


 **trueddd / UniversityExams** Public[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#) master ▾

...

[UniversityExams](#) / [spp](#) / [spp.md](#)**trueddd** string shit cleaned up ✓ History 13 contributors 1206 lines (1122 sloc) | 109 KB

...

1

1. Just-In-Time-компилятор позволяет

JIT-компиляция (англ. Just-in-time compilation, компиляция «на лету»), динамическая компиляция (англ. dynamic translation) — технология увеличения производительности программных систем, использующих байт-код, путём компиляции байт-кода в машинный код или в другой формат непосредственно во время работы программы.

2. Для разработки программ на языке Java необходима установка

JDK

3. Запуск Java-программы производится

Чтобы запустить программу, нужно ввести команду `java` с именем класса (не файла!) в качестве параметра: `java Main` И чтобы запустить программу с аргументами, пишем в консоль: `java Main arg0 arg1 arg2`

4. Использование языковой виртуальной машины позволяет добиться

Кроссплатформенности

5. Компиляция исходного кода производится

Для того, чтобы скомпилировать файл нужно набрать в консоли команду `javac` и в качестве параметра передать имя нашего файла. Например, `javac Main.java`. Эта команда вызовет компилятор, который создаст файл `Main.class`, содержащий скомпилированный код нашей `java` программы.

6. Особенностью языка программирования Java является

- Отсутствие множественного наследования
- Интерфейсы
- Перечисления
- Поддержка механизма обработки исключений
- Обобщенные (generics) классы
- Аннотации
- Средства параллельного программирования
- Поддержка лямбда-выражений (начиная с 8 в.)

7. Пакеты служат для

Пакет (package) — это некий контейнер, который используется для того, чтобы изолировать имена классов. Например, вы можете создать класс `List`, заключить его в пакет и не думать после этого о возможных конфликтах, которые могли бы возникнуть если бы кто-нибудь еще создал класс с именем `List`. Пакеты — это механизм, который служит как для работы с пространством имен, так и для ограничения видимости.

8. Сигнатура функции `main` имеет вид

- `main(String[] args)` - сигнатура
- `public static void main(String[] args)` - объявление

9. Синтаксис языка программирования Java схож с синтаксисом языка

`C#`

10. Структурными частями простейшего Java-проекта являются

Структура простейшего проекта в IntelliJ IDEA:

- папка `.idea` с конфигурационными `xml` файлами

- папка `src` в которой находятся Java-классы, один из которых содержит функцию `main`.
- файл `.iml`

Структура проекта:

- `src` - исходный код
- `lib` - библиотеки
- `res` - прочие ресурсы
- `test` - исходный код тестов

11. Точкой входа в любую Java-программу является

```
main(String[] args)
```

12. Язык программирования Java является

- Языком высокого уровня
- Императивным
- Со строгой статической типизацией
- Объектно-ориентированный

2

13. В выражении, содержащем переменные типа `byte`, `short`, `char` происходит автоматическое повышение типа до

```
int
```

14. В выражении, содержащем переменные типа `double` и `long`, происходит автоматическое повышение типа до

```
double
```

15. В выражении, содержащем переменные типа `float` и `double`, происходит автоматическое повышение типа до

```
double
```

16. В выражении, содержащем переменные типа `float`, происходит автоматическое повышение типа до

float

17. В выражении, содержащем переменные типа `int` и `long`, происходит автоматическое повышение типа до

`long`

18. К классам-оберткам над элементарными типами данных не относится

Относится: `Double`, `Float`, `Long`, `Integer`, `Short`, `Byte`, `Character` и `Boolean`

19. К типам с плавающей запятой относится тип

`float`, `double`

20. К целочисленным типам данных Java не относится тип

Относится: `long`, `int`, `short`, `byte`

3

21. В Java ссылки всегда передаются

объекты

22. Значение `null` используется в Java

значение по умолчанию для ссылочных типов

23. Объект может быть передан по ссылке

передается по ссылке, но не влияет на сам объект

24. Объект называется достижимым, если

если на него ссылается другой достижимый объект.

25. Преобразование типа переменной подкласса к типу суперкласса

происходит автоматически

26. Преобразование типа переменной суперкласса к типу подкласса

требует явного преобразования

27. Приведение ссылочных типов возможно

если классы в одной иерархии

28. Прямое присваивание одной ссылке значения другой ссылки приведет к тому

переприсваивание ссылки

4

29. Для представления символов в Java используется кодировка

UTF-16 Из лекции: Для представления символов используется кодировка Unicode.

30. Изменение отдельного символа строки

невозможно, если не использовать `StringBuilder`

31. Получение отдельного символа строки

```
String.charAt()
```

32. При именовании переменных в Java не допускаются

ключевые или зарезервированные слова, пробелы, не начинать с цифр, Может состоять из букв (Unicode), цифр и символа подчеркивания «_».

33. Строковые литералы в Java являются

набор символов в двойных кавычках Из лекции: Строковые литералы (Пример: "abc") - объекты

34. Тип `char` относится к

примитивным типам Из лекции: `char` – формально целочисленный, но используется только для хранения символов.

35. У переменных типа `String` нет встроенных функций, поддерживающих следующую операцию над строками

основные методы

- `concat()` : объединяет строки
- `valueOf()` : преобразует объект в строковый вид
- `join()` : соединяет строки с учетом разделителя
- `compare()` : сравнивает две строки
- `charAt()` : возвращает символ строки по индексу
- `getChars()` : возвращает группу символов
- `equals()` : сравнивает строки с учетом регистра
- `equalsIgnoreCase()` : сравнивает строки без учета регистра
- `regionMatches()` : сравнивает подстроки в строках
- `indexOf()` : находит индекс первого вхождения подстроки в строку
- `lastIndexOf()` : находит индекс последнего вхождения подстроки в строку
- `startsWith()` : определяет, начинается ли строка с подстроки
- `endsWith()` : определяет, заканчивается ли строка на определенную подстроку
- `replace()` : заменяет в строке одну подстроку на другую
- `trim()` : удаляет начальные и конечные пробелы
- `substring()` : возвращает подстроку, начиная с определенного индекса до конца или до определенного индекса
- `toLowerCase()` : переводит все символы строки в нижний регистр
- `toUpperCase()` : переводит все символы строки в верхний регистр

5

36. Выберите метод, не преобразующий строку

Методы, преобразующие строку:

`concat()`, `replace()`, `replaceAll()`, `replaceFirst()`, `split()`, `subSequence()`, `substr`

37. Для какого потока не является необходимым вызов метода `close`

Если поток открыт в конструкции `try-with-resources`.

38. К стандартным потокам ввода-вывода Java не относится

`InputStream` – ПОТОК ВВОДА. К нему относятся: `FileInputStream`, `ByteArrayInputStream`, `FilterInputStream` (`DataInputStream`, `BufferedInputStream`), `ObjectInputStream`. `OutputStream` – ПОТОК ВЫВОДА. К нему относятся: `FileOutputStream`, `ByteArrayOutputStream`, `FilterOutputStream` (`DataOutputStream`, `BufferedOutputStream`), `ObjectOutputStream`.

39. Пакет `java.lang` является

Пакет `java.lang` является наиболее важным из всех пакетов, входящих в Java API, поскольку включает классы, составляющие основу для всех других классов. Каждый класс в Java неявным образом импортирует все классы данного пакета, поэтому данный пакет можно не импортировать.

40. Пакет `java.util` является

Java включает большое количество вспомогательных классов, широко используемых в других встроенных пакетах Java. Эти классы расположены в пакете `java.util` и используются для работы с набором объектов, взаимодействия с системными функциями низкого уровня, для работы с математическими функциями, генерации случайных чисел и манипуляций с датой и временем.

41. Строка не может быть преобразована в число типа `double` применением

Может быть преобразована:

- С использованием конструктора

```
(Double d1 = new Double("4.4e10");)
```

- С использованием метода `valueOf` класса `Double`

```
(Double d2 = Double.valueOf(str1);)
```

- С использованием метода `parseDouble` класса `Double`

```
(d3 = Double.parseDouble(str2);)
```

42. Строка не может быть преобразована в число типа `int` применением

Может быть преобразована:

- С использованием конструктора

```
(Integer i1 = new Integer("20349"));
```

- С использованием метода `valueOf` класса `Integer`

```
(Integer i2 = Integer.valueOf(str1));)
```

- С использованием метода `parseInt` класса `Integer`

```
(i3 = Integer.parseInt(str2));)
```

6

43. Глубокое копирование массива объектов может быть достигнуто

Используем Java Object Serialization, чтобы сделать глубокую копию.

44. Глубокое копирование массива объектов — это...

Глубокая копия возникает, когда объект копируется вместе с объектами, к которым он относится.

45. К основному недостатку массива по сравнению с коллекцией относят

Массивы имеют фиксированную длину, индексную адресацию.

46. Коллекция — это...

классы, основная цель которых – хранить набор других элементов.

47. Основные типы коллекций в Java не включают

Основные типы коллекций в Java:

- `Collection` : базовый интерфейс для всех коллекций и других интерфейсов коллекций
- `Queue` : наследует интерфейс `Collection` и представляет функционал для структур данных в виде очереди

- Deque : наследует интерфейс Queue и представляет функционал для двунаправленных очередей
- List : наследует интерфейс Collection и представляет функциональность простых списков
- Set : также расширяет интерфейс Collection и используется для хранения множеств уникальных объектов
- SortedSet : расширяет интерфейс Set для создания сортированных коллекций
- NavigableSet : расширяет интерфейс SortedSet для создания коллекций, в которых можно осуществлять поиск по соответствию
- Map : предназначен для созданий структур данных в виде словаря, где каждый элемент имеет определенный ключ и значение. В отличие от других интерфейсов коллекций не наследуется от интерфейса Collection

7

48. В языке Java



49. Выберите оператор, не являющийся оператором цикла

операторы цикла: while , do...while , for .

50. Выберите оператор, являющийся оператором ветвления

if...else , switch .

51. Выберите оператор, являющийся оператором перехода

break , continue , return .

52. Оператор switch чаще всего используется

Оператор switch — проверяет переменную на равенство в отношении списка значений. Каждое значение называется case , и переменная переключаясь проверяется для каждого case.

53. Скомпилируется ли следующий код:

```
int i; int j; (false ? i: j) = 55;
```

нет

54. Цикл `for` в форме `for each` используется

используется для перебора элементов массива или коллекции

8

55. Выберите ложное высказывание относительно конструкторов в классах Java

Истинное:

- Конструктор — это метод класса, который инициализирует новый объект после его создания.
- Имя конструктора совпадает с именем класса.
- У конструкторов нет типа возвращаемого результата.
- Конструкторы можно перегружать.

56. Выберите ложное высказывание относительно конструкторов в классах Java

[см.выше](#)

57. Для метода с одним параметром, которым является Java-объект, его поля будут передаваться

копия ссылки на область памяти, в которой находится объект

58. Методы с модификатором `final`

Суть модификатора `final` - сделать дальнейшее изменение объекта невозможным. При наследовании данный метод нельзя переопределить

59. Перегрузка методов — это

Перегрузка методов — один из способов поддержки полиморфизма в Java(возможность создавать несколько методов с одинаковым названием, но разными параметрами).

60. Поле с модификатором `final` не может быть проинициализирована

Если вы хотите, чтобы после инициализации никто не мог бы изменить вашу переменную, напишите слово `final`. Теперь, изменить переменную нельзя. Если вы попытаете поменять значение, то получите ошибку. Тем не менее, вы не должны сразу задавать значение переменной. Суть в том, что первое заданное значение меняться не будет.

61. При перегрузке методов

Перегрузка метода заключается в следующем — вы создаете метод с таким же именем, но с другим набором параметров. Здесь необходимо добавить важное замечание — имя параметра НЕ ИМЕЕТ значения. Т.е. если вы сделаете два метода `summa` с двумя параметрами типа `double` и с разными именами, это будет ошибкой.

62. Статические методы и поля класса — это элементы...

Модификатор `static` в Java напрямую связан с классом, если поле статично, значит оно принадлежит классу, если метод статичный, аналогично — он принадлежит классу. Исходя из этого, можно обращаться к статическому методу или полю используя имя класса. Например, поле `count` статично в классе `Counter`, значит, обращаемся: `Counter.count`. Не забываем про правила `private`, `protected`.

9

63. Анонимный класс — это

Анонимный класс (anonymous class) - это локальный класс без имени. На основании анонимного класса создается поток и запускается с помощью метода `start` класса `Thread`. Синтаксис создания анонимного класса базируется на использовании оператора `new` с именем класса (интерфейса) и телом новосозданного анонимного класса. Основное ограничение при использовании анонимных классов - это невозможность описания конструктора, так как класс не имеет имени.

```
new Thread(new Runnable() {  
    public void run() {  
        ...  
    }  
}).start();
```

64. Выберите ложное высказывание



65. Выберите ложное высказывание о вложенных классах



66. Для вызова метода суперкласса в классе-потомке (при наличии в классе-потомке одноименного метода) нужно

Ключевое слово `super` можно использовать для вызова конструктора суперкласса и для обращения к члену суперкласса, скрытому членом подкласса.

67. Для переопределения метода в классе потомке достаточно

Если вы собираетесь переопределить метод, используйте `@Override`, и компилятор выдаст сообщение об ошибке, если вместо переопределения будет случайно выполнена перегрузка

68. Для указания того, что класс является подклассом используется

Использовать ключевое слово `extends`

69. Если в суперклассе есть один конструктор с параметрами, а в классе-потомке определен новый конструктор, не содержащий вызов конструктора суперкласса, то

конструкторы вызываются по порядку выведения классов: от суперкласса к подклассу.

70. Ключевое слово `super` нельзя использовать

Ключевое слово `super` можно использовать для вызова конструктора суперкласса и для обращения к члену суперкласса, скрытому членом подкласса. У суперкласса могут быть несколько перегруженных версий конструкторов, поэтому можно вызывать метод `super()` с разными параметрами. Программа выполнит тот конструктор, который соответствует указанным аргументам.

71. Переопределение методов при наследовании позволяет

изменять функциональность класса, его поведение

10

72. В интерфейсе все поля являются по умолчанию

Все поля, определяемые в интерфейсе являются неявно финальными и статическими и должны быть проинициализированы.

73. В интерфейсе все функции являются по умолчанию

Все методы являются неявно общедоступными.

74. Выберите ложное высказывание об абстрактных классах

Классы с модификатором `abstract`

- Нельзя инстанцировать объект такого класса
- Реализация предоставляется в классах-потомках
- Если классы-потомки не содержат реализации, то они тоже должны быть абстрактными
- Абстрактный класс может содержать полностью реализованные методы

75. Выберите ложное высказывание об интерфейсах

Интерфейсы:

- Определяются с помощью ключевого слова `interface`
- Назначение интерфейсов – предоставлять альтернативу множественному наследованию в том, что касается наследования поведения (т.е. определенных методов)
- Интерфейс может содержать описание методов без их реализации
- Реализация интерфейса производится в классе, в заголовке которого явно указывается необходимость реализации данного интерфейса (ключевое слово `implements`). Если реализация частичная, то такой класс должен быть абстрактным (`abstract`)
- Интерфейсы могут формировать иерархии (расширение интерфейсов)
- Методы в интерфейсе могут быть статическими

76. Динамическая диспетчеризация методов позволяет

Динамическая диспетчеризация методов важна потому, что именно с ее помощью Java реализует полиморфизм времени выполнения.

Динамическая диспетчеризация - это когда ссылочной переменной суперкласса указываем ссылку на подкласс и вызываем переопределенный метод.

77. Для расширения интерфейса используется ключевое слово

`extends`

78. Для создания классов, которые не могут иметь потомков, применяется

`final`

79. Для указания того, что класс реализует интерфейс используется

`implements`

80. Интерфейс — это

Определяется с помощью ключевого слова `interface`

Назначение интерфейсов – предоставлять альтернативу множественному наследованию в том, что касается наследования поведения (т.е. определенных методов)

81. Класс с модификатором `final`

Классы с модификатором `final` :

- Считаются запечатанными, то есть от таких классов запрещено наследование
- Содержат только финальные методы (неявно)

11

82. Выберите ложное высказывание о перечислениях

Истинное:

- Перечисления – списки именованных констант
- Перечисления в Java представляют собой классы (могут содержать в себе конструкторы, методы, поля данных)
- Перечисления не могут быть суперклассами и не могут наследоваться от других классов

83. Выберите ложное высказывание о перечислениях

[click](#)

84. Выберите неверное утверждение о пакетах

Пакет (package) — это некий контейнер, который используется для того, чтобы изолировать имена классов. Например, вы можете создать класс `List`, заключить его в пакет и не думать после этого о возможных конфликтах, которые могли бы возникнуть если бы кто-нибудь еще создал класс с именем `List`.

Пакеты — это механизм, который служит как для работы с пространством имен, так и для ограничения видимости.

Подключается директивой `import`.

85. Ключ `-classpath` при запуске утилиты `java` используется для

Определяет список каталогов, JAR-файлов и ZIP-архивов для поиска файлов классов

86. Ключ `-d` при запуске утилиты `javac` используется для

указания каталога назначения для файлов классов.

87. Пакеты в Java подключаются с помощью директивы

`import`

88. Перечисление имеет predefined метод

- `public static тип_перечисления[] values()`
- `public static тип_перечисления valuesOf(String строка)`

89. Перечисление имеет predefined метод

[см 88](#)

12

90. Выберите ложное утверждение об исключениях

Истинные:

- Обработка исключений – механизм языка программирования Java, применяемый для корректной обработки т.н. исключительных ситуаций (деление на ноль, выход за пределы диапазона индексов массива, открытие несуществующего файла и т.д.)
- Включает блоки: `try` , `catch` , `finally`
- Существует 2 вида исключений : проверяемые/непроверяемые
- Существует 3 класса: `Exception` , `RuntimeException` и `Error`

91. Выберите признак проверяемого исключения

Блок `catch` должен присутствовать в той же функции, где генерируется исключение. Если же данный блок отсутствует, то заголовок функции должен содержать оператор `throws` с указанием типа исключения для обработки этого исключения в вызывающей функции.

92. Выход за пределы индексации массива относится к типу исключения

`IndexOutOfBoundsException` (`RuntimeException`)

93. К классам исключений не относится

Относится:

- `Exception` используется для перехвата пользовательских исключений
- `RuntimeException` используется для стандартных исключений времени выполнения (деление на ноль, выход за диапазон индексации и т.д.)
- `Error` используется для внутрисистемных исключений

94. К основным методам класса `Throwable` не относится

Относится:

- `String getMessage()` – возврат подробного сообщение об исключении
- `Throwable initCause(Throwable имяИсключения) / getCause`
- `void printStackTrace()`

- `void printStackTrace(PrintStream поток)`
- `StackTraceElement [] getStackTrace()`
- `String toString()`

95. Ошибка деления на ноль относится к классу исключения

`ArithmeticException (RuntimeException)`

96. Ошибка доступа к файлу относится к классу исключения

`FileNotFoundException (RuntimeException)`

13

97. Знак «*» в регулярном выражении означает

любое количество экземпляров элемента (в том числе и нулевое)

98. Знак «.» в регулярном выражении означает

представляет собой сокращенную форму записи для символьного класса, совпадающего с любым символом

99. Квантификаторы нужны для

позволяют задавать количество вхождений символа в строку

100. Круглые скобки в регулярном выражении служат для

для выделения групп регулярных выражений

101. Символьный класс определяет

перечень символов которые могут быть (или НЕ могут) на месте данного символа

102. Укажите, какая строки будут соответствовать указанному регулярному выражению

```
[a-zA-Z]{1}[a-zA-Z\d\.\_\-]+\@([a-zA-Z]+\.\.){1,2}((net)|(com)|(org))
```

email

14

103. Восстановление объекта при сериализации производится

- нужно упаковать `InputStream` в `ObjectInputStream` и вызвать метод `readObject()`
- необходимо выполнить обработку исключения `ClassNotFoundException`
- нужно привести полученный объект к правильному типу
- из системы, в которой происходит восстановления объекта, должен быть доступен файл класса.

104. Выберите истинное утверждение о сериализации

- Сохранены и восстановлены абсолютно все поля, даже те у которых указан тип доступа `private` и `protected`.
- Обработаны поля, указанные как `val`.
- Новый объект будет создан, хотя конструктора без параметров у него нет, а существующий не вызывался.
- Состояние объекта сохраняется и восстанавливается, минуя все синтаксические ограничения, которые указаны в тексте программы.
- `Serializable` работает только с полями
- Возможна передача сериализованного объекта по сети, сохранение его на диске и т.д.
- Сериализация уменьшает возможность изменения реализации класса, поэтому нужно подходить к ней с умом
- Класс, для которого возможна сериализация, должен реализовывать интерфейс `java.io.Serializable`
- `ObjectOutputStream` кэширует объекты в потоке. Решение: использование функции `reset`, очищающей поток.
- Контроль версий: сериализация в Java предоставляет удобный механизм для корректной работы с модифицируемыми классами

105. Для указания того, что во время сериализации объекта некоторое поле нужно игнорировать, используется модификатор

`transient`

106. К несериализуемому системному типу относится

Классы `Object` , `Thread` , `OutputStream` , `Socket`

107. К сериализуемым системным типам относится

не `Object` , `Thread` , `OutputStream` , `Socket`

108. Необходимым условием для сериализации объектов класса является

Класс, для которого возможна сериализация, должен реализовывать интерфейс `java.io.Serializable`

109. Сериализация — это

это процесс перевода структуры данных или состояния объекта в формат, который может быть сохранен и восстановлен потом в другой компьютерной среде. После приема серии битов, они пересчитываются в соответствии с форматом сериализации, и могут быть использованы для создания семантически идентичного клона исходного объекта

110. Сохранение объекта при сериализации производится

Сохранение объекта выполняется с помощью класса `java.io.ObjectOutputStream` . Необходимо создать выходной поток `OutputStream` , упаковать его в `ObjectOutputStream` и вызвать метод `writeObject()`

15

111. К методам, общим для всех объектов не относится

У класса есть несколько важных методов.

- `Object clone()` - создаёт новый объект, не отличающийся от клонируемого
- `boolean equals(Object obj)` - определяет, равен ли один объект другому
- `void finalize()` - вызывается перед удалением неиспользуемого объекта
- `Class<?> getClass()` - получает класс объекта во время выполнения
- `int hashCode()` - возвращает хеш-код, связанный с вызывающим объектом
- `void notify()` - возобновляет выполнение потока, который ожидает вызывающего объекта
- `void notifyAll()` - возобновляет выполнение всех потоков, которые ожидают

вызывающего объекта

- `String toString()` - возвращает строку, описывающий объект
- `void wait()` - ожидает другого потока выполнения
- `void wait(long millis)` - ожидает другого потока выполнения
- `void wait(long millis, int nanos)` - ожидает другого потока выполнения

Методы `getClass()`, `notify()`, `notifyAll()`, `wait()` являются финальными и их нельзя переопределять.

Из лекции:

К общим методам относятся:

- `hashCode`
- `equals`
- `toString`
- `clone`
- `finalize`
- `compareTo`

112. Функция `clone()`

Создаёт новый объект, не отличающийся от клонируемого. Находится в классе `Object`. Объявлен как `protected`. Реализует поверхностное клонирование.

113. Функция `compareTo()`

Метод `compareTo` в Java сравнивает вызывающий объект с объектом, переданным в качестве параметра, и возвращает в результате выполнения сравнения целое число:

- положительное, если вызывающий объект больше объекта, переданного в качестве параметра;
- отрицательное, если вызывающий объект меньше объекта, переданного в качестве параметра;
- нуль, если объекты равны. Метод описан в интерфейсе `Comparable`.
Используется при сортировках.

114. Функция `equals()`

Принадлежит классу `Object`. По дефолту сравнивает ссылки `==`. Необходимо переопределять для корректного сравнения. В `String` классе переопределен на корректное сравнение строк.

115. Функция `hashCode()`

Возвращает хеш-код, связанный с вызывающим объектом. Хеш-код - это целое число, генерируемое на основе конкретного объекта. Его можно рассматривать как шифр с уникальным значением. У любого объекта имеется хеш-код, определяемый по умолчанию, который вычисляется по адресу памяти, занимаемой объектом. Значение хеш-кода возвращает целочисленное значение, в том числ и отрицательное. Если в вашем классе переопределяется метод `equals()`, то следует переопределить и метод `hashCode()`.

116. Функция `toString()`

Возвращает строку, описывающий объект.

Очень часто при использовании метода `toString()` для получения описания объекта можно получить набор бессмысленных символов, например, `[I@421199e8`. На самом деле в них есть смысл, доступный специалистом. Он сразу может сказать, что мы имеем дело с одномерным массивом (одна квадратная скобка), который имеет тип `int` (символ `I`).

Обычно принято переопределять метод, чтобы он выводил результат в читаемом виде.

16

117. Выберите истинное высказывание

Выбрал

118. Выберите корректный пример отношения композиции

Разновидность отношения агрегации, при которой составные части целого имеют такое же время жизни, что и само целое.

Это отношение служит для выделения специальной формы отношения «часть-целое». Графически отношение композиции изображается сплошной линией, один из концов которой представляет собой закрашенный внутри ромб. Этот ромб указывает на тот из классов, который представляет собой класс-композицию или «целое».

119. Выберите ложное высказывание о спецификации абстрактного класса в UML

Название абстрактного классификатора показано курсивом, где разрешено используемым шрифтом. В качестве альтернативы или, кроме того, абстрактный классификатор может быть показан с использованием текстовой аннотации {abstract} после или под ее именем.

120. Выберите ложное высказывание о спецификации интерфейса в UML

Интерфейс (interface) служит для спецификации параметров модели, которые видимы извне без указания их внутренней структуры. В языке UML интерфейс является классификатором и характеризует только ограниченную часть поведения моделируемой сущности. Применительно к диаграммам вариантов использования, интерфейсы определяют совокупность операций, которые обеспечивают необходимый набор сервисов или функциональности для актеров. Интерфейсы не могут содержать ни атрибутов, ни состояний, ни направленных ассоциаций. Они содержат только операции без указания особенностей их реализации. Формально интерфейс эквивалентен абстрактному классу без атрибутов и методов с наличием только абстрактных операций.

На диаграмме вариантов использования интерфейс изображается в виде маленького круга, рядом с которым записывается его имя. Кроме этого, интерфейсы могут соединяться с вариантами использования пунктирной линией со стрелкой/сплошной без стрелки.

121. Для обозначения статических атрибутов и операций в UML используется

Нижнее сплошное подчеркивание

122. Для обозначения абстрактного класса его имя и абстрактные функции

Используется курсив

123. Для обозначения отношения обобщения в языке UML используется

Сплошная линия с треугольной стрелкой на одном из концов. Стрелка указывает на более общий класс.

124. Для обозначения реализации интерфейса в языке UML используется

Если класс реализует интерфейс - сплошная линия с кружком(интерфейсом). Если требует интерфейс - то значок "гнездо" и пунктирная линия.

125. Зависимость – это...

Зависимость обозначает такое отношение между классами, что изменение спецификации класса-поставщика может повлиять на работу зависимого класса, но не наоборот.

126. Постусловие операции это

Условие, которое должно быть истинным, когда вызов операции успешно завершился, в предположении, что все предусловия были удовлетворены.

17

127. Антипаттерн – это

Антипаттерн — это распространённый подход к решению класса часто встречающихся проблем, являющийся неэффективным, рискованным или непродуктивным.

128. Выберите метод, являющийся методом рефакторинга

Наиболее употребимые методы рефакторинга:

- Изменение сигнатуры метода (change method signature)
- Инкапсуляция поля (encapsulate field)
- Выделение класса (extract class)
- Выделение интерфейса (extract interface)
- Выделение локальной переменной (extract local variable)
- Выделение метода (extract method)
- Генерализация типа (generalize type)
- Встраивание (inline)
- Введение фабрики (introduce factory)

- Введение параметра (introduce parameter)
- Подъём метода (pull up method)
- Спуск метода (push down method)
- Переименование метода (rename method)
- Перемещение метода (move method)
- Замена условного оператора полиморфизмом (replace conditional with polymorphism)
- Замена наследования делегированием (replace inheritance with delegation)
- Замена кода типа подклассами (replace type code with subclasses)
- Замена кода стек в рекурсию и обратно

129. К принципам Grasp не относится

GRASP выделяет следующие принципы-шаблоны:

- Information Expert (Информационные эксперт)
- Creator (Создатель)
- Controller (Контроллер)
- Low Coupling (Слабая связанность)
- High Cohesion (Высокая сцепленность)
- Pure Fabrication (Чистая выдумка или чистое синтезирование)
- Indirection (Посредник)
- Protected Variations (Соккрытие реализации или защищенные изменения)
- Polymorphism (Полиморфизм)

130. Принцип "Open-Closed" гласит

Программные сущности (классы, модули, функции и т. п.) должны быть открыты для расширения, но закрыты для изменения

131. Принцип "The Dependency Inversion Principle" гласит

Модули верхних уровней не должны зависеть от модулей нижних уровней. Оба типа модулей должны зависеть от абстракций. Абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций.

132. Принцип "The Single Responsibility Principle" гласит

каждый объект должен иметь одну обязанность и эта обязанность должна быть полностью инкапсулирована в класс. Все его сервисы должны быть направлены исключительно на обеспечение этой обязанности.

133. Рефакторинг направлен на

Цель рефакторинга — сделать код программы более легким для понимания; без этого рефакторинг нельзя считать успешным.

18

134. К поведенческим паттернам относится паттерн

- Цепочка обязанностей
- Команда
- Итератор
- Посредник
- Снимок
- Наблюдатель
- Состояние
- Стратегия
- Шаблонный метод
- Посетитель

135. К порождающим паттернам проектирования не относится

Поведенческие

- Цепочка обязанностей
- Команда
- Итератор
- Посредник
- Снимок
- Наблюдатель
- Состояние
- Стратегия
- Шаблонный метод

- Посетитель

Структурные

- Адаптер
- Мост
- Компоновщик
- Декоратор
- Фасад
- Легковес
- Заместитель

136. К структурным паттернам относится паттерн

- Адаптер
- Мост
- Компоновщик
- Декоратор
- Фасад
- Легковес
- Заместитель

137. Паттерн Абстрактная Фабрика

Порождающий паттерн проектирования, который позволяет создавать семейства связанных объектов, не привязываясь к конкретным классам создаваемых объектов.

144. Паттерн Одиночка

Порождающий паттерн проектирования, который гарантирует, что у класса есть только один экземпляр, и предоставляет к нему глобальную точку доступа.

147. Паттерн Строитель

Порождающий паттерн проектирования, который позволяет создавать сложные объекты пошагово. Строитель даёт возможность использовать один и тот же код строительства для получения разных представлений объектов.

148. Паттерн Фабрика

Порождающий паттерн проектирования, который определяет общий интерфейс для создания объектов в суперклассе, позволяя подклассам изменять тип создаваемых объектов.

138. Паттерн Адаптор

Структурный паттерн проектирования, который позволяет объектам с несовместимыми интерфейсами работать вместе.

139. Паттерн Декоратор

Структурный паттерн проектирования, который позволяет динамически добавлять объектам новую функциональность, оборачивая их в полезные «обёртки».

142. Паттерн Компоновщик

Структурный паттерн проектирования, который позволяет сгруппировать множество объектов в древовидную структуру, а затем работать с ней так, как будто это единичный объект.

143. Паттерн Мост

Структурный паттерн проектирования, который разделяет один или несколько классов на две отдельные иерархии — абстракцию и реализацию, позволяя изменять их независимо друг от друга.

140. Паттерн Итератор

Поведенческий паттерн проектирования, который даёт возможность последовательно обходить элементы составных объектов, не раскрывая их внутреннего представления.

141. Паттерн Команда

Поведенческий паттерн проектирования, который превращает запросы в объекты, позволяя передавать их как аргументы при вызове методов, ставить запросы в очередь, логировать их, а также поддерживать отмену операций.

145. Паттерн Состояние

Поведенческий паттерн проектирования, который позволяет объектам менять поведение в зависимости от своего состояния. Извне создаётся впечатление, что изменился класс объекта.

146. Паттерн Стратегия

Поведенческий паттерн проектирования, который определяет семейство схожих алгоритмов и помещает каждый из них в собственный класс, после чего алгоритмы можно взаимозаменять прямо во время исполнения программы.

19

149. Выберите истинное высказывание относительно ограничений на обобщенные типы

- Нельзя создавать экземпляры обобщённых типов с примитивными типами в качестве аргументов типа.
- Нельзя создавать экземпляры параметров типа
- Нельзя объявлять статические поля с типом параметра типа
- Нельзя использовать приведения типа или `instanceof` с параметризованными типами
- Невозможно создавать массивы параметризованных типов
- Нельзя создавать, ловить (`catch`) или бросать (`throw`) объекты параметризованных типов
- Нельзя перегружать метод так, чтобы формальные параметры типа стирались в один и тот же сырой тип

150. Выберите ложное высказывание относительно ограничений на обобщенные типы

Все что не входит в вопрос выше

151. Метасимвольный аргумент

Обозначается знаком `?` и представляет неизвестный тип.

```
boolean sameAvg(Average<?> ob) {  
    return average() == ob.average();  
}
```

Метасимвол не оказывает никакого влияния на тип создаваемых объектов класса `Average` . Это определяется оператором `extends` в объявлении класса `Average` . Метасимвол просто совпадает с любым достоверным объектом класса `Average` .

152. Обобщенные типы задаются с помощью

С помощью буквы `T` в определении класса `class Account<T>` мы указываем, что данный тип `T` будет использоваться этим классом. Параметр `T` в угловых скобках называется универсальным параметром, так как вместо него можно подставить любой тип. При этом пока мы не знаем, какой именно это будет тип: `String`, `int` или какой-то другой. Причем буква `T` выбрана условно, это может и любая другая буква или набор символов.

153. Ограничение на метасимвольный аргумент, заданное с помощью ключевого слова `extends`

В целом верхняя граница для метасимвольного аргумента задается в следующей общей форме:

```
<? extends суперкласс >
```

где после ключевого слова `extends` указывается суперкласс, т.е. имя класса, определяющего верхнюю границу, включая и его самого. Это означает, что в качестве аргумента допускается указывать не только подклассы данного класса, но и сам этот класс. Ограниченные типы оказываются особенно полезными в тех случаях, когда нужно обеспечить совместимость одного параметра типа с другим.

154. Ограничение на метасимвольный аргумент, заданное с помощью ключевого слова `super`

По мере необходимости можно также указать нижнюю границу для метасимвольного аргумента. Для этой цели служит ключевое слово `super`, указываемое в следующей общей форме:

```
<? super подкласс >
```

В данном случае в качестве аргумента допускается использовать только суперклассы, от которых наследует подкласс, исключая его самого. Это означает, что подкласс, определяющий нижнюю границу, не относится к числу классов, передаваемых в качестве аргумента.

155. При использовании generic типов

Существует возможность создавать более статически типизированный код. Соответственно, программы становятся более надежными и проще в отладке.

Свойства Generics

- Строгая типизация.
- Единая реализация.
- Отсутствие информации о типе.

156. При использовании non-generic типов

Теряется преимущество безопасности типов, предоставляемое дженериками. Generics не дадут скомпилировать код с ошибкой, а у простого типа мы сможем заметить ошибку уже во время работы приложения.

20

157. Базовая концепция Java — Collection — является

Базовым интерфейсом для всех коллекций и других интерфейсов коллекций. Определяет основные методы работы с простыми наборами элементов, которые будут общими для всех его реализаций (например `size()`, `isEmpty()`, `add(E e)` и др.).

158. В какую иерархию коллекций входит Map

Представляет отдельную коллекцию.

159. Выберите операцию, которая не является общей для разных типов коллекций

ОБЩИЕ МЕТОДЫ: `add`, `addAll`, `clear`, `contains`, `isEmpty`, `iterator`, `remove`, `removeAll`, `retainAll`, `size`, `toArray`.

160. Итератор — это

Объект, возвращаемый методом `iterator()`. Используется для построения объектов, которые обеспечивают доступ к элементам коллекции. Такой объект позволяет просматривать содержимое коллекции последовательно, элемент за элементом.

161. К коллекции, специально созданной для работы в многопоточном режиме, относится

Очередь(Queue)

162. Коллекция типа `List` — это

Упорядоченная коллекция, в которой допустимы дублирующие значения и она представляет функциональность простых списков. Иногда их называют последовательностями (sequence). Элементы такой коллекции пронумерованы, начиная от нуля, к ним можно обратиться по индексу.

163. Коллекция типа `Map` — это

Коллекция, состоящая из пар "ключ — значение". У каждого ключа только одно значение. В отличие от других интерфейсов коллекций не наследуется от интерфейса `Collection`.

164. Коллекция типа `Queue` — это

Коллекция, предназначенная для хранения элементов в порядке, нужном для их обработки. В дополнение к базовым операциям интерфейса `Collection`, очередь предоставляет дополнительные операции вставки, получения и контроля.

165. Коллекция типа `Set` — это

Неупорядоченная коллекция, не содержащая повторяющихся элементов. Это соответствует математическому понятию множества.

166. Основные типы коллекций находятся в пакете

```
java.util
```

21

167. `List` в Java является

Интерфейсом для операций с коллекцией, которая является списком.

168. Интерфейс `List` не содержит описание следующего метода

СОДЕРЖИТ: `add`, `addAll`, `get`, `indexOf`, `lastIndexOf`, `listiterator()`, `remove`, `set`, `sort`, `subList`.

169. К основным реализациям `List` не относится

ОТНОСИТСЯ: `ArrayList`, `LinkedList`, `Vector`, `Stack`.

170. Контейнер типа `ArrayList` представляет собой

Простой список объектов, т.е. инкапсулирует в себе обычный массив, длина которого автоматически увеличивается при добавлении новых элементов.

171. Контейнер типа `LinkedList` представляет собой

Связанный список, т.е. структура данных, состоящая из узлов, каждый из которых содержит как собственно данные, так и две ссылки на следующий и предыдущий узел списка.

172. Одна из особенностей `ListIterator` по сравнению с обычным `Iterator` в том

- `Iterator` может использоваться для перебора элементов `Set`, `List` и `Map`. В отличие от него, `ListIterator` может быть использован только для перебора элементов коллекции `List`.
- `Iterator` позволяет перебирать элементы только в одном направлении, при помощи метода `next()`. Тогда как `ListIterator` позволяет перебирать список в обоих направлениях, при помощи методов `next()` и `previous()`.
- При помощи `ListIterator` вы можете модифицировать список, добавляя/удаляя элементы с помощью методов `add()` и `remove()`. `Iterator` не поддерживает данного функционала.

22

173. `Map` в Java является

Интерфейс `Map<K, V>` представляет отображение или иначе говоря словарь, где каждый элемент представляет пару "ключ-значение". При этом все ключи уникальные в рамках объекта `Map`. Такие коллекции облегчают поиск элемента, если нам известен ключ - уникальный идентификатор объекта. Не расширяет интерфейс `Collection`.

174. Взаимодействие потоков может осуществляться с помощью функции

- `run()`
- `start()`

- `sleep(int)`
- `wait()`
- `notify()`
- `notifyAll()`
- `interrupt()`
- `join()`

175. Выберите ложное утверждение о многопоточном программировании

Свойство платформы (например, операционной системы, виртуальной машины и т. д.) или приложения, состоящее в том, что порождённый процесс может состоять из нескольких потоков, выполняющихся «параллельно», то есть без предписанного порядка во времени. При выполнении некоторых задач такое разделение может достичь более эффективного использования ресурсов вычислительной машины. Все потоки выполняются в адресном пространстве процесса и имеют общие дескрипторы файлов. Выполняющийся процесс имеет как минимум один (главный) поток.

176. Выберите ложное утверждение о множествах (Set)

Интерфейс `Set` расширяет интерфейс `Collection` и представляет набор уникальных элементов. `Set` не добавляет новых методов, только вносит изменения унаследованные. Не содержит повторяющихся элементов. Более формально, множества не содержат пары элементов `e1` и `e2`, таких как `e1.equals(e2)`, и не более одного `null` элемента.

177. Выберите ложное утверждение об отображениях (Map)

Интерфейс `Map<K, V>` представляет отображение или иначе говоря словарь, где каждый элемент представляет пару "ключ-значение". При этом все ключи уникальные в рамках объекта `Map`. Такие коллекции облегчают поиск элемента, если нам известен ключ - уникальный идентификатор объекта. Не расширяет интерфейс `Collection`.

178. Главная особенность реализаций `SortedSet` в

Интерфейс `SortedSet` предназначен для создания коллекций, который хранят элементы в отсортированном виде (сортировка по возрастанию). `SortedSet` расширяет интерфейс `Set`, поэтому такая коллекция опять же хранит только уникальные значения.

179. Интерфейс `Comparable` предназначен для

Интерфейс накладывает полное упорядочение на объекты каждого класса, который его реализует. Этот порядок называется естественным порядком класса, а метод `compareTo` класса называется его естественным методом сравнения.

180. Интерфейс `Map` не содержит описание следующего метода

Содержит следующие методы

- `clear()`
- `compute()`
- `computeIfAbsent()`
- `computeIfPresent()`
- `containsKey()`
- `containsValue()`
- `entrySet()`
- `equals()`
- `forEach()`
- `get()`
- `getOrDefault()`
- `hashCode()`
- `isEmpty()`
- `keySet()`
- `merge()`
- `put()`
- `putAll()`
- `putIfAbsent()`
- `remove()`
- `replace()`
- `replaceAll()`
- `size()`
- `values()`

181. Итерация по отображению

```
Map<String,String> example = new HashMap<String,String>();

// через for-each
for (Map.Entry<String,String> entry : example.entrySet())

// используя keySet() для итерирования через ключи
for (String key : example.keySet())

// используя values() для итерирования через значения
for (String value : example.values())

// через итераторы
Iterator<Map.Entry<String, String>> itr = example.entrySet().iterator();

while(itr.hasNext()) {
    Map.Entry<String, String> entry = itr.next();
    System.out.println("Key = " + entry.getKey() +
        ", Value = " + entry.getValue());
}

// через forEach(action)
example.forEach((k, v) -> System.out.println("Key = " + k + ", Value = " + v));
```

182. К основным реализациям Set не относится

Относятся

- HashSet
- LinkedHashSet
- TreeSet

183. К отображению, которое сохраняет элементы в отсортированном порядке относится

TreeMap

184. К преимуществам многопоточности не относится

Преимущества

- Улучшенная реакция приложения
- Более эффективное использование мультипроцессирования
- Улучшенная структура программы

- Эффективное использование ресурсов системы

Недостатки

- Тщательная разработка
- Сложная отладка
- Не всегда вычисления разделенные на несколько частей будут работать быстрее чем в одном потоке

185. К состояниям потока не относится

Состояния потока

- выполнение
- готовность
- блокировка

186. Компараторы позволяют

Обеспечить полный контроль над порядком сортировки коллекций/массивов объектов.

187. Почему Map не расширяет интерфейс Collection?

Коллекции представляют собой совокупность некоторых элементов, а map совокупность пар "ключ-значение".

23

188. Test-Driven Development — это

техника разработки программного обеспечения, которая основывается на повторении очень коротких циклов разработки: сначала пишется тест, покрывающий желаемое изменение, затем пишется код, который позволит пройти тест, и под конец проводится рефакторинг кода.

189. Библиотека JUnit применяется для

модульного тестирования программного обеспечения.

190. Выберите истинное утверждение



191. Выберите ложное утверждение



192. Исчерпывающее тестирование (входное или тестирование путей)

Исчерпывающее тестирование это методика, в которой набор тестов включает в себя все комбинации входных данных и предусловий.

193. Не существует стратегии тестирования методом

Стратегии тестирования

- Тестирование методом «черного ящика» (black-box testing, data-driven testing, input/output-driven testing). Исчерпывающее входное тестирование
- Тестирование методом «белого ящика» (white-box testing). Исчерпывающее тестирование путей
- Сочетание методологий – тестирование методом «серого» ящика (grey-box testing).

194. Тестирование – это

процесс или последовательность процессов, позволяющих удостовериться в том, что программный код делает все то, для чего он предназначался, и наоборот, не делает того, для чего не предназначался.

195. Тестирование методом белого ящика предполагает, что

Тестирование «белого ящика» — это стратегия или метод тестирования, базируется только лишь на коде программы.

Методы «белого ящика»

- Покрытие операторов – добиться выполнения каждой инструкции программы
- Покрытие решений – каждая логическая ветвь должна быть выполнена хотя бы один раз
- Покрытие условий – каждое элементарное логическое условие должно быть выполнено хотя бы один раз
- Покрытие решений и условий
- Комбинаторное покрытие условий

196. Тестирование методом черного ящика предполагает, что

Тестирование «черного ящика» — это стратегия или метод тестирования, базируется только лишь на тестировании по функциональной спецификации и требованиям, при этом не смотря во внутреннюю структуру кода и без доступа к базе данных.

Методы «черного ящика»

- Эквивалентное разбиение
- Анализ граничных значений
- Причинно-следственные диаграммы
- Прогнозирование ошибок

24

197. К методам тестирования "белого ящика" не относятся

К методам тестирования белого ящика ОТНОСЯТСЯ:

- Покрытие операторов
- Покрытие решений
- Покрытие условий
- Покрытие решений и условий
- Комбинаторное покрытие условий

198. К методам тестирования "черного ящика" не относится

К методам тестирования черного ящика ОТНОСЯТСЯ:

- Эквивалентное разбиение
- Анализ граничных значений
- Причинно-следственные диаграммы
- Прогнозирование ошибок

199. Тестирование методом анализа граничных значений

Метод черного ящика

Анализ граничных значений проверяет поведение программы на границах. При проверке диапазона значений после выбора набора данных, находящихся в допустимых пределах, следует проверить, как программа ведет себя с граничными значениями допустимых пределов. Анализ граничных значений наиболее часто используется при проверке диапазона чисел.

Для каждого диапазона существуют две границы: нижняя граница (начало диапазона) и верхняя граница (конец диапазона), и границы — начало и конец каждого действительного раздела. Мы должны проектировать тестовые сценарии, которые дают возможность программе функционировать на границах и со значениями внутри и вне границ.

200. Тестирование методом комбинаторного покрытия условий

Метод белого ящика (лучший из методов белого ящика)

Метод комбинаторного покрытия условий требуют написание тестов, которые реализуют все возможные комбинации истинности условий в каждом решении.

Пример: надо проверить решение 1) $x > 0$ or $x > 2$ и решение 2) $y < 5$

Достаточно заполнить тестами всю таблицу:

Тест	$x > y$	$x > 0$	$y < 5$
$x = 1, y = 0$	true	true	true
$x = -3, y = -2$	false	false	true
$x = 1, y = 2$	false	true	true
<тест 3>	true	false	true
...
<тест 7>	false	false	false

Но количество тестов может уменьшаться из-за противоречия и/или повторения некоторых условий.

201. Тестирование методом покрытия операторов

Метод белого ящика (худший из методов белого ящика)

Метод покрытия операторов требует написания такого количества тестов, чтобы при выполнении их всех каждый оператор был выполнен хотя бы один раз.

202. Тестирование методом покрытия решений

Метод белого ящика

Под **решениями** здесь подразумеваются высказывания в операторах условного перехода, выбора, цикла, взятые целиком. Эти высказывания часто составлены из более простых – элементарных высказываний. Так, в примере `if (x>y or x>0)`, `"x>y or x>0"` – решение, а его части `"x>y"` и `"x>0"` – условия.

Метод покрытия решений требует такого количества тестов, чтобы при выполнении их всех по каждой траектории, соединяющей соседние элементы блок-схемы вычисление прошло хотя бы один раз. Это означает, что каждое решение должно принимать как истинные, так и ложные значения.

203. Тестирование методом покрытия условий

Метод белого ящика

Метод покрытия условий состоит в таком подборе тестов, когда каждое условие (элементарное суждения в условных операторах) принимает как истинное так и ложное значение.

204. Тестирование методом эквивалентного разбиения

Метод черного ящика

Идея тестирования по методу разбиения классов эквивалентности состоит в том, чтобы исключить набор входных данных, которые заставляют систему вести себя одинаково и давать одинаковый результат при тестировании программы.

Положения:

а) каждый тест должен включать столько различных входных условий, сколько это возможно, с тем чтобы минимизировать общее число тестов;

б) необходимо пытаться разбить входную область программы на конечное число классов эквивалентности так, чтобы можно было предположить, что каждый тест, являющийся представителем некоторого класса, эквивалентен любому другому тесту этого класса. Другими словами, если один тест класса эквивалентности обнаруживает ошибку, то следует ожидать, что и все другие тесты этого класса эквивалентности будут обнаруживать эту ошибку. И наоборот, если тест не обнаруживает ошибки, то следует ожидать, что ни один тест этого класса эквивалентности не будет обнаруживать ошибки.

Эти два положения составляют основу методологии тестирования, известного как **эквивалентное разбиение**. Второе положение используется для разработки набора "интересных условий", которые должны быть протестированы, а первое - для разработки минимального набора тестов, покрывающих эти условия.

25

205. Инкрементное тестирование – это

При выполнении **инкрементного тестирования** модули НЕ тестируются изолированно друг от друга, а постепенно подключаются к набору уже протестированных модулей и тестируются в составе сборки.

Процедуру инкрементного тестирования можно выполнять в нисходящем или восходящем порядке.

206. К достоинствам восходящего тестирования относится

1. Имеет преимущества, если основные ошибки встречаются главным образом на нижних иерархических уровнях программы
2. Легче создавать тестовые условия
3. Проще обеспечить наблюдение за результатами тестирования

207. К достоинствам нисходящего тестирования относится

1. Имеет преимущества, если основные ошибки встречаются главным образом на верхних иерархических уровнях программы
2. Представление тестов упрощается после подключения функций ввода-вывода
3. Ранее формирование каркаса программы обеспечивает возможность демонстрации ее работы

208. К недостаткам восходящего тестирования относится

1. Необходимо разрабатывать модули-драйверы
2. Программа как единое целое не существует до тех пор, пока не добавлен последний модуль

209. К недостаткам нисходящего тестирования относится

1. Необходимо разрабатывать модули-заглушки
2. Модули-заглушки часто оказываются сложнее, чем предполагалось изначально
3. До подключения функций ввода-вывода представление тестовых данных в заглушках может вызывать затруднения
4. Невозможность или значительная сложность создания тестовых условий
5. Труднее обеспечить наблюдение за результатами тестирования
6. Наталкивает на мысль о возможности совмещения стадий проектирования и тестирования
7. Поощряет отсрочку окончательного тестирования некоторых модулей

210. Модульное тестирование – это

Модульное тестирование – это процесс тестирования отдельных блоков, подпрограмм, классов и процедур, образующих крупную программу.

Цель модульного тестирования – сравнение функций, реализуемых модулем, со спецификациями, описывающими его функциональные или интерфейсные характеристики.

211. Неинкрементное тестирование – это

При традиционном **неинкрементном подходе** тестирование выполняется следующим образом:

- Сначала выполняется модульное тестирование всех модулей, причем каждый тестируется как независимая сущность. Модули могут тестироваться параллельно.
- Модули объединяются или интегрируются в программу

26

212. Категория "Возможности" в системном тесте предполагает

Проверяется полнота реализации функциональных возможностей, определенных целями

213. Категория "Восстанавливаемости" в системном тесте предполагает

Тестируется способность средств восстановления выполнять свои функции

214. Категория "Надежности" в системном тесте предполагает

Определяется соответствие программы специфицированным показателям надежности, таким как длительность непрерывной работы и среднее время наработки на отказ

215. Цель интеграционного теста

Интеграционное тестирование — это такой процесс, который подразумевает тестирование модулей программы, объединенных в группу. Проходит после модульного.

216. Цель модульного теста

Сравнение функций, реализуемых модулем, со спецификациями, описывающими его функциональные или интерфейсные характеристики.

217. Цель приемочного теста

Формальный процесс тестирования, который проверяет соответствие системы требованиям и проводится с целью:

- определения удовлетворяет ли система приемочным критериям;
- вынесения решения заказчиком или другим уполномоченным лицом принимается приложение или нет

218. Цель системного теста

Системное тестирование – это процесс, направленный на проверку функциональных и нефункциональных требований системы в целом, не принимая во внимания ее отдельные компоненты.

219. Цель тестирования установки

Тестирование установки направленно на проверку успешной инсталляции и настройки, а также обновления или удаления программного обеспечения.

220. Цель функционального теста

Процесс, нацеленный на выявление расхождений между поведением программы и внешней спецификацией.

27

221. Аппендер - это объект, который определяет

что нужно делать с логируемыыми сообщениями.

222. Библиотека log4j используется для

логгирования сообщений

223. Инварианты класса – это

инвариант, используемый для ограничения объектов класса.

224. Методика проектирования по контракту

предполагает, что проектировщик должен определить формальные, точные и верифицируемые спецификации интерфейсов для компонентов системы. При этом, кроме обычного определения абстрактных типов данных, также используются предусловия, постусловия и инварианты.

225. Механизм утверждений (assertions) используется в Java для проверки

инвариантов

226. Наивысший приоритет в log4j имеют сообщения с уровнем

FATAL

227. Наивысший приоритет среди представленных в log4j имеют сообщения с уровнем

FATAL

228. Наинизший приоритет в log4j имеют сообщения с уровнем

TRACE

229. Наинизший приоритет среди представленных в log4j имеют сообщения с

уровнем

TRACE

230. Постусловия – это

условие, истинность которого проверяется после выполнения тела цикла или процедуры.

231. Предусловия - это

условие, истинность которого проверяется в начале выполнения тела цикла или процедуры.

232. Условная компиляция в Java позволяет

- можно создавать программы различных конфигураций.
- Приводит к эффективному использованию памяти, так как ненужный код не хранится в памяти во время выполнения.
- Решение о включении той или иной части программы принимается на этапе компиляции, а не во время выполнения. Это повышает эффективность программы

233. Форматер нужен для определения

Layout (форматер) - объекты, расширяющие абстрактный класс `org.apache.log4j.Layout`, предназначенные для форматирования вывода. Форматирование обычно задаётся в конфигурационном файле.

28

234. Объект, представляющий собой окно программы и содержащий все объекты JavaFX-приложения называется

Stage

235. Объект, представляющий физический контент JavaFX-приложения называется

Scene

236. Основным паттерном, используемым при разработке JavaFX-приложения, является

MVC

237. Разметка JavaFX-приложения хранится в файле с расширением

fxml

238. Макет JavaFX, размещающий все компоненты приложения последовательно друг на друге, называется

StackPane

239. Макет JavaFX, добавляющий компоненты приложения в форме плиток одинакового размера, называется

GridPane

240. Одновременные манипуляции, выполняемые несколькими потоками, над графом сцены JavaFX

недопустимы

241. Классы, применяемые для организации многопоточности в JavaFX, называются

Task и Worker

29

242. Этот интерфейс применяется при использовании статических SQL-запросов, не изменяющихся в процессе работы

Statement

243. Этот интерфейс применяется, если SQL-запросы используют параметры, которые многократно изменяются в процессе работы

PreparedStatement

244. Метод Statement, возвращающий ResultSet -объект, называется

```
ResultSet executeQuery (String SQL)
```

245. Интерфейс `CallableStatement` применяется для

организации доступа к хранимым процедурам БД

246. С помощью этого интерфейса можно получить доступ к данным, полученным оператором `SELECT`

`ResultSet`

247. По выборке, получаемой с помощью объекта `ResultSet`, можно двигаться

Режимы:

- `ResultSet.TYPE_FORWARD_ONLY` – курсор может двигаться только в прямом направлении
- `ResultSet.TYPE_SCROLL_INSENSITIVE` – курсор может перемещаться вперед и назад, а выборка не чувствительна к изменениям, производимыми другими пользователями БД после формирования выборки.
- `ResultSet.TYPE_SCROLL_SENSITIVE` – курсор может перемещаться вперед и назад, а выборка чувствительна к изменениям, производимыми другими пользователями БД после формирования выборки.

248. Выборка, получаемая с помощью `ResultSet` является

по умолчанию `read_only`

249. Ключевая особенность транзакций -

Транзакция либо выполняется полностью, либо не выполняется вообще.

250. Типы данных JDBC

SQL	JDBC/Java
VARCHAR	<code>java.lang.String</code>
CHAR	<code>java.lang.String</code>
BIT	<code>boolean</code>
NUMERIC	<code>java.math.BigDecimal</code>

SQL	JDBC/Java
INTEGER	int
BIGINT	long
TINYINT	byte
SMALLINT	short
DOUBLE	double
TIME	java.sql.Time
DATE	java.sql.Date
BLOB	java.sql.Blob
ARRAY	java.sql.Array
BINARY	byte[]

251. Для закрепления изменений, произведенных транзакцией

```
conn.commit();
```

252. Для включения режима транзакций при наличии объекта Connection conn используется

```
conn.setAutoCommit(false)
```

253. Для внесения изменений в таблицу базы данных

Для обновления соответствующей таблицы нужно вызвать один из следующих методов:

- public void updateRow()
- public void deleteRow()
- public void refreshRow()
- public void cancelRowUpdates()
- public void insertRow() – может применяться только тогда, когда курсор находится в строке вставки.

30

254. При использовании ссылок на методы можно ссылаться на

- Статический метод
- Параметризованный метод
- Метод экземпляра
- Конструктор

255. Функция является чистой, если

- Выполнение функции не имеет побочных эффектов
- Возвращаемое функцией значение зависит только от входных параметров

256. Функция имеет высший порядок, если

- Функция принимает один или более функций в качестве параметров
- Функция возвращает другую функцию в качестве результата

257. Функция не изменяет состояние, если

Идея неизменяемости заключается в том, что созданное значение никогда не может быть изменено.

258. Функциональный интерфейс — это

интерфейс, который содержит лишь один абстрактный метод.

259. Для сопоставления лямбда-выражения и интерфейса необходимо

Правила сопоставления:

- Интерфейс содержит только один абстрактный (нереализованный) метод
- Параметры лямбда-выражения совпадают с параметрами единственного метода
- Тип возвращаемого значения лямбда-выражения совпадает с типом возвращаемого значения единственного метода функционального интерфейса

260. К встроенным функциональным интерфейсам Java не относится

Относится:

- Function
- Predicate
- UnaryOperator
- BinaryOperator
- Supplier
- Consumer

261. Функциональный интерфейс, представляющий функцию, принимающий один параметр и возвращающий единственное значение, это (выбрать наиболее подходящий)

Function

262. Функциональный интерфейс, представляющий функцию, принимающий один параметр и возвращающий булево значение, это (выбрать наиболее подходящий)

Predicate

263. Функциональный интерфейс, представляющий функцию, принимающий один параметр и возвращающий значение того же типа, это (выбрать наиболее подходящий)

UnaryOperator

264. Consumer — это функциональный интерфейс, представляющий функцию, которая

принимает один параметр без возврата значения. Значение этого параметра может быть выведено на консоль, записано в файл, передано по сети и т.д.

265. Техника, которая комбинирует множественные функции в единую функцию, которая использует все комбинируемые функции, это

Функциональная композиция

266. Функциональная композиция предикатов может выполняться

функциями `and()` и `or()`

267. Функциональная композиция функций может выполняться

функциями `compose()` и `andThen()`

268. Что такое Java Stream API

JSA предоставляет функциональный подход к обработке коллекции объектов. Это компонент, который способен к внутренней итерации своих элементов.

269. Нетерминальная потоковая операция это

возвращает трансформированный поток

270. Терминальная потоковая операция это

возвращает конкретный результат

271. К терминальным потоковым операциям не относится метод

272. К терминальным потоковым операциям относится метод

Относится:

- `findFirst`
- `findAny`
- `collect`
- `count`
- `anyMatch`
- `noneMatch`
- `allMatch`
- `min`
- `max`
- `forEach`
- `forEachOrdered`
- `toArray`
- `reduce`

273. К нетерминальным потоковым операциям не относится метод

274. К нетерминальным потоковым операциям относится метод

Относится:

- filter
- skip
- distinct
- map
- peek
- limit
- sorted
- mapToInt , mapToDouble , mapToLong
- flatMap , flatMapToInt , flatMapToDouble , flatMapToLong

275. К недостаткам Java Stream API относится

todo