Chaining Constructors

**class Point {**

**int xCoord;**

**int yCoord;**

**Point() {**

**this(0, 0);**

**}**

**Point(int xCoord, int yCoord) {**

**this.xCoord = xCoord;**

**this.yCoord = yCoord;**

**}**

**}**

String concat ( String head, String body ) – parameters

Concat(“pesho”, “is”) – arguments

* Static elements
  + Belong to the class
  + Only one instance exists
  + Can be used to refer common property of all objects

Methods - Placed between access modifier and return type. Invoke method without creating new instance.

this & super can’t be used in static contexts

Used to initialize static data

Static blocks are executed at the time of class loading

public class Main {

static {

System.out.println(“Static block”);

}

public static void main (String[] args) {

}

}

Access modifier – protected – видим е само от класове, които наследяват нашия клас или от класовете от същия пакет. Protected = default + класове, които наследяват.

* Encapsulation hides internal data
* Encapsulation Reduces complexity
* Ensures that structural changes remain local
* Access modifiers
* Hides the implementation details

Encapsulation hides the implementation details

Class announces only a few operations (methods) available for its clients – its public interface

All data members (fields) of a class should be hidden

Accessed via properties (read-only and read-write)

No interface members should be hidden

Encapsulation == hide (encapsulate) data behind constructors and properties

**Access Level "public"**

When we use the modifier **public** in front of some element, we are telling the compiler, that this element **can be accessed from every class**, no matter from the current project (assembly), from the current package. The access level **public** defines the miss of restrictions regarding the visibility. This access level is the least restricted access level in Java.

**Access Level "private"**

The access level **private** is the one, which defines **the most restrictive level of visibility** of the class and its elements. The modifier **private** is used to indicate, that the element, to which is issued, **cannot be accessed from any other class** (except the class, in which it is defined), even if this class exists in the same package.

**Access Level “protected"**

The access level **private** is the one, which defines **the most restrictive level of visibility** of the class and its elements. The modifier **private** is used to indicate, that the element, to which is issued, **cannot be accessed from any other class** (except the class, in which it is defined), even if this class exists in the same namespace.

**Access Level “default"**

This is the default access level, i.e. it is used when there is no access level modifier in front of the respective element of a class. Members can be accessed only from the same package.

Interface methods are always **public**

Not explicitly declared with **public**

Non-interfacemethods are declared **private** / **protected**

**Exception** in java is an **object**, which signals an error or an event, which is not anticipated in the normal program flow.

* Base (parent) class
* Derived (child) class

A Class, could implement many interfaces, but could extend only single class !

Do not use the Inheritance for building a “has – a” relationships between an objects. It is a rough design mistake !

The right way of creating an object relationships is to use “is – a”.

When you have a situation to decide, do you need inheritance or composition, ask your self, does the object “is – a” or it “has – a”.

Inheriting constructors:

public Cicrcle(int x, int y, String name) {

super(x,y);

this.name = name;

}

**Access Modifiers in Java**

* **public** – access is not restricted
* **private** – access is restricted to the containing type
* **protected** – access is limited to the containing type and all types derived from it
* **no modifier** – access is limited to the current package

We are not allowed to inheriting more than one class, but we are allowed to implement multiple interfaces !

Constructors are not inherited. Inheritance is used to reduce the code and especially to reuse it. So the subclass is creating its own constructor, which is calling

the constructor of the Parent class. So it is right design decision to be forbidden, inheritance of constructors.

You should always, only add new functionalities, but never reduce or change them! Good practice is to add new methods if it is needed and never change the behaviour of already defined ones.

**UML Class Diagram – Unified Modeling Language**

* + Classes are represented by rectangles
  + Relations between classes are shown as arrows
    - Closed triangle arrow means inheritance
    - Other arrows mean some kind of associations

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

A child class may override (change) some of the parent's methods

Public abstract class Animal {

}

Public class Deer extends Animal {

}

Public class Elephant extends Animal {

}

Animal babyDeer = new Deer();

Animal babyElephant = new Elephant();

Why? Mix different data types in same collection.

public static void main(String[] args) {

Vegetarian babyDeer = new Deer();

Vegetarian babyElephant = new Elephant();

List<Vegetarian> vegetarianAnimals = new ArrayList<>();

vegetarianAnimals.add(babyDeer);

vegetarianAnimals.add(babyElephant);

}

To pass more specific object to method

Public static void call (Animal deer) instead of (Deer deer)

Abstract classes cannot be instantiated But they can be subclassed.

An abstract class may or may not

include abstract methods.

They function as base classes

which can be extended

The ability of a subclass to override a method allows a class to inherit from a superclass whose behavior is “close enough” and then to modify behavior as needed.

The argument list should be exactly the same

as that of the overridden method.

The access level cannot be more restrictive

than

the overridden method's access level.

A method declared final cannot be overridden.

A method declared static cannot

be overridden but can be re-declared.

A subclass within the same package as the instance's superclass can override any superclass method that is not declared private or final.