1.Перечислите состав класса.

Поля, методы, конструктор(ы), геттеры-сеттеры…

2. Где и как могут использоваться [**static**] [**abstract**] [**final**] в контексте

класса?

Static – статическое поле/метод/класс

Abstract – абстрактный класс, от которого наследуются другие классы

Final – константа

3. Где могут использоваться слова super и this?

Super – basic класс (базовый), this – указатель на вызвавший объект

4. Для чего используется модификатор native?

Это модификатор, который пишется перед объявлением метода. Он означает, что код метода написан не на Java, а на С++ и встроен в Java-машину (ну или подключаемую DLL-библиотеку). Сама Java-машина, как вы уже, наверное, догадываетесь, тоже написана на C++. Как и многие методы стандартных библиотек.

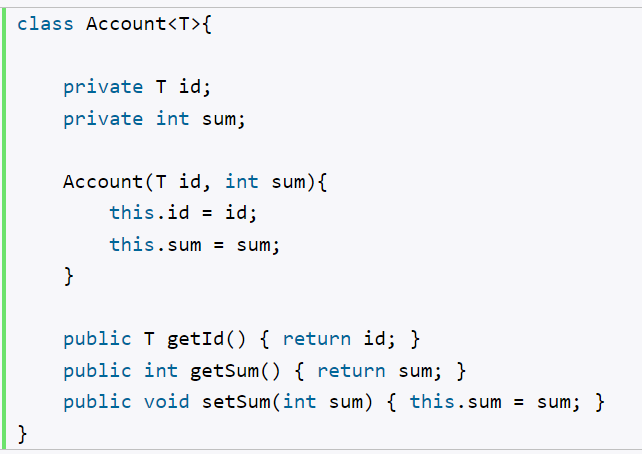
5. Что такое логический и статический блок?

Логический блок (англ. instance initializer block) - это блок кода, который находится внутри класса и выполняется при создании объекта этого класса. Логические блоки используются для инициализации полей объекта, которые не могут быть проинициализированы при объявлении.

Статический блок (англ. static initializer block) - это блок кода, который находится внутри класса и выполняется при первой загрузке класса. Статические блоки используются для инициализации статических полей класса, которые не могут быть проинициализированы при объявлении.

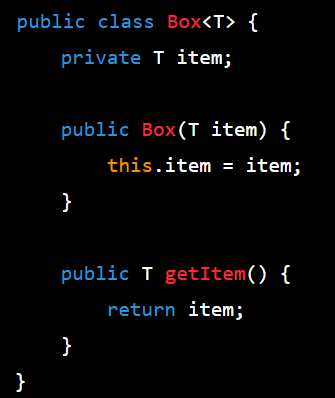
6. Определите параметризованный класс.

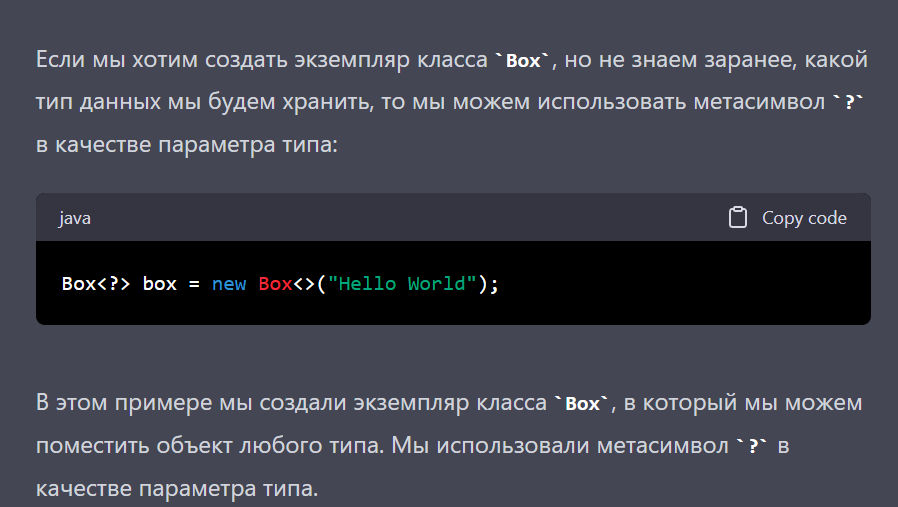
Параметризованный класс (англ. parameterized class) - это класс, который имеет параметры типа (generics). Параметризованный класс позволяет создавать обобщенный тип данных, который может быть использован для хранения различных типов объектов.



7. Как используется метасимвол «?»

Метасимвол "?" (англ. wildcard) используется для обозначения неизвестного типа в параметризованном классе (generics). Метасимвол "?" может быть использован в качестве типа параметра, ограничивая обобщенный тип данных.





8. Какие существуют generic-ограничения?

Существуют следующие generic-ограничения:

* Ограничение сверху (upper bounded wildcard): <? extends T> - ограничивает обобщенный тип данных сверху, т.е. тип данных может быть T или любым подтипом T.
* Ограничение снизу (lower bounded wildcard): <? super T> - ограничивает обобщенный тип данных снизу, т.е. тип данных может быть T или любым супертипом T.
* Ограничение типом (type parameter bound): <T extends SomeClass> - ограничивает тип параметра T конкретным классом SomeClass.

9. Что могут содержать перечисления? Приведите пример

Перечисления (англ. enum) могут содержать константы (перечислимые значения) и методы. Константы в перечислении обычно используются для представления ограниченного набора значений. Методы в перечислении могут использоваться для реализации специфических поведений для каждой константы.

10. Какие существуют ограничения для перечислений?

Ограничения для перечислений в Java:

* Перечисления могут реализовывать интерфейсы и наследоваться от классов;
* Перечисления не могут быть унаследованы от других классов, кроме класса **Enum**;
* Конструкторы перечислений должны быть объявлены как **private** или **default**;
* Перечисления могут быть использованы в switch-конструкциях.

11. Что такое методы подставки?

Методы подстановки (substitution methods) - это методы, которые позволяют использовать подклассы вместо суперклассов в методах и конструкторах. Они используются в полиморфизме для обеспечения более гибкого и удобного использования объектов разных типов.

12. Состав класса Object.

Класс **Object** - это базовый класс для всех объектов в Java. Он содержит набор методов, которые могут быть переопределены в подклассах:

* **equals()** - проверяет, равен ли текущий объект другому объекту;
* **hashCode()** - возвращает уникальный числовой идентификатор объекта;
* **toString()** - возвращает строковое представление объекта;
* **getClass()** - возвращает класс объекта;
* **wait()** - приостанавливает выполнение потока;
* **notify()** - возобновляет выполнение потока, ожидающего на объекте;
* **notifyAll()** - возобновляет выполнение всех потоков, ожидающих на объекте;
* **finalize()** - вызывается перед уничтожением объекта сборщиком мусора.

13. Перечислите соглашения по equlas.

Соглашения по методу **equals()**:

* Рефлексивность - объект должен быть равен самому себе;
* Симметричность - если объект **A** равен объекту **B**, то объект **B** также должен быть равен объекту **A**;
* Транзитивность - если объект **A** равен объекту **B**, а объект **B** равен объекту **C**, то объект **A** также должен быть равен объекту **C**;
* Согласованность - если объекты не изменились, то результат вызова **equals()** не должен меняться;
* Неравенство с **null** - результат сравнения объекта с **null** должен быть **false**.

14. Перечислите соглашения по hashCode() .

* Если метод equals() возвращает true для двух объектов, то их хэш-коды должны быть равны.
* Если метод equals() возвращает false для двух объектов, то их хэш-коды могут быть равны или не равны. Равенство хэш-кодов для неравных объектов улучшает производительность при использовании хэш-таблиц.
* Если поле объекта не используется при сравнении с помощью метода equals(), то его значение не должно учитываться при вычислении хэш-кода.
* Результатом метода hashCode() должно быть целое число.

15. Перечислите соглашения по toString().

Ответ: При переопределении метода toString() необходимо соблюдать следующие соглашения:

* Метод toString() должен возвращать строку, которая представляет объект в человеко-читаемом формате.
* Результат метода toString() не должен содержать конфиденциальную информацию, так как он может быть использован для отладки или вывода информации пользователю.
* Если метод equals() возвращает true для двух объектов, то их метод toString() должен возвращать одинаковые строки.

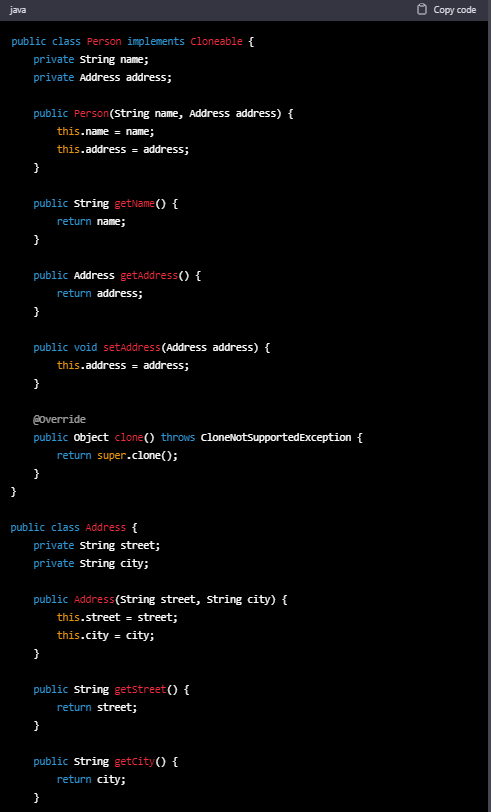
16. Поясните разницу между «неглубким» и «глубоким»

клонированием? Приведите пример.

При клонировании объекта в Java существует два подхода: неглубокое (shallow) и глубокое (deep) клонирование.

При неглубоком клонировании создается новый объект, но его поля ссылаются на те же самые объекты, что и поля клонируемого объекта. То есть, если изменить значение поля в клонированном объекте, то это изменение будет отражено и в оригинальном объекте. Пример:

При глубоком клонировании создается новый объект, и его поля также клонируются, т.е. создаются новые объекты, которые являются копиями объектов, на которые ссылается клонируемый объект. Таким образом, новый объект имеет полностью новый набор объектов, и он не зависит от объекта-источника. Пример:





17. Как можно использовать метод void finalize()?

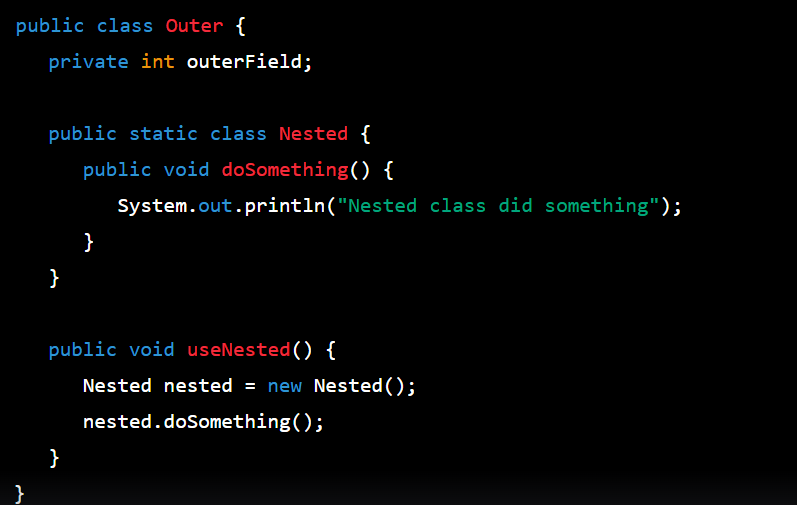
Метод finalize() является методом объекта в Java, который вызывается перед тем, как объект будет удален сборщиком мусора. Этот метод может быть переопределен в пользовательском классе, чтобы выполнить определенные действия перед удалением объекта. Однако, использование этого метода не рекомендуется, так как время его вызова не гарантировано, и он может привести к задержкам в работе программы и проблемам с производительностью. Вместо этого, для выполнения действий перед удалением объекта лучше использовать блок finally или метод close().

18. Что такое внутренние классы (inner)? Привила использования.

Внутренний класс (inner class) - это класс, определенный внутри другого класса. Он может быть объявлен как статическим или нестатическим. Нестатический внутренний класс имеет доступ к полям и методам внешнего класса, а также может использовать ключевое слово this для обращения к своим собственным полям и методам. Статический внутренний класс может быть использован без создания экземпляра внешнего класса.

19. Что такое вложенные (nested) классы? Привила использования.

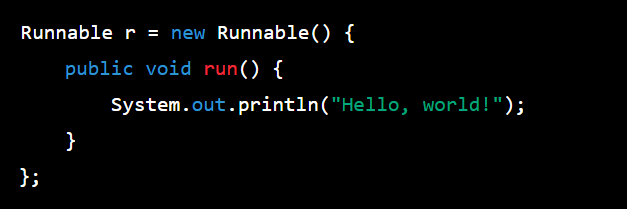
Вложенный класс (nested class) - это класс, объявленный внутри другого класса или интерфейса. Вложенные классы могут быть статическими или нестатическими и могут использоваться для логической группировки классов и уменьшения видимости классов. Вложенный класс также может быть объявлен как локальный класс, определенный внутри метода, или как анонимный класс.



20. Что такое анонимные (anonymous) классы?

Анонимные классы в Java - это специальный тип классов, который позволяет создавать объекты классов "на лету" без явного определения класса в отдельном файле. Они обычно используются для создания объектов интерфейсов, таких как слушатели событий, которые требуют только один метод для реализации.

Пример создания анонимного класса, реализующего интерфейс Runnable:



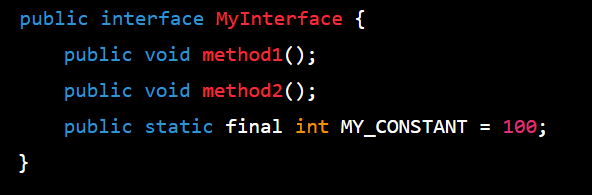
В этом примере мы создали объект **Runnable**, который содержит только один метод **run()**, и определили его анонимно внутри блока кода. Мы можем вызвать метод **run()** на объекте **r**, чтобы вывести строку "Hello, world!" в консоль.

Вот и тут замашки js потихоньку появляются…

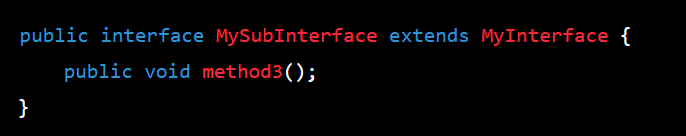
21. Правила определения и наследования интерфейсов.

В Java интерфейс определяет список методов, которые должны быть реализованы классом, который реализует этот интерфейс. Он определяет только абстрактные методы и константы, но не содержит реализации. Интерфейс также может наследовать другой интерфейс.

Определение интерфейса выглядит следующим образом:



Интерфейсы наследуются с помощью ключевого слова **extends**. Например, чтобы создать интерфейс, который наследует другой интерфейс, вы можете использовать следующий код:



В этом примере интерфейс **MySubInterface** наследует интерфейс **MyInterface** и определяет новый метод **method3()**. Класс, который реализует **MySubInterface**, должен реализовать все методы из обоих интерфейсов.

* Интерфейс это соглашение о том, какие методы будут реализованы классами, которые реализуют данный интерфейс.
* Определение интерфейса происходит с помощью ключевого слова "interface".
* В интерфейсе можно определить только методы без реализации (абстрактные методы) и константы.
* Класс может реализовать несколько интерфейсов через ключевое слово "implements".
* Класс, который реализует интерфейс, должен реализовать все методы, объявленные в интерфейсе.
* Интерфейс может наследовать другой интерфейс с помощью ключевого слова "extends".
* При наследовании интерфейса все его методы и константы также наследуются.

22. Приведите иерархию исключений и ошибок? Поясните проверяемые

и непроверяемые исключения.

* В Java исключения и ошибки являются подклассами класса Throwable.
* Все исключения делятся на две категории: проверяемые (checked) и непроверяемые (unchecked).
* Проверяемые исключения – это исключения, которые должны быть обработаны в программе. Они являются подклассом класса Exception, за исключением RuntimeException и его подклассов. Примеры проверяемых исключений: IOException, SQLException, ClassNotFoundException.
* Непроверяемые исключения – это исключения, которые не обязательно должны быть обработаны в программе. Они являются подклассом класса RuntimeException. Примеры непроверяемых исключений: NullPointerException, ArrayIndexOutOfBoundsException, ArithmeticException.
* Класс Error – это подкласс Throwable, который представляет собой серьезные ошибки виртуальной машины Java или окружения, которые не могут быть обработаны программно. Примеры: OutOfMemoryError, StackOverflowError.

1. Object
2. Throwable
   1. Error
      1. AssertionError
      2. OutOfMemoryError
      3. StackOverflowError
      4. и др.
   2. Exception
      1. RuntimeException
         1. NullPointerException
         2. ClassCastException
         3. IndexOutOfBoundsException
         4. и др.
      2. IOException
         1. FileNotFoundException
         2. EOFException
         3. и др.
      3. SQLException
      4. и др.