1.Перечислите состав класса.

Поля конструкторы методы свойства

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

**static** модификатор перед методом или полем говорит о том, что они не принадлежат к экземпляру данного класса. Что это означает для нас? Когда мы описали поле класса или метод как static, его можно вызвать без использования экземпляра класса. То есть вместо такой конструкции: Cat cat = new Cat(); cat.method(), можно написать просто Cat.method(). При условии, что метод объявлен как static. Статические переменные едины для всех объектов класса. У них одна ссылка.

*Абстрактным class* является class, который объявляется abstract— это может или, возможно, не включает абстрактные методы. Абстрактные классы нельзя инстанцировать, но они могут быть разделены на подклассы.

*Абстрактный метод* является методом, который объявляется без реализации (без фигурных скобок, и сопровождается точкой с запятой), как это:

abstract void moveTo(double deltaX, double deltaY);

Если class включает абстрактные методы, сам class *должен* быть объявлен abstract, как в:

public abstract class GraphicObject {

// declare fields

// declare non-abstract methods

abstract void draw();

}

Применяя **final** модификатор Вы говорите, что поля не могут быть изменены, методы переопределены, а классы нельзя наследовать (о наследовании будет отдельная статья). Этот модификатор применяется только к классам, методам и переменным (также и к локальным переменным).

3. Где могут использоваться слова super и this?

**this** представляет текущий экземпляр класса, в то время как **super** - текущий экземпляр родительского класса. Вот один из примеров использования переменных **this** и **super** — вы наверняка уже видели примеры вызовов конструкторов одного из другого, т.н. вызовы конструкторов по цепочке, это возможно благодаря использованию ключевых слов **this** и **super**. Внутри класса для вызова своего конструктора без аргументов используется **this()**, тогда как **super()**используется для вызова конструктора без аргументов, или как его ещё называют, конструктора по умолчанию родительского класса. Между прочим, таким способом вызывать можно не только конструктор без аргументов, а и вообще любой другой конструктор, передав ему соответствующие параметры

4. Для чего используется модификатор native?

Java Native Interface (JNI) — стандартный механизм для запуска кода, под управлением виртуальной машины Java (JVM), который написан на языках С/С++ или Ассемблера, и скомпонован в виде динамических библиотек, позволяет не использовать статическое связывание. Это даёт возможность вызова функции С/С++ из программы на Java, и наоборот.

Модификатор native сигнализирует о том, что метод реализован в платформо-зависимом коде, часто на языке С.

Этот модификатор может быть применен только к методам, но не классам и переменным.

Тело нативного метода должно заканчиваться на (;) как в абстрактных методах, идентифицируя то, что реализация опущена.

*Пример:*

**class** SomethingProcessor {   
   
**public** **native** **void** doSomethingNow();   
   
**static** {   
**try** {   
System.loadLibrary("external\_library");   
} **catch** (UnsatisfiedLinkError e) {   
// catch it!   
}   
}

5. Что такое логический и статический блок?

* блоки инициализации используются для инициализации переменных внутри класса;
* есть два типа блоков инициализации - статический и нестатический;
* синтаксис очень простой - просто пишем выражение внутри блока из двух скобок
* **статический блок** используется для инициализации **статических переменных**, а "обычный"  - **для всех остальных**.

6. Определите параметризованный класс.

Параметризация позволяет создавать классы, интерфейсы и методы, в которых тип обрабатываемых данных задается как параметр

*/\*пример # 9 : объявление класса с двумя параметрами : Subject.java \*/*

**package** chapt03;

**public** **class** Subject <T1, T2> {

**private** T1 name;

**private** T2 id;

**public** Subject() {

}

**public** Subject(T2 ids, T1 names) {

id = ids;

name = names;

}

}

7. Как используется метасимвол «?»

8. Какие существуют generic-ограничения?

Для установки ограничения после универсального параметра ставится слово **extends**, после которого указывается базовый класс ограничения:

|  |  |
| --- | --- |
| 1  2 | class Account{ }  class Transaction<T extends Account>{ } |

К примеру, в данном случае для параметра T в Transaction ограничением является класс Account. То есть на место параметра T мы можем передать либо класс Account, либо один из его классов-наследников. То же самое с интерфейсами

9. Что могут содержать перечисления? Приведите пример

может содержать поля, конструкторы и методы, реализовывать интерфейсы

Тип **enum** — это тип, поля которого состоят из набора некоторых констант.

10. Какие существуют ограничения для перечислений?

конструкторы вызываются автоматически при инициализации

- > не может public и protected

►не может быть суперклассом

►не может быть подклассами

►не может быть абстрактными

►не может создавать экземпляры, используя ключевое слово new

11. Что такое методы подставки?

12. Состав класса Object.

13. Перечислите соглашения по equlas.

14. Перечислите соглашения по hashCode() .

15. Перечислите соглашения по toString().

16. Поясните разницу между «неглубким» и «глубоким» клонированием? Приведите пример.

17. Как можно использовать метод void finalize()?

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

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

20. Что такое анонимные (anonymous) классы?

21. Правила определения и наследования интерфейсов.

22. Приведите иерархию исключений и ошибок? Поясните проверяемые и непроверяемые исключения