






# ЗЛаба

 Assign	
 Status	Completed
 Priority	
 Date Created	@January 25, 2022 10:20 PM
 Due Date	

Вопросы:

1.Перечислите состав класса.

Поля, свойства , методы, внутренние классы, индексаторы и т.д.

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

Static . Метод, поле, класс . Используется, чтобы показать, что данное сущность не принадлежит экземпляру данного класса.

Abstract . Класс, метод . Экземляр такого класса не может быть создан и может содержать абстрактные методы, которые должны переопределяться потомками этого класса.

Final . Поле, метод, класс . Поля не может быть изменено, метод переопределён , а класс - унаследован .

3. Где могут использоваться слова super и this?

super для обращения к экземпляру родителя, this - на экземляр текущего класса . С помощью этих слов мы можем обращаться к членам родителя и текущего класса ,например вызывать родительский конструктор.

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 C/C++ или Ассемблера, и скомпонован в виде динамических библиотек, позволяет не использовать статическое связывание. Это даёт возможность вызова функции C/C++ из программы на Java, и наоборот

5. Что такое логический и статический блок?

Блоки инициализации представляют собой наборы выражений инициализации полей, заключенные в фигурные скобки и размещаемые внутри класса вне объявлений методов или конструкторов. Логические блоки инициализации выполняются последовательно при создании каждого экземпляра, статические — лишь при первом создании экземпляра.

6. Определите параметризованный класс.

```
public class Subject <T1, T2> {  
private T1 name;  
private T2 id;  
}
```

7. Как используется метасимвол «?»

Подстановочный символ указывающий на неизвестный тип.

8. Какие существуют generic-ограничения?

- Невозможно выполнить явный вызов конструктора generic-типа
- generic-поля не могут быть статическими
- Статические методы не могут иметь generic-параметры или обращаться к generic полям.

9. Что могут содержать перечисления? Приведите пример

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

```
public enum AirType {  
    Boeing707 (430), TY144(144), A320(480);  
    private int freePlaces; // поле вместительность  
    AirType() { // конструктор класса перечисления  
    }  
    AirType(int place) { // конструктор класса перечисления  
        freePlaces = place;  
    }  
    public int getFreePlaces() {  
        return freePlaces;  
    }  
    public void setFreePlaces(int place) {  
        freePlaces = place;  
    }  
    @Override  
    public String toString() {  
        return String.format("%s : free places = %d", name(),  
            freePlaces);  
    }  
}
```

используются для сохранения дополнительной информации

Все конструкторы вызываются автоматически при инициализации любого из элементов

не может быть public и protected

Перечисление не может быть суперклассом

10. Какие существуют ограничения для перечислений?

## Перечисления ограничения:

- ▶ 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?

## Методы подставки

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

12. Состав класса Object.

## Переопределение из Object для информационных классов

```
protected Object clone() ;  
public boolean equals(Object ob)  
public Class<? extends Object> getClass()  
protected void finalize()  
public int hashCode()  
public String toString()  
  
    notify(), notifyAll() и  
    wait(), wait(int millis)
```

13. Перечислите соглашения по equals.

# Соглашения по equals

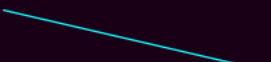
```
//реализация из Object  
public boolean equals(Object obj) {  
    return (this == obj);  
}
```

- ▶ рефлексивность
- ▶ симметричность
- ▶ транзитивность
- ▶ непротиворечивость
- ▶ ненулевая ссылка при сравнении с литералом null всегда возвращает значение false

14. Перечислите соглашения по hashCode() .

## Соглашения по hashCode()

```
class Main {  
    public static void main(String[] args) {  
        System.out.println(new Main().hashCode());  
        System.out.println(new Main().hashCode());  
        System.out.println(new Main().hashCode());  
    }  
}
```



509886383

1854778591

2054798982

- 1) все одинаковые по содержанию объекты одного типа должны иметь одинаковые хэш-коды;
- 2) различные по содержанию объекты одного типа могут иметь различные хэш-коды;
- 3) во время работы приложения значение хэш-кода объекта не изменяется, если объект не был изменен.

15. Перечислите соглашения по toString().

# Соглашения по toString()

- 1) стандартная информация о пакете (опционально)
- 2) имя класса (опционально)
- 3) значения полей объекта

```
// в Object  
getClass().getName() + '@' + Integer.toHexString(hashCode())
```

```
package by.patsei.lection;  
class Main {  
    public static void main(String[] args) {  
        System.out.println(new Main());  
    }  
}  
  
by.patsei.lection.Main@1e643faf
```

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

Поверхностное копирование копирует настолько малую часть информации, насколько это возможно. По умолчанию, клонирование в Java является поверхностным, т.е. не знает о структуре класса, которого он копирует. При клонировании, JVM делает такие вещи:

1. Если класс имеет только члены примитивных типов, то будет создана совершенно новая копия объекта и возвращена ссылка на этот объект.
2. Если класс содержит не только члены примитивных типов, а и любого другого типа класса, тогда копируются ссылки на объекты этих классов. Следовательно, оба объекта будут иметь одинаковые ссылки.

Глубокое копирование дублирует все. Глубокое копирование — это две коллекции, в одну из которых дублируются все элементы оригинальной коллекции. Мы хотим сделать копию, при которой внесение изменений в



любой элемент копии не затронет оригинальную коллекцию.

Примеры сами находите.

17. Как можно использовать метод `void finalize()`?

## ► 2) `protected void finalize()`

- вызывается перед сборкой мусора
- метод может быть вообще не выполнен!
- если возникнет исключительная ситуация, она будет проигнорирована

```
class Main {  
    @Override  
    protected void finalize()  
    {  
        System.out.println("finalize method called");  
    }  
    public static void main(String[] args) {  
        Main t = new Main();  
        System.out.println(t.hashCode());  
        t = null;  
        System.gc();  
        System.out.println("end");  
    }  
}
```

```
509886383  
end  
finalize method called
```

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

Нестатические вложенные классы называются внутренними (inner).

## 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

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

Статический внутренний класс логически связанный с классом владельцем называется вложенным

**Из него (самого класса) видны:**

- статические свойства и методы OuterClassa (даже private).
- статические свойства и методы родителя OuterClassa public и protected. То есть те, которые видны в OuterClassa.

**Из его экземпляра видны:**

- все (даже private) свойства и методы OuterClassa обычные и статические.
- public и protected свойства и методы родителя OuterClassa обычные и статические. То есть те, которые видны в OuterClassa.

**Его видно:**

- согласно модификатору доступа.

**Может наследовать:**

- обычные классы.
- такие же статические внутренние классы в OuterClassa 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.**

20. Что такое анонимные (anonymous) классы?

## 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

21. Правила определения и наследования интерфейсов.

# Интерфейсы

спецификация функциональности, которую должен реализовывать каждый класс, его имплементирующий

- ▶ Виды:
- ▶ интерфейсы, определяющие функциональность для классов
- ▶ интерфейсы, придающие классу определенные свойства (Cloneable, Serializable)

```
[public] interface Имя  
    [extends Имя1, Имя2,..., ИмяN]  
{ /* */ }
```

- ▶ методы автоматически - public и abstract –реализации нет
- ▶ поля автоматически— public, static и final – только проинициализированные

```
public interface MyInterface {  
    int MY_CONSTANT = 9;  
}
```

- ▶ - static методы
- ▶ - private [+static] методы
- ▶ - default методы
- ▶ - другие типы

## +Статические методы

- ▶ Public по умолчанию
- ▶ Не наследуются
- ▶ Нельзя переопределять в классах при реализации – статик метода

```
public interface MyInterface {  
  
    // This works  
    static int foo() {  
        return 0;  
    }  
  
    // This does not work,  
    // static methods in interfaces need body  
    static int bar();  
}
```

## +Методы по умолчанию

- ▶ наследуются классами, реализующими интерфейс
- ▶ классы могут переопределять их поведение.

```
public interface MyInterface {  
    default int doSomething() {  
        return 0;  
    }  
}
```



## +Private методы интерфейса

- ▶ есть тело, они не абстрактные
- ▶ могут быть как статическими, так и нестатическими
- ▶ не наследуются классами, реализующими интерфейс, и интерфейсами
- ▶ могут вызывать другие методы интерфейса
- ▶ могут вызывать другие приватные, абстрактные, статические методы или методы по умолчанию
- ▶ могут вызывать только другие статические и приватные статические методы

```
public interface Logging{  
    private void log(String message){  
        //реализация  
    }  
}
```

## + Вложенность

- ▶ Интерфейсы могут содержать внутри себя другие интерфейсы и классы. Они `public`, `static`

```
public interface MyInterface {  
    class MyClass {  
        //...  
    }  
  
    interface MyOtherInterface {  
        //...  
    }  
}
```

## + Перечисления и Аннотации

```
public interface MyInterface {  
    enum MyEnum {  
        FOO, BAR;  
    }  
  
    @interface MyAnnotation {  
        //...  
    }  
}
```

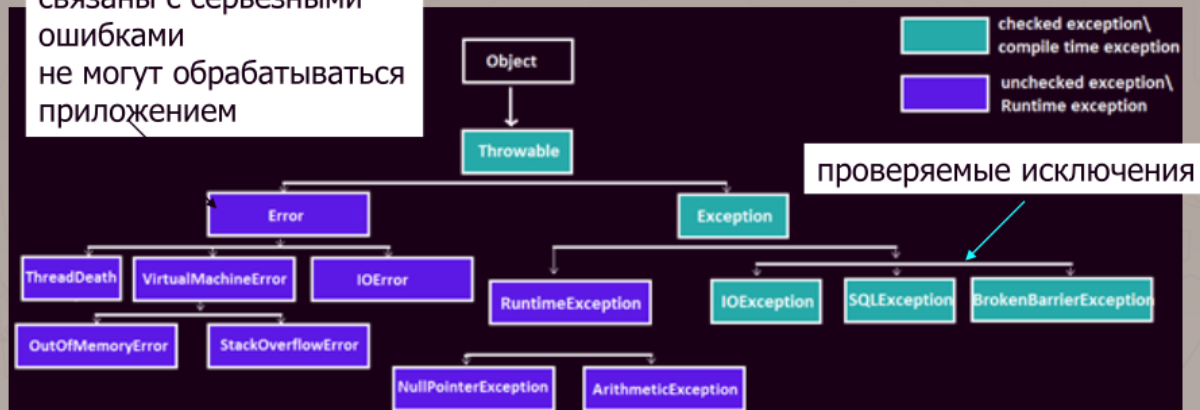
## +Параметризация интерфейсов

```
interface Box<T> {  
    void insert(T item);  
}  
  
class ShoeBox implements Box<Shoe> {  
    public void insert(Shoe item) {  
        //...  
    }  
}
```

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

# Иерархия исключений и ошибок

возникают только во время выполнения программы  
связаны с серьезными ошибками  
не могут обрабатываться приложением



Проверяемые - должны быть обработаны в методе, который может их генерировать, или включены в throws-список метода для дальнейшей обработки в вызывающих методах

Проверяемые исключения - исключения, на которые код проверяет компилятор .  
Непроверяемые - исключения рантайма, которые компилятор не проверяет