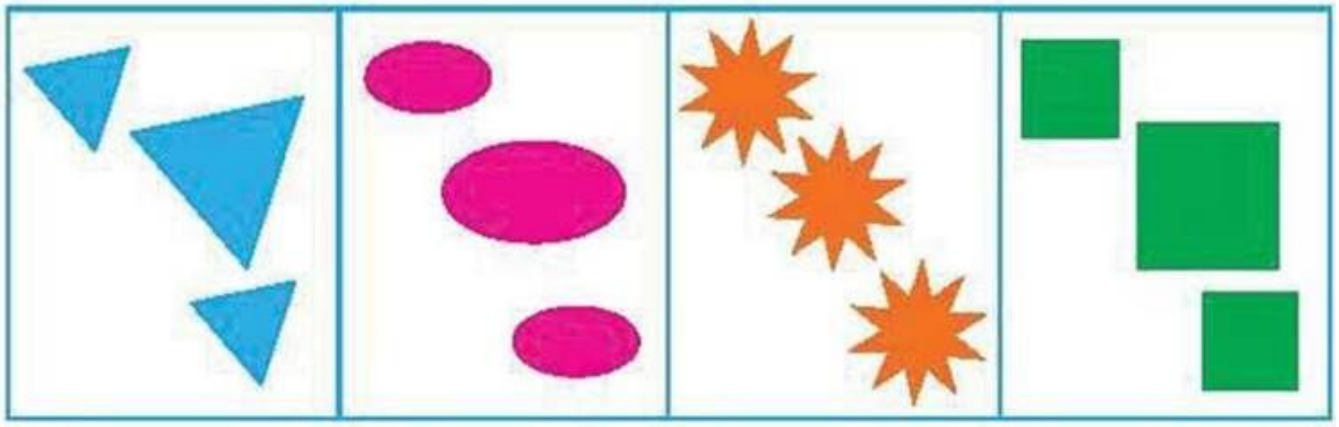
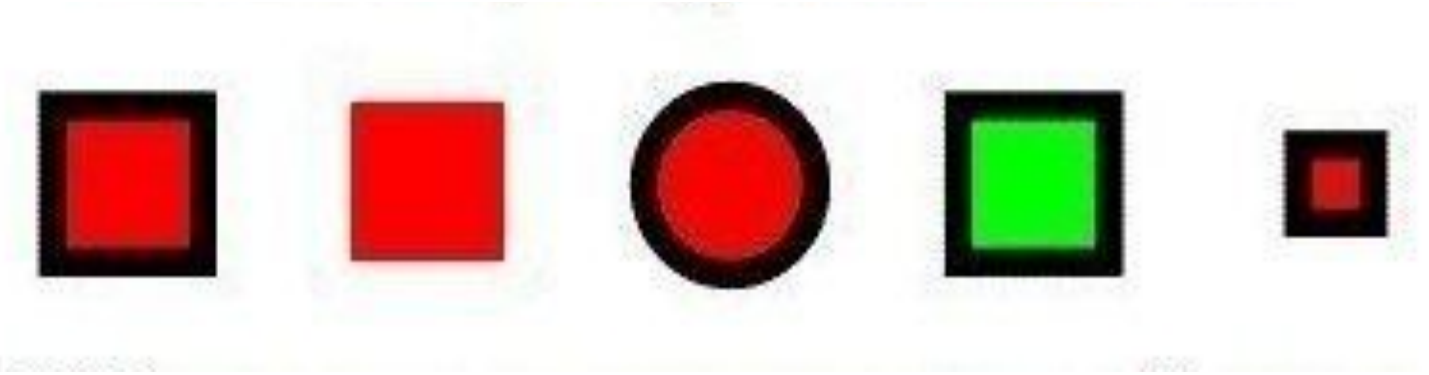
Лекция 3

Что лишнее



Что лишнее



Пакеты

1. Проект может состоять из сотен или больше разных классов.
2. Чтобы как-то их упорядочить используются «пакеты»!
3. Пакеты это тоже самое что и папки. Например

com

oracle

printers

Canon.java

Epson.java

1. Если ваш класс принадлежит пакету, в первой строчке файла надо указать пакет

package com.oracle.printers;

2. *Пакеты отсчитываются от корня проекта (обычно это директория src/)*

Импорт

1. Если вы хотите в своем классе использовать другой класс, нужно этот класс подключить к вашему классу:
   import com.oracle.printers.Canon;

2. Если подключаемый класс находится в том же пакете(папке) что и исходный класс, то его не нужно подключать.
3. При подключении указывается полное название пакета и название класса.
4. Можно подключить сразу все классы в указанном пакете использую звездочку:
   import com.oracle.printers.*;

Наследование

1. **Наследование – механизм позволяющий расширять/изменять структуру уже имеющихся классов, создавая на основе их другие.**
2. **Т.е. Если вы хотите создать новый класс похожий на уже имеющийся, но немного другой, при этом не меняя имеющийся класс, то вы можете его унаследовать.**
3. **Наследуемый класс называется родительским классом, а наследующий дочерним классом или потомком.**
4. **Дочерний класс будет иметь при себе все свойства и методы родительского класса.**
5. **При этом дочерний класс может добавлять новые или переопределять старые.**
6. **Дочерний класс не имеет доступа к**

Наследование


```
class Car{
        String model;
        String color;
        public void go(){
                …
        }
}
…
class UsedCar extends Car{
        int kilometers;

        public boolean isWorking(){
        }
}
…

UsedCar bmw = new UsedCar();
bmw.model = “BMW x5”;   //есть свойства родительского класса
bmw.isWorking();
```

Наследование: еще пример


```
class Summator {
        public int sum(int num1, num2){
                return num1+num2;
        }
}
…
class Arithmetician extends Summator{
        public int multiply(int num1, int num2){
                return num1 + num2;
        }
}
…

Summator summator1 = new Summator();
summator1.sum(10, 93);     //результат 103
summator1.multiply(2,3);   //У Summator нет метода multiply!

Arithmetician r2d2 = new Arithmetician();
r2d2.sum(23, 34);     //Этот метод унаследован от Summator
r2d2.multiply(5, 100);
```

Наследование: Переопределение


```
class StandardTax {
        public double calculateTax(double sum){
                return sum * 0.12;
        }
}
…
class EasyTax extends StandardTax{
        public double calculateTax(double sum){  //Переопределение метода
                return sum * 0.05;
        }
}
```

1. Все классы, если не указан родительский класс, наследуют от класса Object. Т.е. все классы являются потомками класса Object.
2. Так как потомок имеет те же свойства что и родитель, потомок можно использовать вместо родителя. Например:
3. Если есть метод который принимает объект класса Car. То в этот метод мы можем передать объект класса UsedCar, т.е. потомка класса Car.

Приведение типов

1. Вы можете создать объект любого класса и хранить его в переменной типа Object, так как Object родитель для всех.
2. Object document = new Document();
3. Но теперь этот объект document мы не можем передать методу, который принимает тип Document. Например метод: *public void print(Document doc)*. Потому что мы объявили переменную document как Object. И родитель не может заменять потомка.
   printer.print(document); //Ошибка!
4. Родитель не может заменять потомка, потому что не имеет всех свойств потомка.
5. А потомок может заменять родителя, потому что имеет все свойства родителя!
6. «Но наш объект document на самом же деле типа Document, а не Object! Потому что мы создаем new Document()» - скажете вы, и будете правы.
7. Поэтому чтобы передать объект класса Document, но который объявлен как Object в метод который принимает тип Document, мы должны явно указать его тип или привести в другой тип:
   printer.print((Document)document);

Наследование

1. Родитель не может заменять потомка, потому что не имеет всех свойств потомка.
2. А потомок может заменять родителя, потому что имеет все свойства родителя!
3. Т.Е. Банан это всегда фрукт. Но фрукт не всегда банан!
4. Если кто-то может есть только фрукты, вы можете дать ему банан. Потому что банан это фрукт.
5. Но если кто-то ест только бананы, вы не можете ему дать какой-то фрукт, потому что фрукт это не всегда банан. Поэтому вы явно должны указать что ваш фрукт это банан.

```
class Banana extends Fruit {}   //Банан наследует Фрукт
class Monkey{
        public void eat(Banana banana){}     //ест только бананы
        public void throw(Fruit any){}    //а кидается фруктами
}
Friut friut1 = new Fruit();
monkey.eat(fruit1);    //Ошибка

Banana banan = new Banana();
monkey.throw(banan);   //Не ошибка, потому что Банан потомок Фрукта
```

Конструкторы

1. Методы которые вызываются при создании объекта класса и называются также как и сам класс это – Конструкторы.
2. Конструкторы создают и инициализируют объект класса.
3. Конструкторы всегда идут после ключевого слова new:

```
Printer printer = new Printer();
Printer printer2 = new Printer("Canon", 2900);
```

Конструкторы!!!

4. Конструкторы могут иметь параметры:
   `public Printer(String name)`
5. Могут и не иметь параметров:
   `public Printer()`
6. Один класс может иметь несколько конструкторов
7. Так как конструктор всегда выполняется первым, в нем можно и нужно инициализировать все поля(свойства) класса.

```
class Printer{
        public String name;
        public int version;
        public Printer(){
                this.name = "Printer";
                this.version = 0;
        }
```

Интерфейсы

1. Интерфейс – это способ или протокол взаимодействия между разными системами.
2. Например: Человек взаимодействует с компьютером через устройства ввода и вывода – это человеко –машинный интерфейс.
3. Или: Пользователь сайта взаимодействует с сайтом через кнопки, текст и другие html элементы – это пользовательский интерфейс.

4. Также при отдельной разработке разных модулей одной большой системы вам необходимо организовать взаимодействие между модулями системы.
5. При этом вы не можете заранее знать как будут реализованы другие части системы.
6. Но вы можете дать определенные требования к реализации подключаемых частей к вашему модулю. А конкретно вы можете определить протокол взаимодействия или интерфейс.

7. В Java тоже есть понятие интерфейсов, и интерфейсы Java применяются к классам. Вы можете описать какие точки входа и выхода должен иметь какой-либо класс. Т.Е. Какие

Интерфейсы

1. Допустим есть некий класс Говоритель(Speaker) у которого есть метод «сказать» (speak), который может воспроизвести текст голосом. Поэтому этот метод может принимать любой класс который имеет метод getText(), который возвращает текст.
2. В Java, требование «*любой класс который имеет метод getText(), который возвращает текст*» мы можем описать с помощью интерфейса так:

```
public interface HasText{
        String getText();
}
```

3. А класс Speaker может выглядеть примерно так:

```
public class Speaker{
        public void speak(HasText object){
                String text = object.getText();
                …
        }
}
```

4. Так как интерфейс это просто описание, в нем нет ничего кроме описания методов, нет тел методов и нет свойств.
5. Также нельзя создавать объекты из интерфейсов

```
HasText text = new HasText();  //Ошибка, HasText не класс
```

Интерфейсы

1. Но интерфейсы можно реализовывать в классах. И там где требуется интерфейс, передать объект класса, который реализует этот интерфейс.
2. Например наш класс Document имеет текст и он вполне может реализовать интерфейс HasText. Это делается так:

```
class Document implements HasText{
        …   //тут все описание класса

        public String getText(){   // это реализация метода, который
              return this.title;      // описан в интерфейсе HasText
        }
}
```

3. Теперь мы можем наш документ передавать Говорителю, чтобы он проговорил его:

```
Speaker speaker = new Speaker();
Document doc = new Document(“Онегин”, “Мой дядя…”);
speaker.speak(doc);
```

Интерфейсы

1. Что значит реализовать интерфейс?
2. Например в театр впускают только тех, кто одет классически, т.е. Требуется интерфейс «Одет В Классическую Одежду». И пока вы не реализуете это требование, вы не можете зайти в театр.

3. Или чтобы получить пи озетки, ваше устройство должно реализовать ин

т.е. В имеющееся устройство вы должны добавить такую вилку, и таким образом вы реализуете интерфейс, который требует розетка.

Спасибо!