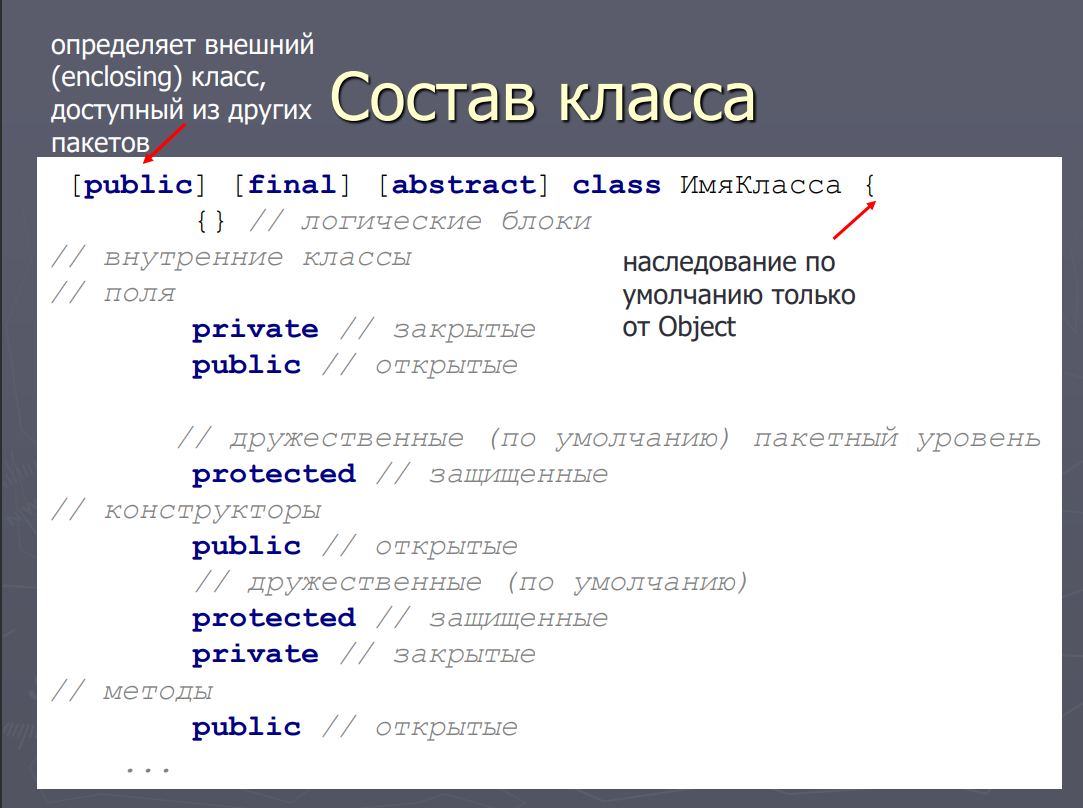
**Ответы к лабораторной работе №3:**

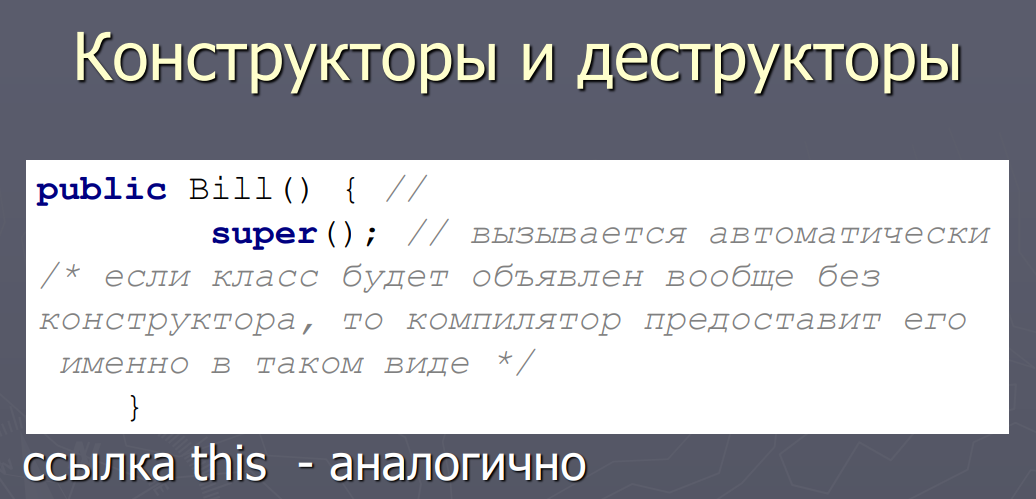
1. Перечислите состав класса.

**Класс** — это, по сути, шаблон для объекта. Он определяет, как объект будет выглядеть и какими функциями обладать. Каждый объект является объектом какого-то класса. Они могут содержать внутренние классы, поля, конструкторы, методы и другие логические блоки.

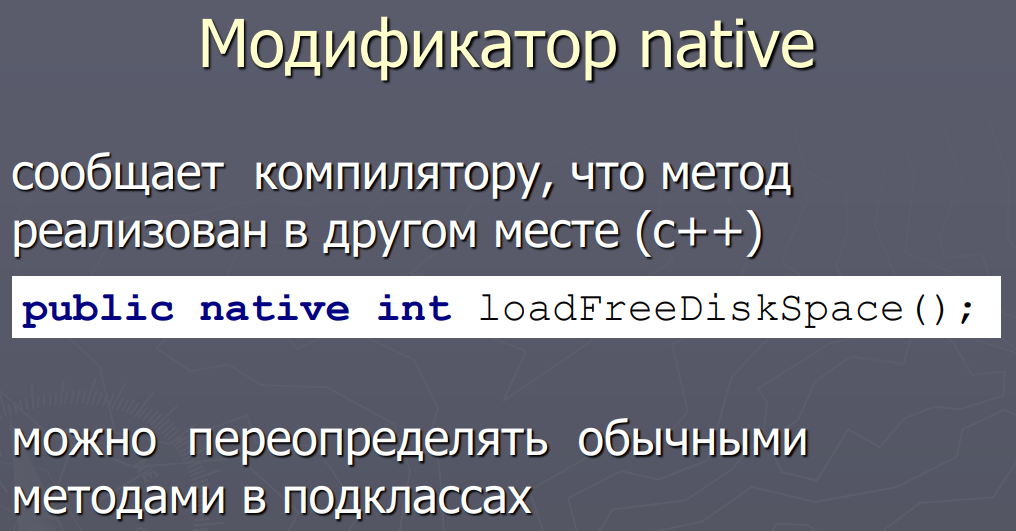
1. Где и как могут использоваться [static] [abstract] [final] в контексте класса?



1. Где могут использоваться слова super и this?

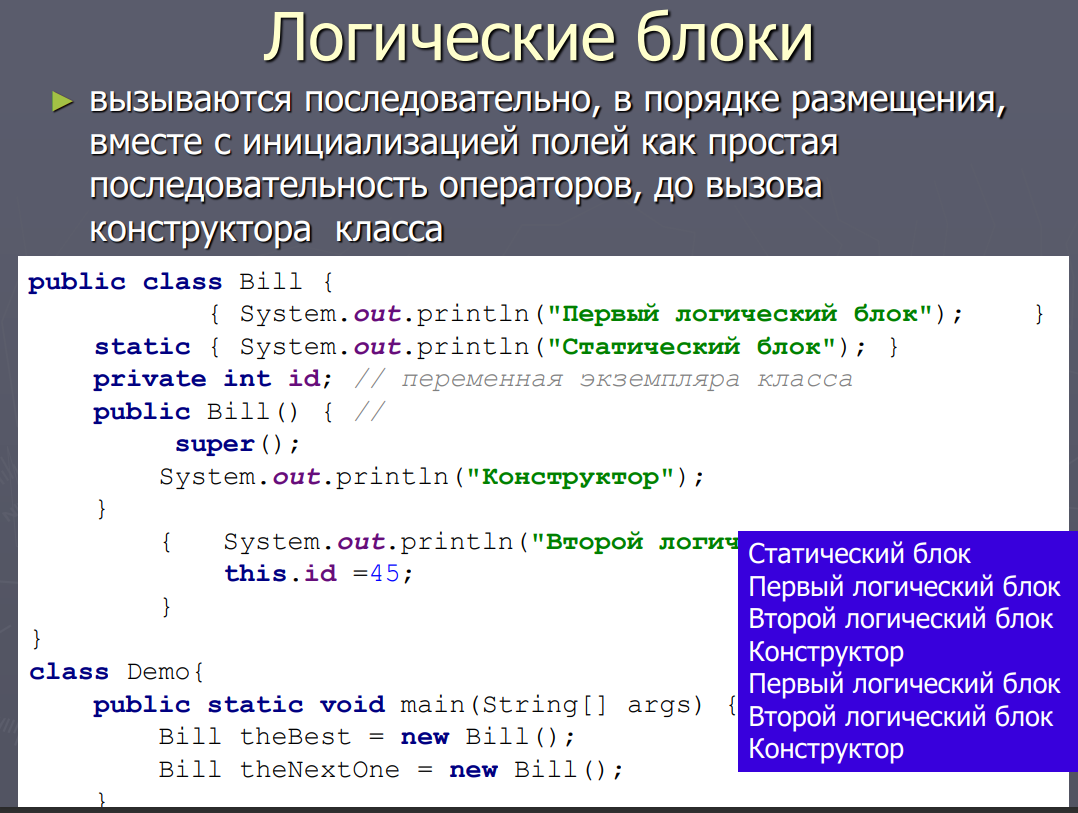


1. Для чего используется модификатор native?



1. Что такое логический и статический блок?

**Cтатический блок** используется для инициализации **статических переменных**, а логический - **для всех остальных**.



1. Определите параметризованный класс.

**Параметризированный класс** – некоторый шаблон, на основе которого можно строить другие классы. Этот класс можно рассматривать как некоторое описание множества классов, отличающихся только типами их данных.

Следующий пример демонстрирует использование параметризованного класса, который описывает матрицу:

public class Matrix<T> {

private T[] array;

public Matrix(T[] array) {

this.array = array.clone();

}

public static void main(String[] args) {

Matrix<Double> doubleMatrix = new Matrix<>(new Double[2]);

Matrix<Integer> integerMatrix = new Matrix<>(new Integer[4]);

Matrix<Byte> byteMatrix = new Matrix<>(new Byte[7]);

}

}

В объявлении *Matrix<Integer>* является аргументом типа.

1. Как используется метасимвол «?»

? — делает символ необязательным, означает 0 или 1. То же самое, что и {0,1}.

1. Какие существуют generic-ограничения?

В качестве ограничений для обобщений могут использоваться только следующие конструкции:

* where T : struct
* where T : class
* where T : new()
* where T : <base class name>
* where T : <interface name>
* where T : U, где U -- другой параметр-тип

В качестве обходных путей можно использовать следующие варианты:

* создать перегрузку для каждого типа.

- оставить параметр T без ограничений и проверять тип в начале метода.

1. Что могут содержать перечисления? Приведите пример

1) *java.lang.Enum* - класс (**new** не требует)

2) при инициализации хотя бы одного - инициализация всех

3) может содержать поля, конструкторы и методы, реализовывать интерфейсы.

1. 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.

1. Что такое методы подставки?

При переопределении методов можно указывать другой тип возвращаемого значения – но только тип, находящийся ниже в иерархии наследования, чем исходный тип.

1. Состав класса Object.

protected Object clone() ;

public boolean equals(Object ob)

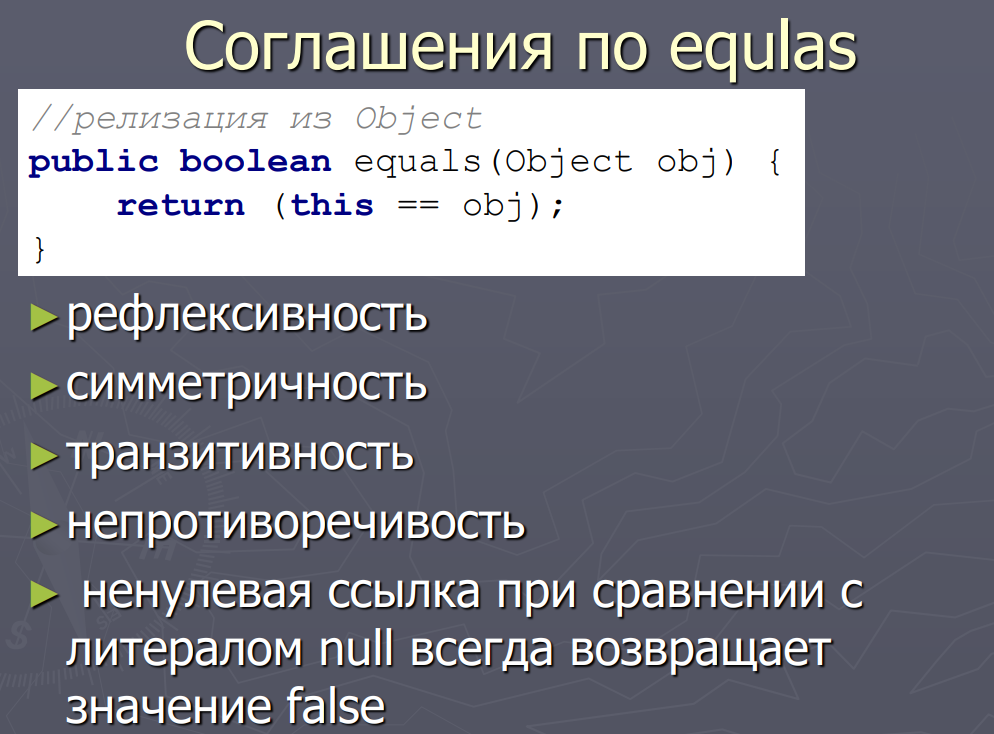
public Class<? extends Object> getClass()

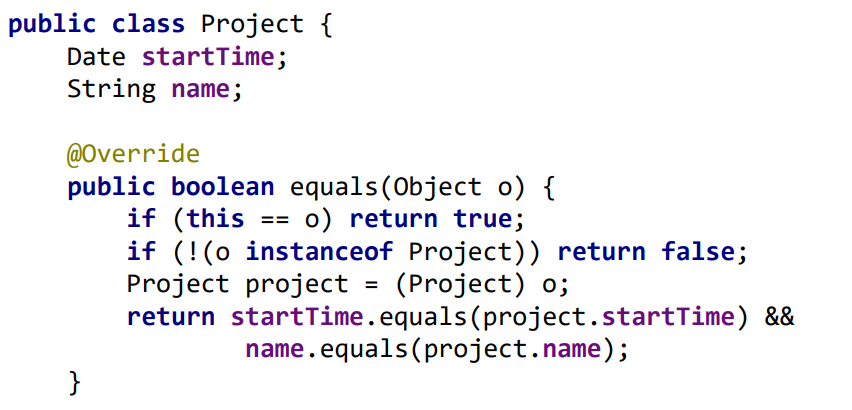
protected void finalize()

public int hashCode()

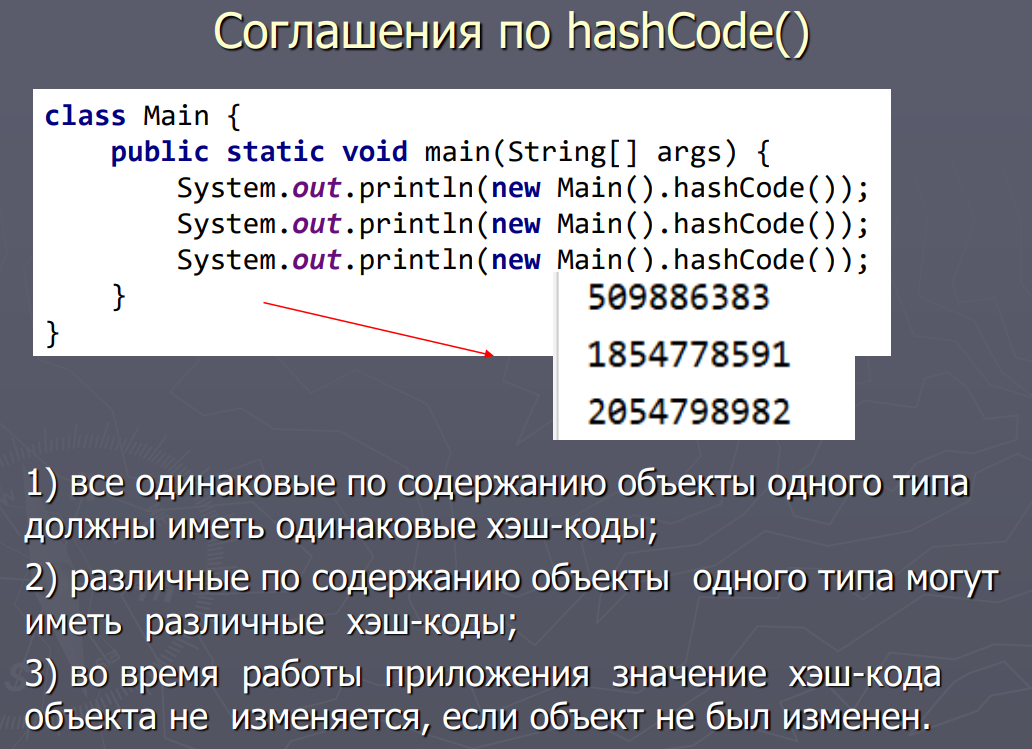
public String toString()

1. Перечислите соглашения по equlas.

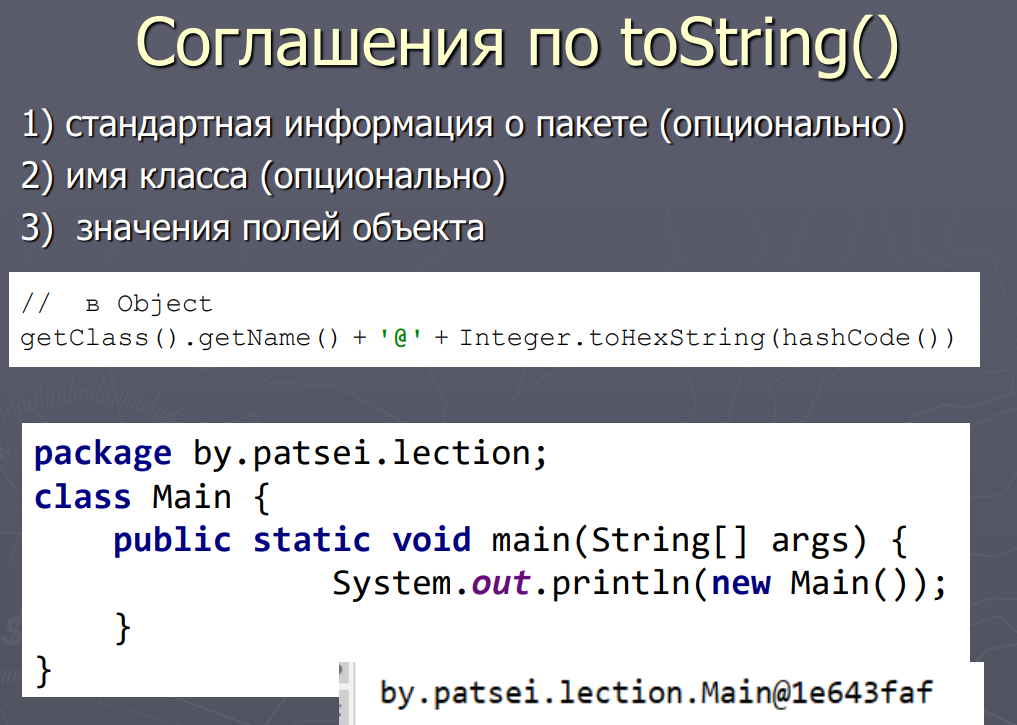




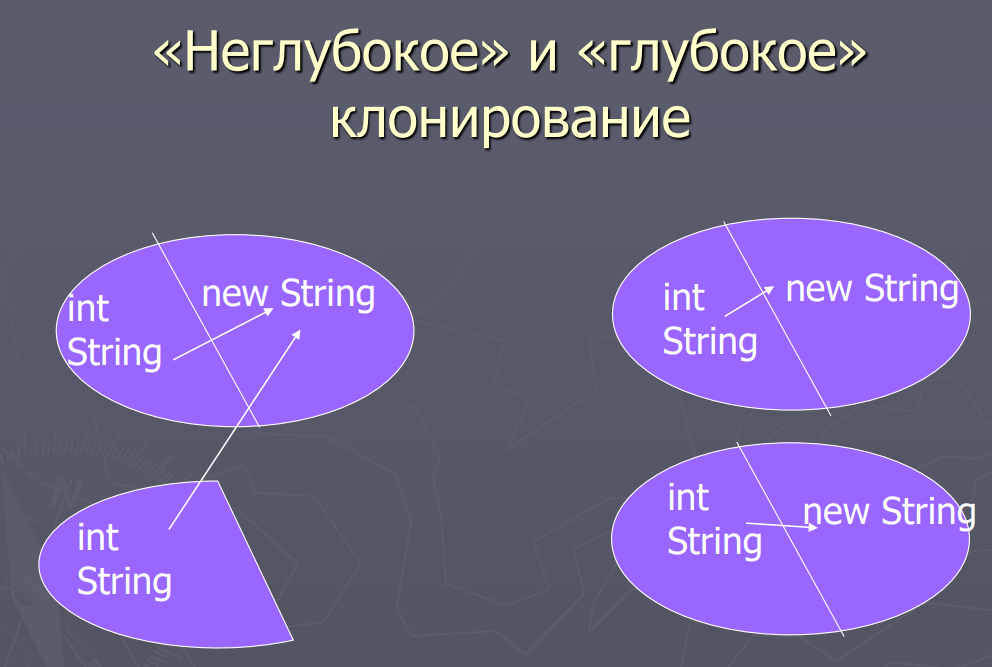
1. Перечислите соглашения по hashCode() .



1. Перечислите соглашения по toString().

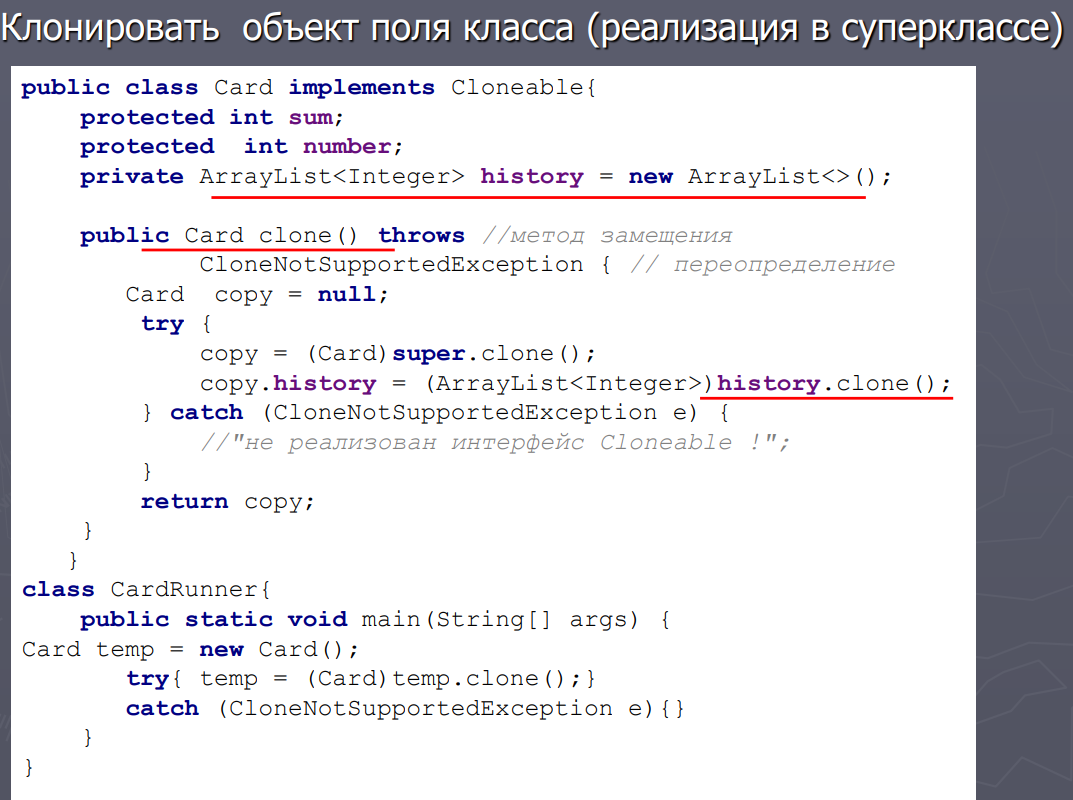


1. Поясните разницу между «неглубким» и «глубоким» клонированием? Приведите пример.



При глубоком клонировании клонируются и ссылки, и объекты.

При клонировании только ссылок используется неглубокое клонирование.



1. Как можно использовать метод void finalize()?

* **Finalize** – это защищенный нестатический метод, который определен в классе Object и, таким образом, доступен для всех без исключения объектов в Java. Этот метод вызывается сборщиком мусора до полного уничтожения объекта.

Финализаторы вызываются, когда JVM выясняет, что этот конкретный экземпляр должен быть собран как мусор. Такой финализатор может выполнять любые операции, в том числе возвращать объект к жизни.

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

Нестатические вложенные классы называются **внутренними (inner).**

Объект внутреннего класса всегда ассоциируется (скрыто хранит ссылку) с создавшим его объектом внешнего класса — так называемым внешним (enclosing) объектом.

Доступ к элементам внутреннего класса возможен из внешнего только через объект внутреннего класса.

*Правила использования:*

► Методы внутреннего класса имеют прямой доступ ко всем полям и методам внешнего класса

► Внешний класс может получить доступ к содержимому внутреннего класса только после создания объекта внутреннего класса. Доступ будет разрешен по имени в том числе и к полям, объявленным как private.

► Внутренние классы не могут содержать статические атрибуты и методы, кроме констант (final static).

►Внутренний класс может быть объявлен также внутри метода или логического блока внешнего (owner) класса. Внутренние классы имеют право наследовать другие классы, реализовывать интерфейсы и выступать в роли объектов наследования

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

Статический внутренний класс логически связанный с классом-владельцем называется **вложенным (nested)**.

Если класс вложен в интерфейс, то он становится статическим по умолчанию.

Статический вложенный класс для доступа к нестатическим членам и методам внешнего класса должен создавать объект внешнего класса, а напрямую имеет доступ только к статическим полям и методам внешнего класса.

*Правила использования:*

► Из него (самого класса) видны:

— статические свойства и методы OuterClassа (даже private).

— статические свойства и методы родителя OuterClassа public и protected. То есть те, которые видны в OuterClassе. Из его экземпляра видны:

— все (даже private) свойства и методы OuterClassа обычные и статические.

— public и protected свойства и методы родителя OuterClassа обычные и статические. То есть те, которые видны в OuterClassе.

► Его видно: — согласно модификатору доступа.

► Может наследовать:

— обычные классы.

— такие же статические внутренние классы в OuterClassе и его предках.

► Может быть наследован:

— любым классом:

— вложенным

— не вложенным

— статическим

— не статическим!

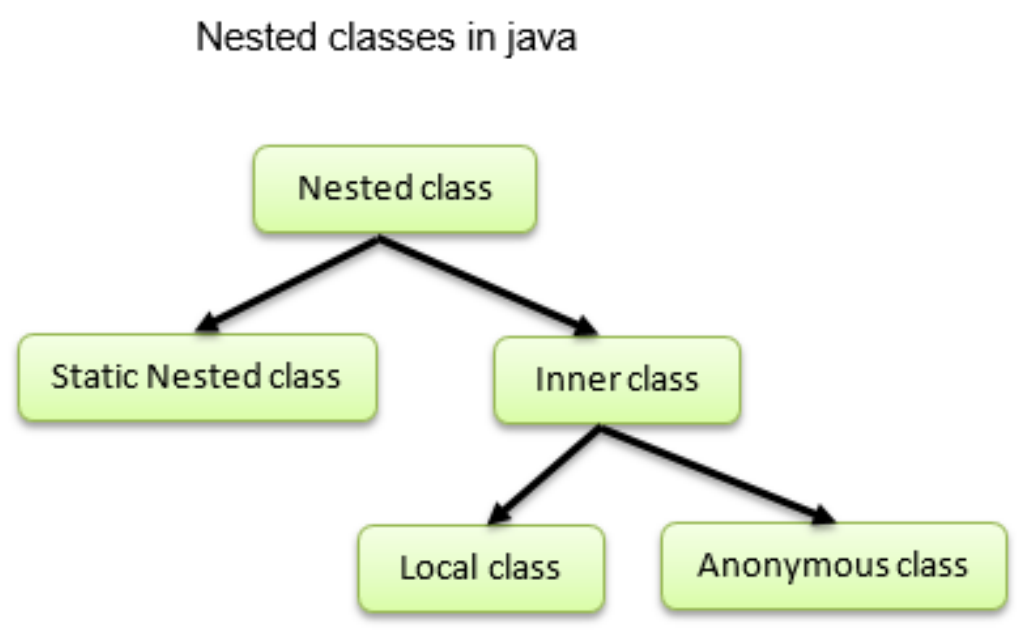
Может имплементировать интерфейс.

Может содержать:

— статические свойства и методы.

— не статические свойства и методы.

► Для создания объекта вложенного класса объект внешнего класса создавать не надо.



1. Что такое анонимные (anonymous) классы?

* **Анонимный класс** является разновидностью локального класса без

полноценного имени. Анонимно могут быть определены интерфейсы,

перечисления и аннотации.

Анонимные (безымянные) классы применяются для:

► придания уникальной функциональности отдельно взятому экземпляру,

► для обработки событий,

► реализации блоков прослушивания,

► реализации интерфейсов,

► запуска потоков,

► для реализации (переопределения) нескольких методов

► создания собственных методов объекта и т. д.

Конструктор анонимного класса определить невозможно!

1. Правила определения и наследования интерфейсов.

В Java **интерфейс**-это абстрактный тип, содержащий набор методов и постоянных переменных. Это одна из основных концепций в Java и используется для достижения абстракции, полиморфизма и множественного наследования.

Мы можем реализовать интерфейс в классе Java с помощью ключевого слова **implements**.

В интерфейсе нам разрешено использовать:

* константы переменные
* абстрактные методы
* статические методы
* методы по умолчанию.

Мы также должны помнить, что:

* мы не можем создавать экземпляры интерфейсов напрямую
* интерфейс может быть пустым, в нем нет методов или переменных
* мы не можем использовать слово final в определении интерфейса, так как это приведет к ошибке компилятора
* все объявления интерфейса должны иметь модификатор доступа public или по умолчанию; модификатор abstract будет добавлен компилятором автоматически
* переменные интерфейса являются общедоступными, статическими и окончательными по определению.

Интерфейсы, как и классы, могут наследоваться:

interface BookPrintable extends Printable{

    void paint();

}

При применении этого интерфейса класс Book должен будет реализовать как методы интерфейса BookPrintable, так и методы базового интерфейса Printable.

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

* **Исключение** — это проблема(ошибка) возникающая во время выполнения программы. Все исключения в Java являются объектами. Поэтому они могут порождаться не только автоматически при возникновении исключительной ситуации, но и создаваться самим разработчиком.

**Checked** исключения, это те, которые должны обрабатываться блоком catch или описываться в сигнатуре метода. **Unchecked** могут не обрабатываться и не быть описанными.

**Unchecked** исключения в Java — наследованные от *RuntimeException*, **checked** — от *Exception* (не включая unchecked).

**Checked** исключения отличаются от **Unchecked** исключения в Java, тем что:

Наличие\обработка **Checked** исключения проверяются на этапе компиляции. Наличие\обработка **Unchecked** исключения происходит на этапе выполнения.