Object-oriented programming

Lecture №3

Inheritance, Generics, Containers

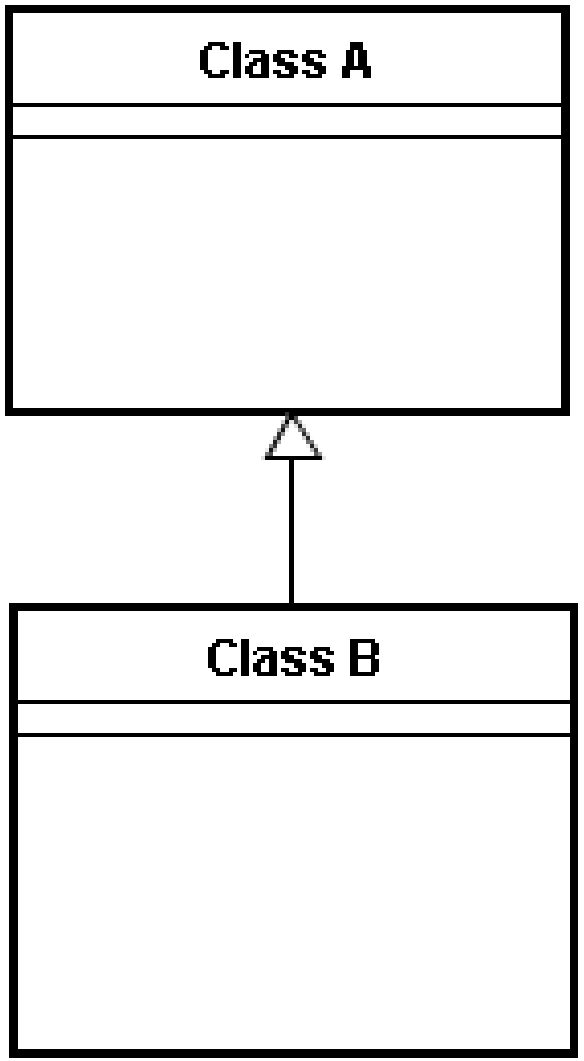
PhD, Alexander Vlasov

Question?

- Object
- Class
- Primitive Data Types
- Packages

Inheritance (is-a)

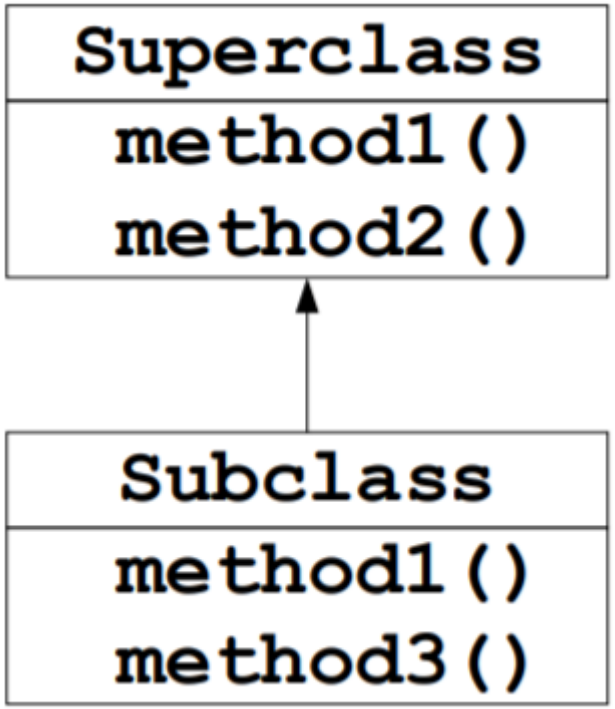
- Class B is class A
- Code reuse



Inheritance (Java)

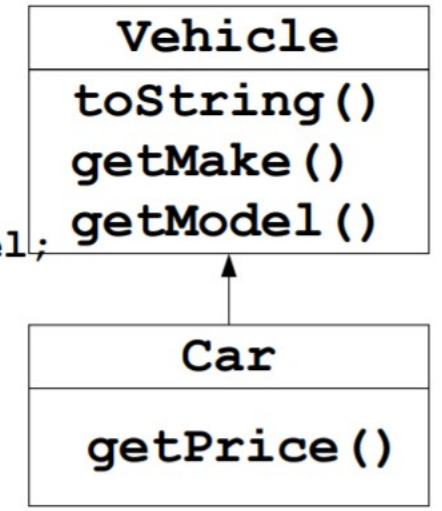
- Все типы данных (кроме базовых) наследуют класс java.lang.Object
- Только один Superclass

```
class Subclass extends Superclass {
// <class body>
}
```

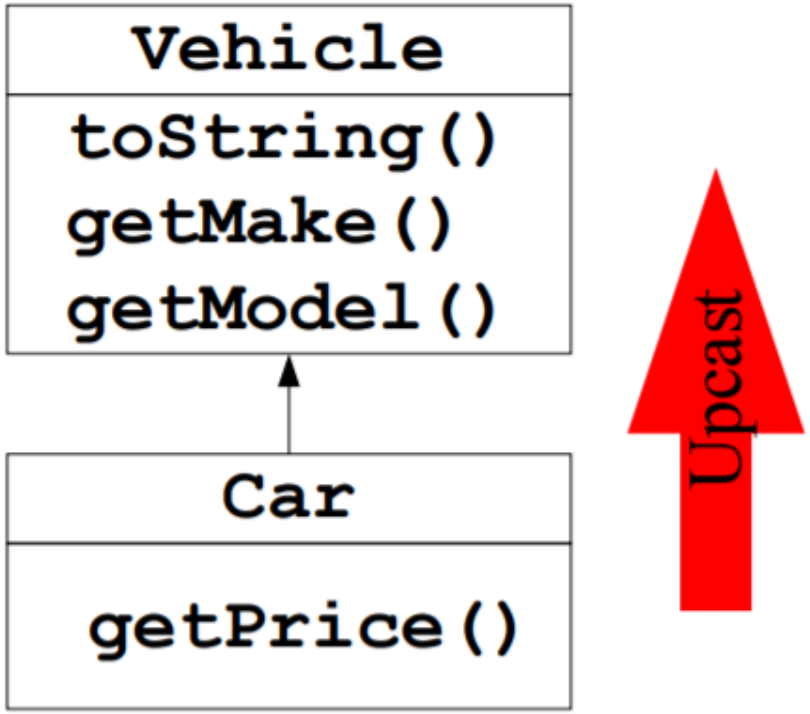


Inheritance (Java)

```
public class Vehicle {
    private String make;
    private String model;
    public Vehicle() { make = ""; model = "";}
     public String toString() {
         return "Make: " + make + " Model: " + model;
     }
    public String getMake(){ return make; }
    public String getModel() { return model; }
}
public class Car extends Vehicle {
    private double price;
    public Car() {
        super(); // called implicitly can be left outs
        price = 0.0;
    }
    public String toString() { // method overwrites
        return "Make: " + getMake() + " Model: " + getModel()
                        + " Price: " + price;
    }
    public double getPrice(){ return price; }
}
```



Upcasting (Treat a subclass as its superclass)




```
// example
Car c = new Car();
Vehicle v;
v = c;                  // upcast

v.toString();   // okay
v.getMake();    // okay
//v.getPrice(); // not okay
```

Array

- MyClass array[] = new MyClass[3];
- array[0].a();
- array[0] = new MyClass();
- MyClass array2[][] = new MyClass[2][];
- array2[0] = new MyClass[5];
- array2[1] = new MyClass[4];
- int array3[] = new int[3]; array3[0] = 0;
- int array4 [] = new int [] {1,2,3,4};
- MyClass array5[] = {new MyClass(“a”), new MyClass(“b”)};

Array

- class Array is class Object
- public final int length
- public Object clone()

```
class Test {
        public void test1() {
            int a1[][] = { { 1 , 2}, null };
            int a2[][] = (int[][]) a1.clone();
            System.out.print( (a1 == a2) + " ");
            System.out.println(a1[0] == a2[0] && a1[1] == a2[1]);
            a1[0][0]=0;
            System.out.println(a2[0][0]);
            }
}
```

for

```
class Ex {
        public static void main(String args[] ) {
                for (String s : args ){
                        System.out.println(s);
                }
                for (int i = 0; i < args.length; i++) {
                        System.out.println( args[i] ) ;
                }
        }
}
```

Параметризованные классы

- В первых версиях языка Java контейнеры кроме массивов содержали объекты класса Object
- ArrayList a; a.add(new Integer(1)); Integer b=(Integer)a[0];
- ArrayList<Integer> a; Integer b=a[0];
- Generics

Generics

Early JDK 5.0 :

```
public class A {
private Object ref;
public Envelope (Object arg) {
ref = arg;
}
public Object get() { return ref; }
}
…
A a = new A (new B());
…
B d = (B)e.get();
C g =  (C)e.get(); // ClassCastException in runtime:
```

JDK 5.0:

```
public class A <T> {
private T ref;
public Envelope (T arg) {
ref = arg;
}
public T get() { return ref; }
}
…
A<B> a = new A<B>(new B());
…
B m = e.get();
C a = e.get(); //compile time error:
```

Generics

Позволяют объявлять классы, интерфейсы и методы, где тип данных и ограничения на него, которыми они оперируют указан в виде параметра

Позволяют реализовать код, который будет работать единообразно с разными типами данных

В отличие от использования ссылок на тип Object предоставляют безопасные с точки зрения типизации средства обобщенного программирования

Спецификация параметра задается явно (классы, интерфейсы и ссылки) или выводится на основании аргументов (вызов методов)

Соответствие типа проверяется на стадии компиляции

Restriction

```
class A<T> {
    static T t; //compile time error
}

class A<T> {
    T t = new T(); //compile time error
    T array = new T[10]; //compile time error
}

List<Object> list = new List<String>(); //compile time error
```

Generics & extends

```
public class Box<T> {
   private T t;
   public void set(T t) {
      this.t = t;
   }
   public T get() {
      return t;
   }
   public <U extends Number> void inspect(U u){
      System.out.println("T: " + t.getClass().getName());
      System.out.println("U: " + u.getClass().getName());
      System.out.println(n.intValue());
   }
   public static void main(String[] args) {
      Box<Integer> integerBox = new Box<Integer>();
      integerBox.set(new Integer(10));
      integerBox.inspect("some text"); // error: this is still String!
   }
}
```

A type variable with multiple bounds is a subtype of all the types listed in the bound. If one of the bounds is a class, it must be specified first. For example:

```
Class A { /* ... */ }
interface B { /* ... */ }
interface C { /* ... */ }

class D <T extends A & B & C> { /* ... */ }
```

If bound A is not specified first, you get a compile-time error:

```
class D <T extends B & A & C> { /* ... */ }  // compile-time error
```

Generics

- При объявлении специфической ссылки имеющей типом generic класс или интерфейс не происходит генерация кода класса (интерфейса)
- Все объекты – специфические экземпляры generic класса разделяют один и тот же обобщенный тип
- Параметрами generic класса могут быть только типы
- На параметры generic компоненты можно наложить ограничения по наследованию определенного типа и/или интерфейса (интерфейсов)
- Исключения и перечисления не могут быть generic типами
- В качестве значений параметров при спецификации не могут использоваться примитивные типы Java

Generics, Integer & Double

```
class MyArray <T extends Number> {
        private T [ ] nums;
        public MyArray ( T [] array ) { nums = array; }
        public double average ( ) { … }
        public boolean myEquals ( MyArray <? extends Number>
        another) {...}
}
```

Generics

Generics классы (интерфейсы) могут наследоваться от обычных классов (интерфейсов) и наоборот

Возможно приведение ссылки дочернего generics класса (интерфейса) на базовый только в том случае, если совпадают значения аргументов – типов заданные при спецификации

Если классы A и B связаны наследованием (A extends B), то это не значит, что специфицированные ссылки параметризованных типов которым переданы эти классы в качестве параметров-типов совместимы между собой !