print);
       }
    }
```

475. Результат работы программы?

```
record User(String name) {}

interface UserBuilder {
    User f(String name);
}

public class Demo
{
    public static void main(String[] args) {
        UserBuilder ub = User::new;
        User user = ub.f("John");
        System.out.println(user.name());
    }
}
```

476. Результат работы программы?

```
interface Able{
    int f(int x, int y);
}

public class Demo
{
    private static Able action(int number){
        switch(number){
            case 1: return (x, y) -> x + y;
            case 2: return (x, y) -> x - y;
            default: return (x, y) -> 0;
        }
    }

    public static void main(String[] args) {
        Able func = action(1);
        int a = func.f(3, 2);
        System.out.println(a);
    }
}
```

477. Результат работы программы?

```
import java.util.function.Predicate;
public class Demo
{
   public static void main(String[] args) {
      Predicate<Integer> isProperty = x -> x > 0;
      System.out.print(isProperty.test(9));
   }
}
```

478. Результат работы программы?

```
import java.util.function.BinaryOperator;
public class Demo
{
   public static void main(String[] args) {
      BinaryOperator<Integer> f = (x, y) -> x*y;
      System.out.println(f.apply(2, 9));
   }
}
```

479. Результат работы программы?

```
import java.util.function.UnaryOperator;
public class Lambda
{
    public static void main(String[] args) {
        UnaryOperator<Integer> square =
                                    x -> x * x;
        System.out.println(square.apply(12));
    }
}
```

480. Результат работы программы?

```
import java.util.function.Function;
public class Demo
{
   public static void main(String[] args) {
     Function<Integer,String> f =
                    x -> String.valueOf(x)+ " штрафов";
     System.out.println(f.apply(5));
   }
}
```

481. Результат работы программы?

```
import java.util.function.Consumer;
public class Demo
{
    public static void main(String[] args) {
        Consumer<Integer> f =
        x-> System.out.printf("%d листков\n", x);
        f.accept(9);
    }
}
```

482. Результат работы программы?

```
import java.util.function.Supplier;

record User(String str) {}
```

```
public class Demo
{
    public static void main(String[] args) {
        Supplier<User> userFactory = ()-> {
            return new User("Flag");
        };
        User user = userFactory.get();
        System.out.println("Результат: " +
                           user.str());
    }
}
```

Потоки ввода-вывода

483. Выберите верные утверждения для потоков ввода-вывода:

- поток ввода-вывода - это абстракция, которая используется для чтения или записи информации;
- поток ввода-вывода всегда связан с реальным физическим устройством;
- может быть определен поток, который связан с файлом для чтения или записи информации;
- объект, из которого можно считать данные, называется потоком ввода;
- объект, в который можно записывать данные - потоком вывода.

484. Выберите абстрактные суперклассы управляющие потоком ввода-вывода:

- InputStream;
- OutputStream;
- Reader;
- Writer;
- FileOutputStream;
- FileInputStream;
- FileReader;
- FileWriter.

485. Имя метода закрывающего поток ввода-вывода?

486. Что записано в файле notes.txt в результате работы программы?

```
import java.io.*;
public class Main
```

```
{
   public static void main(String[] args) {
      String text = "Bill Gates!";
      try(FileOutputStream fos =
            new FileOutputStream("notes.txt")) {
            byte[] buffer = text.getBytes();
            fos.write(buffer, 0, buffer.length);
        }
        catch(IOException ex){
            System.out.println(ex.getMessage());
        }
    }
}
```

487. Что записано в файле notes.txt в результате работы программы?

```
import java.io.*;
public class Main
{
   public static void main(String[] args) {
      String text = "Bill Gates!";
      try(FileOutputStream fos =
            new FileOutputStream("notes.txt")) {
            byte[] buffer = text.getBytes();
            fos.write(buffer, 6, 1);
        }
        catch(IOException ex){
            System.out.println(ex.getMessage());
        }
    }
}
```

488. Что записано в файле notes.txt в результате работы программы?

```
import java.io.*;
public class Main
{
   public static void main(String[] args) {
      String text = "Elon Reeve Musk!";
      try(FileOutputStream fos =
            new FileOutputStream("notes.txt")) {
            byte[] buffer = text.getBytes();
            fos.write(buffer[4]);
        }
        catch(IOException ex){
```

```
            System.out.println(ex.getMessage());
         }
     }
}
```

489. Файл notes.txt содержит фразу "Bill Gates!" Результат работы программы?

```
import java.io.*;
public class Main
{
   public static void main(String[] args) {
      try(FileInputStream fin =
                    new FileInputStream("notes.txt")) {
         String result = " ";
         System.out.print(fin.available());
         int i = -1;
         while((i = fin.read()) != -1){
             result = result + (char) i;
         }
         System.out.print(result);
      }
      catch(IOException ex){
         System.out.println(ex.getMessage());
      }
   }
}
```

490. Файл notes.txt содержит фразу "Elon Reeve Musk!" Результат работы программы?

```
import java.io.*;
public class Main
{
   public static void main(String[] args) {
      try(FileInputStream fin =
                    new FileInputStream("notes.txt")) {
         byte[] buffer = new byte[fin.available()];
         fin.read(buffer, 0, fin.available());
         for(int i = 0; i < buffer.length; i++){
               System.out.print((char) buffer[i]);
         }
      }
      catch(IOException ex){
         System.out.println(ex.getMessage());
      }
   }
}
```

491. Файл notes.txt содержит фразу "Bill Gates!" Результат работы программы?

```
import java.io.*;
public class Main
{
   public static void main(String[] args) {
      try(FileInputStream fin =
                  new FileInputStream("notes.txt")) {
         byte[] buffer = new byte[fin.available()];
         fin.read(buffer, 0, fin.available());
         for(int i = 3; i < buffer.length - 2; i++){
               System.out.print((char) buffer[i]);
         }
      }
      catch(IOException ex){
         System.out.println(ex.getMessage());
      }
   }
}
```

492. Файл notes.txt содержит фразу "Elon Reeve Musk!" Результат работы программы?

```
import java.io.*;
import java.util.*;
public class Main
{
   public static void main(String[] args)
                                  throws IOException {
       Scanner in =
        new Scanner(new FileInputStream("notes.txt"));
       String s = in.nextLine();
       System.out.print(s);
       in.close();
   }
}
```

493. Файл notes.txt содержит фразу "Bill Gates!" Результат работы программы?

```
import java.io.*;
import java.util.*;
public class Main
{
   public static void main(String[] args)
                                 throws IOException {
```

```
        Scanner in =
          new Scanner(new FileInputStream("notes.txt"));
        System.out.print(in.nextLine());
        in.close();
    }
}
```

494. Файл notes.txt содержит фразу "Elon Musk!". Какую информацию содержит файл notes_new.txt после работы программы?

```
import java.io.*;
public class Main
{
   public static void main(String[] args) {
      try(FileInputStream fin =
                new FileInputStream("notes.txt");
          FileOutputStream fos =
                new FileOutputStream("notes_new.txt")) {
         byte[] buffer = new byte[fin.available()];
         fin.read(buffer, 0, buffer.length);
         fos.write(buffer, 0, buffer.length);
      }
      catch(IOException ex){
         System.out.println(ex.getMessage());
      }
   }
}
```

495. Верно ли утверждение: «Необходимо явно закрыть поток или потоки, открытые в конструкции `try-with-resources`**.»**

- да;
- нет;

496. Укажите конструкцию `try-with-resources`**.**

```
try { fin = new FileInputStream("notes.txt");
    //блок try
}
try(FileInputStream fin = new
                    FileInputStream("notes.txt")) {
    //блок try
}
try(FileInputStream fin =
                new FileInputStream("notes.txt");
    FileOutputStream fos =
                new FileOutputStream("notes_new.txt")) {
        //блок try
}
```

497. Сколько ресурсов можно перечислить в конструкции try-with-resources?

- один;
- два;
- неограниченное число.

498. Исключает ли конструкции try-with-resources использование блока catch()?

- да;
- нет;

499. Результат работы программы?

```
import java.io.*;
import java.util.Scanner;
public class Main
{
   public static void main(String[] args) {
      Scanner scanner =
                  new Scanner("раз\nдва\nтри\n");
      while (scanner.hasNextLine()) {
         System.out.println(scanner.nextLine());
      }
      scanner.close();
   }
}
```

Ответы:

- раз
  два
  три
- раз
- два
- три
- раз
  два

500. Результат работы программы?

```
import java.io.*;
import java.util.Scanner;
public class Main
{
   public static void main(String[] args) {
      String init = "раз \n два \n три \n";
```

```
      Scanner scanner =
        new Scanner(init).useDelimiter("\\s*\n\\s*");
      while (scanner.hasNext()) {
        System.out.print(scanner.next() + " ");
      }
      scanner.close();
   }
}
```

501. Результат работы программы?

```
import java.io.*;
import java.util.Scanner;
public class Main
{
   private static void createFile(String str,
                                  String fileName) throws
                                  IOException {
      FileWriter fw = new FileWriter(fileName);
      fw.write(str);
      fw.close();
   }
   private static String readFile(String fileName) throws
                                  IOException {
      FileReader fr = new FileReader(fileName);
      Scanner scan = new Scanner(fr);
      String str = scan.nextLine();
      scan.close();
      fr.close();
      return str;
   }
   public static void main(String[] args) throws
                                            IOException {
      createFile("Привет\n Мир", "file.txt");
      String result = readFile("file.txt");
      System.out.println(result);
   }
}
```

502. Укажите размер используемого в программе буфера в килобайтах?

```
import java.io.*;
public class Main
{
   public static void main(String[] args) {
      String text = "Buffered: Elon Musk!";
      try(FileOutputStream out =
                new FileOutputStream("notes.txt");
          BufferedOutputStream bos =
```

```
                    new BufferedOutputStream(out))
      {
            byte[] buffer = text.getBytes();
            bos.write(buffer, 0, buffer.length);
       }
        catch(IOException ex){
            System.out.println(ex.getMessage());
        }
    }
}
```

503. Укажите размер используемого в программе буфера в килобайтах?

```
import java.io.*;
public class Main
{
   public static void main(String[] args) {
     try(FileInputStream fin =
                    new FileInputStream("notes.txt");
         BufferedInputStream br =
                 new BufferedInputStream(fin, 20480))
     {
        System.out.printf("Размер файла: %d байт \n",
                          br.available());
        int i = -1;
        while((i = br.read()) != -1){
           System.out.print((char) i);
        }
     }
     catch(IOException ex) {
         System.out.println(ex.getMessage());
     }
   }
}
```

504. Выберите верные утверждения:

- сериализация – это процесс перевода какой-либо структуры данных в последовательность битов;
- десериализации - восстановление начального состояния структуры данных из битовой последовательности;
- сериализация предшествует записи состояния объекта в поток;
- десериализация проводится в процессе восстановления состояния объекта из потока.

505. Выберите верные утверждения:

- сериализовать можно только те объекты, которые реализуют интерфейс `Serializable`;
- интерфейс `Serializable` определяет методы сериализации;
- интерфейс `Serializable` указывает системе, что объект класса, реализующий его, может быть сериализован;
- десериализация – это обратный процесс чтения ранее сериализованных данных из потока
- «по умолчанию» сериализуются все переменные объекта;
- невозможно исключить поля объекта из сериализации.

506. Выберете варианты правильного синтаксиса применения модификатора transient:

- ```
  class Person implements Serializable {
  private String name;
  private transient int age;
      //методы
  }
  ```
- ```
  class transient Person implements Serializable {
  private String name;
  private transient int age;
      //методы
  }
  ```
- ```
  class Person<transient> implements Serializable {
  private String name;
  private transient int age;
      //методы
  }
  ```

507. Выберите правильные утверждения о классе File:

- он определен в пакете `java.io`;
- он работает напрямую с потоками;
- его задачей является только управление информацией о файлах и каталогах;
- файлы и каталоги в Java не описываются одним классом `File`.

508. Может ли объект File соответствовать файлу, которого еще нет?

- не может;
- всегда может;
- может, если передать в конструктор класса `File` значение директории:
  ```

```
String dirPath = "/";
File f = new File(dirPath);
File[] files = f.listFiles().
```

509. Зачем нужен класс FileReader?

- предназначен для чтения потоков символов;
- конструкторы этого класса предполагают, что принимается кодировка символов «по умолчанию»;
- с помощью конструкторов этого класса можно изменить размер байтового буфера.

510. Зачем нужен класс FileWriter?

- предназначен для записи потоков символов;
- с помощью конструкторов этого класса можно установить кодировку символов по выбору;
- размер байтового буфера всегда равен размеру «по умолчанию».

Stream API

511. Выберите верные утверждения о `Stream API`:

- его назначение - упростить работу с наборами данных, в частности, упростить операции фильтрации, сортировки и др.;
- вся основная функциональность `Stream API` сосредоточена в пакете `java.util.stream`;
- применительно к `Stream API` поток представляет канал передачи данных из источника данных, причем в качестве источника могут выступать массивы и коллекции;
- при использовании `Stream API` вместо цикла и условных конструкций можно записать цепочку вызовов методов, которые будут выполнять те же действия;
- потоки производят вычисления в промежуточных операциях, только тогда, когда к ним применяется терминальная операция. (отложенное выполнение);
- применительно к `Stream API` источником данных, является исключительно файл;
- в `Stream API` *не* применяются лямбда-выражения.

512. Результат работы программы?

```
import java.util.stream.*;
public class MyClass {
    public static void main(String args[]) {
```

```
        long s = IntStream.of(-2, -1, 0, 1, 2)
                          .filter(w-> w>0)
                          .count();
        System.out.println(s);
    }
}
```

513. Какие бывают операции, реализуемые методами, работе со Stream API?

- терминальные (`terminal`);
- промежуточные (`intermediate`);
- конечные (`final`).

514. Продолжите фразу: «промежуточная операция…»

- возвращает преобразованный поток и к нему также можно применить ряд промежуточных операций;
- возвращают конкретный результат и после нее промежуточных операций применять нельзя.

515. Продолжите фразу: «терминальная операция…»

- возвращает преобразованный поток и к нему также можно применить ряд промежуточных операций;
- возвращают конкретный результат и после нее промежуточных операций применять нельзя.

516. Результат работы программы?

```
import java.util.stream.Stream;
import java.util.*;
public class Main
{
   public static void main(String[] args) {
      ArrayList<String> a = new ArrayList();
      Collections.addAll(a,
                         "Париж", "Лондон", "Мадрид");
      Stream s = a.stream();
      s.forEach(t->System.out.print(t + " "));
    }
}
```

517. Результат работы программы?

```
import java.util.stream.Stream;
import java.util.*;
public class Main
{
   public static void main(String[] args) {
      ArrayList<String> a = new ArrayList();
```

```
      Collections.addAll(a, "Москва", "Лондон");
      Stream s = a.stream();
      s.forEach(System.out::println);;
    }
}
```

518. Результат работы программы?

```
import java.util.stream.Stream;
import java.util.*;
public class Main
{
   public static void main(String[] args) {
      ArrayList<String> a = new ArrayList();
      Collections.addAll(a,
                     "Вильнюс", "Минск", "Вена");
      Stream<String> s = a.stream();
      s = s.filter(t -> t.length() == 6);
      s.forEach(System.out::println);
    }
}
```

519. Результат работы программы?

```
import java.util.stream.Stream;
import java.util.*;
public class Main
{
   public static void main(String[] args) {
      ArrayList<String> a = new ArrayList();
      Collections.addAll(a,
                      "Копенгаген", "Рим", "Прага");
      a.stream().filter(s -> s.length() == 6)
       .forEach(System.out::println);
    }
}
```

520. Результат работы программы?

```
import java.util.stream.Stream;
import java.util.*;
public class Main
{
   public static void main(String[] args) {
      ArrayList<String> s = new ArrayList();
      Collections.addAll(s,
                     "Брюссель", "Лиссабон", "Мадрид");
      s.stream().filter(t -> t.length()==6)
       .forEach(System.out::println);
```

```
    }
}
```

521. Какие стадии проходит жизненный цикл потока?

- создание потока;
- применение к потоку ряда промежуточных операций;
- применение к потоку терминальной операции с одновременным получением результата и закрытия потока;
- применение к потоку финальной операции с одновременным получением результата и закрытия потока.

522. Результат работы программы?

```
import java.util.stream.Stream;
import java.util.*;
public class Main
{
   public static void main(String[] args) {
      Stream<String> s =
               Arrays.stream(new String[]{"Париж",
                              "Лондон", "Мадрид"}) ;
      s.filter(z -> z.length()==5)
        .forEach(System.out::println);
    }
}
```

523. Результат работы программы?

```
import java.util.stream.*;
import java.util.*;
public class Main
{
   public static void main(String[] args) {
      IntStream s =
                Arrays.stream(new int[]{6,3,1,5,7});
      s.filter(w-> w > 4 && w < 7)
       .forEach(System.out::println);
    }
}
```

524. Результат работы программы?

```
import java.util.stream.*;
import java.util.*;
public class Main
{
   public static void main(String[] args) {
      LongStream s =
```

```
                Arrays.stream(new long[]{0,25,3});
      s.sorted().forEach(System.out::print);
    }
}
```

525. Результат работы программы?

```
import java.util.stream.*;
import java.util.*;
public class Main
{
   public static void main(String[] args) {
      IntStream s = IntStream.of(9, 5, 1, 0);
      System.out.println(s.max());
    }
}
```

526. Результат работы программы?

```
import java.util.stream.*;
import java.util.*;
public class Main
{
   public static void main(String[] args) {
      IntStream s = IntStream.of(11, 12, 0, 5);
      System.out.println(s.max().getAsInt());
    }
}
```

527. Результат работы программы?

```
import java.util.stream.*;
import java.util.*;
public class Main
{
   public static void main(String[] args) {
      Stream<String> s =
             Stream.of("Москва", "Вильнюс", "Рига");
      System.out.println(s.count());
    }
}
```

528. Результат работы программы?

```
import java.util.stream.*;
import java.util.*;
public class Main
{
   public static void main(String[] args) {
```

```
      String[] a = {"Будапешт", "Варшава", "Берлин"};
      Stream<String> b = Stream.of(a);
      b.distinct()
       .forEach(s -> System.out.print(s + " "));
    }
}
```

529. Результат работы программы?

```
import java.util.stream.*;
import java.util.*;
public class Main
{
   public static void main(String[] args) {
      LongStream a =
              LongStream.of(10, 25, 10, 96, 50);
      a.distinct()
       .forEach(s -> System.out.print(s + " "));
    }
}
```

530. Результат работы программы?

```
import java.util.stream.*;
import java.util.*;
public class Main
{
   public static void main(String[] args) {
      DoubleStream a =
              DoubleStream.of(5, 5, 5, 3, 2);
      System.out.print(a.average());
    }
}
```

531. Результат работы программы?

```
import java.util.stream.*;
import java.util.*;
public class Main
{
   public static void main(String[] args) {
      DoubleStream a =
              DoubleStream.of(5, 5, 5, 3, 2);
      System.out.print(a.average().getAsDouble());
    }
}
```

532. Результат работы программы?

```
import java.util.stream.Stream;
import java.util.*;

record UserObj(String s, int n) {}

public class Main
{
   public static void main(String[] args) {
   Stream<UserObj> p =
          Stream.of(new UserObj("iPhone 6 S", 400),
                    new UserObj("Lumia 950", 500),
                new UserObj("Samsung Galaxy", 1000));
   p.filter(w -> w.n()<50000 && w.n()> 500)
    .forEach(w ->
       System.out.println(w.s().replace(" ","")));
   }
}
```

533. Результат работы программы?

```
import java.util.stream.Stream;
import java.util.*;

record UserObj (String s, int n) {}

public class Main
{
   public static void main(String[] args) {
     Stream<UserObj> s =
          Stream.of(new UserObj("iPhone 6 S", 400),
                    new UserObj("Lumia 950", 500),
                new UserObj("Samsung Galaxy", 5000));
     var z = s.map(p -> p.s().substring(8).trim());
         z.forEach(System.out::print);
   }
}
```

534. Результат работы программы?

```
import java.util.stream.Stream;
import java.util.*;

record UserObj(String s, int n) {}

public class Main
```

```
{
   public static void main(String[] args) {
     Stream<UserObj> s =
         Stream.of(new UserObj("iPhone 6 S", 200),
                   new UserObj("Lumia 950", 100),
                  new UserObj("Samsung Galaxy", 1000));
     var z = s.map(p->p.n()).filter(p-> p>100);
         z.forEach(System.out::print);
   }
}
```

535. Результат работы программы?

```
import java.util.stream.Stream;
import java.util.*;

record UserObj(String s, int n) {}

public class Main
{
   public static void main(String[] args) {
     Stream<UserObj> s =
         Stream.of(new UserObj("D", 10),
                   new UserObj("G", 15),
                   new UserObj("B", 20));
     s.map(p -> p.s()).sorted()
      .forEach(System.out::print);
   }
}
```

536. Результат работы программы?

```
import java.util.stream.Stream;
import java.util.*;

record UserObj(String s, int n) {}

public class Main
{
   public static void main(String[] args) {
     Stream<UserObj> s =
         Stream.of(new UserObj("W", 20),
                   new UserObj("Q", 10),
                   new UserObj("S", 15));
     s.map(p -> p.n()).sorted()
      .forEach(System.out::print);
   }
```

```
}
```

537. Результат работы программы?

```
import java.util.stream.Stream;
import java.util.*;

record UserObj(String s, int n, double x) {}

class UserComp implements Comparator<UserObj> {
   public int compare(UserObj a, UserObj b) {
     return a.s().toUpperCase()
                 .compareTo(b.s().toUpperCase());
   }
}

public class Main
{
   public static void main(String[] args) {
     Stream<UserObj> s =
         Stream.of(new UserObj("Q", 15, 0.3),
                   new UserObj("G", 35, 0.2),
                   new UserObj("J", 25, 0.1));
     s.sorted(new UserComp())
      .forEach(System.out::println);
   }
}
```

538. Результат работы программы?

```
import java.util.stream.Stream;
import java.util.*;

record UserObj(String s, int n, double x) {}

class UserComp implements Comparator<UserObj> {
   public int compare(UserObj a, UserObj b) {
     return -a.s().toUpperCase()
                 .compareTo(b.s().toUpperCase());
   }
}

public class Main
{
   public static void main(String[] args) {
     Stream<UserObj> s =
         Stream.of(new UserObj("U", 15, 0.3),
                   new UserObj("S", 35, 0.2),
                   new UserObj("B", 25, 0.1));
```

```
     s.sorted(new UserComp())
      .forEach(System.out::println);
   }
}
```

539. Результат работы программы?

```
import java.util.stream.Stream;
import java.util.*;

record UserObj(String s, int n, double x) {}

class UserComp implements Comparator<UserObj> {
   public int compare(UserObj a, UserObj b) {
     return Integer.compare(a.n(), b.n());
   }
}

public class Main
{
   public static void main(String[] args) {
     Stream<UserObj> s =
         Stream.of(new UserObj("F", 15, 0.3),
                   new UserObj("D", 35, 0.2),
                   new UserObj("V", 25, 0.1));
     s.sorted(new UserComp())
      .forEach(System.out::println);
   }
}
```

540. Результат работы программы?

```
import java.util.stream.Stream;
import java.util.*;

record UserObj(String s, int n, double x) {}

class UserComp implements Comparator<UserObj> {
   public int compare(UserObj a, UserObj b) {
     return -Integer.compare(a.n(), b.n());
   }
}

public class Main
{
   public static void main(String[] args) {
     Stream<UserObj> s =
         Stream.of(new UserObj("FU", 15, 0.3),
                   new UserObj("SU", 35, 0.2),
                   new UserObj("BU", 25, 0.1));
     s.sorted(new UserComp())
```

```
      .forEach(System.out::println);
   }
}
```

541. Результат работы программы?

```
import java.util.stream.Stream;
import java.util.*;

record UserObj(String s, int n, double x) {}

class UserComp implements Comparator<UserObj> {
   public int compare(UserObj a, UserObj b) {
     return Double.compare(a.x(), b.x());
   }
}

public class Main
{
   public static void main(String[] args) {
     Stream<UserObj> s =
         Stream.of(new UserObj("AD", 15, 0.3),
                   new UserObj("CV", 35, 0.2),
                   new UserObj("SF", 25, 0.1));
     s.sorted(new UserComp())
      .forEach(System.out::println);
   }
}
```

Ответы:

- UserObj[s=U, n=15, x=0.3]
  UserObj[s=B, n=25, x=0.1]
  UserObj[s=S, n=35, x=0.2]
- UserObj[s=B, n=25, x=0.1]
  UserObj[s=U, n=15, x=0.3]
  UserObj[s=S, n=35, x=0.2]
- UserObj[s=B, n=25, x=0.1]
  UserObj[s=S, n=35, x=0.2]
  UserObj[s=U, n=15, x=0.3]

542. Результат работы программы?

```
import java.util.stream.Stream;
public class Program
{
   public static void main(String[] args) {
      Stream <Integer> a =
                      Stream.of(-3,-2,0,1,-4,2,3,-1);
        a.takeWhile(n -> n < 0)
         .forEach(n->System.out.print(n + " "));
```

```
   }
}
```

543. Результат работы программы?

```
import java.util.stream.Stream;
public class Program
{
   public static void main(String[] args) {
      Stream <Integer> a =
                     Stream.of(-3,-5, 0, 1,-9, 2);
        a.dropWhile(n -> n < 0)
         .forEach(n->System.out.print(n + " "));
   }
}
```

544. Результат работы программы?

```
import java.util.stream.Stream;
public class Program
{
   public static void main(String[] args) {
      Stream<String> a = Stream.of("DB", "US");
      Stream<String> b = Stream.of("ET", "SU");
      Stream.concat(a, b)
            .forEach(System.out::print);
    }
}
```

545. Результат работы программы?

```
import java.util.stream.Stream;
public class Program
{
   public static void main(String[] args) {
      Stream<String> people =
                Stream.of("TB", "BL", "SI", "TI", "SE");
      people.distinct().forEach(System.out::print);
    }
}
```

546. Результат работы программы?

```
import java.util.*;
public class Main
{
   public static void main(String[] args) {
      List<String> l =
              Arrays.asList("GL", "RL", "VS", "UL", "BN");
```

```
       l.stream().skip(1).limit(2)
        .forEach(System.out::print);
    }
}
```

547. Результат работы программы?

```
import java.util.*;
public class Program
{
    public static void main(String[] args) {
        ArrayList<String> a = new ArrayList<String>();
        a.addAll(Arrays.asList("TE", "US", "BB", "AA"));
        boolean b = a.stream()
                       .anyMatch(s->s.length() > 3);
        System.out.println(b + " ");
        b = a.stream().allMatch(s->s.length() == 3);
        System.out.println(b + " ");
        b = a.stream().noneMatch(s->s == "B");
        System.out.println(b + " ");
    }
}
```

548. Результат работы программы?

```
import java.util.*;
public class Program
{
   public static void main(String[] args) {
      ArrayList<String> a = new ArrayList();
      a.addAll( Arrays.asList("TR", "SS", "BF", "FA") );
      System.out.print(a.stream().count() +
                      a.stream().filter(n->n.length()==2)
                       .count() );
   }
}
```

549. Результат работы программы?

```
import java.util.*;
public class Program
{
   public static void main(String[] args) {
      ArrayList<String> a = new ArrayList();
      a.addAll( Arrays.asList("WT", "DS", "RB", "FA") );
      System.out.println(a.stream().findFirst());
    }
}
```

550. Результат работы программы?

```
import java.util.*;
```

```
public class Program
{
   public static void main(String[] args) {
      ArrayList<String> a = new ArrayList();
      a.addAll( Arrays.asList("TEWS", "BQ", "A") );
      System.out.println(a.stream().findFirst().get());
    }
}
```

551. Результат работы программы?

```
import java.util.*;
public class Program
{
   public static void main(String[] args) {
      ArrayList<Integer> numbers = new ArrayList();
      numbers.addAll(Arrays.asList(10,12,3,4,5));
      Optional<Integer> v =
           numbers.stream().min(Integer::compare);
      if(v.isPresent()){
          System.out.println(v.get());
      }
   }
}
```

552. Результат работы программы?

```
import java.util.*;
import java.util.stream.Collectors;
public class Program
{
   public static void main(String[] args) {
      Set something = new HashSet();
      Collections.addAll(something, "P", "H", false,
                         "S9", 2, "X", true, 3.14);
      List result = (List) something.stream()
                .filter(s -> s.toString().length()<2)
                .collect(Collectors.toList());
      result.forEach(System.out::print);
   }
}
```

553. Результат работы программы?

```
import java.util.*;
import java.util.stream.Collectors;
import java.util.stream.Stream;

record UserObj(String str) {
```

```
    public String userGet() {return str;}
}

public class Program
{
   public static void main(String[] args) {
      Stream<UserObj> stream = Stream.of(
               new UserObj("Tom"),
               new UserObj("John"),
               new UserObj("Ivan"),
               new UserObj("Serg") );
      HashSet result = stream
         .filter(s -> s.str().length() < 4)
         .collect(Collectors.toCollection(HashSet::new));
      result.forEach(s->System.out.print(((UserObj) s)
                                          .userGet()));
   }
}
```

554. Результат работы программы?

```
import java.util.*;
import java.util.stream.Collectors;
import java.util.stream.Stream;

record UserObj(String str) {}

public class Program
{
   public static void main(String[] args) {
      Stream<UserObj> stream = Stream.of(
               new UserObj("Serg"),
               new UserObj("Bill"),
               new UserObj("Ilon"),
               new UserObj("Mask") );
      HashSet result = stream
         .filter(s -> s.name().length() < 4)
         .collect(Collectors.toCollection(HashSet::new));
      result.forEach(s -> System.out.print(s));
   }
}
```

Литература

1. Язык Java. Основы синтаксиса / А.С. Кравчук, А.И. Кравчук, Е.В. Кремень [Электронный документ] URL: https://elib.bsu.by/handle/123456789/290065 (Дата доступа: 06.12.2022)
2. Язык Java. Операторы управления программой / А.С. Кравчук, А.И. Кравчук, Е.В. Кремень [Электронный документ] URL: https://elib.bsu.by/handle/123456789/290066 (Дата доступа: 06.12.2022)
3. Язык Java. Массивы, строки, классы обертки базовых типов / А.С. Кравчук, А.И. Кравчук, Е.В. Кремень [Электронный документ] URL: https://elib.bsu.by/handle/123456789/290067 (Дата доступа: 06.12.2022)
4. Язык Java. Методы, стек и куча / А.С. Кравчук, А.И. Кравчук, Е.В. Кремень [Электронный документ] URL: https://elib.bsu.by/handle/123456789/290068 (Дата доступа: 06.12.2022)
5. Язык Java. Общие сведения / А. С. Кравчук, А. И. Кравчук, Е. В. Кремень. [Электронный документ] URL: https://elib.bsu.by/handle/123456789/291460.
6. Язык Java. Коллекции, реализующие интерфейсы Queue, Deque, Map, Set / А. С. Кравчук, А. И. Кравчук, Е. В. Кремень [Электронный документ] URL: https://elib.bsu.by/handle/123456789/291423.
7. Язык Java. Общие сведения о коллекциях и реализуемых ими интерфейсах. Коллекции, реализующие интерфейс List / А. С. Кравчук, А. И. Кравчук, Е. В. Кремень. [Электронный документ] URL: https://elib.bsu.by/handle/123456789/291422.

Учебное издание

Кравчук Александр Степанович
Кравчук Анжелика Ивановна
Кремень Елена Васильевна

ЯЗЫК JAVA.
СИСТЕМА ТЕСТОВ

Учебные материалы
для студентов специальности 1-31 03 08
«Математика и информационные технологии
(по направлениям)»

В авторской редакции

Ответственный за выпуск *Е. В. Кремень*

Подписано в печать 20.02.2023. Формат 60×84/16. Бумага офсетная.
Усл. печ. л. 8,37. Уч.- изд. л. 8,21. Тираж 50 экз. Заказ

Белорусский государственный университет.
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий № 1/270 от 03.04.2014.
Пр. Независимости 4, 220030, Минск.

Отпечатано с оригинал-макета заказчика
на копировально-множительной технике
механико-математического факультета
Белорусского государственного университета.
Пр. Независимости 4, 220030, Минск.