Object Oriented Programming in Java

Jaanus Pöial, PhD

Tallinn, Estonia

Estonian Information Technology College

Motivation for Object Oriented Programming

- Decrease complexity (use layers of abstraction, interfaces, modularity, ...)
- Reuse existing code, avoid duplication of code
- Support formal contracting between independent development teams
- Detect errors as early as possible (general goal of software engineering)

Motivation

- Object oriented approach was introduced on 1980-s to reduce complexity of programming large software systems (e.g. graphical user interfaces).
- Flat library of standard functions (common for early imperative programming languages) is not flexible enough to create complex software systems.
- Powerful and well organized object oriented framework makes programming easier – programmer re-uses existing codebase and specifies only these properties/functions she needs to elaborate/change (and framework adjusts to these changes).

Object

Object is characterized by

- State (defined by values of instance variables in Java)
- Behaviour (defined by instance methods in Java)
- Identity (defined by memory location in Java)

Object = Instance = Specimen = ...

- Instance variable (Java terminology) = (Object) field = Property = Attribute = ...
- Method = Subroutine = Function / Procedure = ...

Object

Encapsulation – data and operations on the data are integrated into whole (object = capsule)

ADT approach – set of operations is a part of data type

Data hiding – object state can be changed only by dedicated (usually public) methods - instance variables should be protected from direct modification

Object is an instance of the class. E.g. „Rex is a dog“.

Class

- Class defines common features of its objects („template“). E.g. „All dogs have a name“.
- Instantiation – creating a new object of the class.

- Subclass can be derived from the class – subclass inherits all the features of its parent class. Subclass allows to add new (specific) features and redefine (=override) inherited features. E.g. „Dog is (a special kind of) Animal“.
- If A is a subclass of B then B is superclass of A.

Class Hierarchy

- Generalization – common features of similar classes are described on the level of superclass (mental process – design the hierarchy of classes).

- Specialization – subclass is created to concretize (refine) certain general features and add specific data/operations to the subclass (process of coding).

Instance Methods and Class Methods

- Instance methods define the behaviour of an object (=instance).
  - `s.length()` - the length of string s in Java.

- Class methods can be used without creating an object (imperative style).
  - `Math.sqrt(2.)` - square root of 2.

  Keyword `static` in Java is used to define class methods.

Instance Variables and Class Variables

- Instance variables define the state of an object. Each object has individual values of instance variables. In Java, keyword `private` is appropriate.
  - `a.length` – the length of an array a.
- Class variables are common (global variables in class scope). Single value is shared between all objects. Keyword `static` in Java is used.
  - `Math.PI` – constant Pi in Java.

Message Passing

- Objects communicate in OOP system by sending messages. Message, sent to an object, is interpreted by the object and causes appropriate instance method to be executed.
- OOP systems may support:
  Early binding – method to be executed when a message is received is known at compile time (from the program text, statically).
  Late binding („true“ OOP systems) – method to be executed is chosen dynamically (decision depends on runtime type of the receiver object).

Polymorphism

Same notation has different meaning in different contexts

- Two types of polymorphism:

  Overloading – operation is redefined in subclass and is binded to the activating message statically (compile time choice).

  Java constructors support overloading.

  Overriding – operation is redefined in subclass and is binded to the activating message dynamically (runtime choice).

  Java instance methods support overriding.

Inheritance

- Subclass inherits all the variables and methods of its parent class (if it is not explicitly forbidden).
- Single inheritance – up to one superclass is allowed for each class. Class hierarhy is a tree structure. In Java, class `Object` is the root class.
- Multiple inheritance – a class may have more than one superclass. If several parents have the same property defined, it may not be clear, which one is inherited (the so-called „diamond dilemma“).

„Is-a“ vs „Is-able-to“ and „Has-a“ Relations

- Inheritance relation means „is a kind of“. It is possible to model the multiple inheritance using „is able to“ and „has a“ relations.

- Java supports only single inheritance for classes, but allows a class to implement several interfaces („is able to“ contracts).

Constructors

- Constructor is a special class method to create a new object. Memory for the object is allocated dynamically.
- E.g. `new Integer(6)` - returns a new object of type Integer in Java (object is represented by memory location).
- `Calendar.getInstance()` - returns a new object of type Calendar in Java (the so-called „factory“ method).

Names in Constructors

- Constructor name = Class name, if Java new-expression is used.
- Keyword `this` inside constructor is used to call another constructor of the same class (with different signature), constructor overloading.
- Keyword `super` is used to call a constructor of the superclass.

Destructors

- Destructor is used to destroy the object and free the memory. There are no implicit destructors in Java – garbage collection is used instead.
- Garbage collection is „hidden“, sometimes it is possible to make a suggestion to clean up.

Abstract Features

- It is reasonable to define common features of similar classes in superclass (to reduce the duplication of code). But... sometimes it is impossible to implement these features in superclass.
- E.g. *circle*, *square*, *triangle*, ... are *figure*s and have an *area* as a common feature. But for each figure the method to calculate its area is different. It is said that *area* is an abstract feature of a *figure* and its implementation is delegated to corresponding subclasses.

Abstract Classes and Interfaces

- Abstract class has some abstract features that are not implemented. It is impossible to create instances of abstract classes. The role of an abstract class is to be a parent class for its subclasses.
- Interface is a pure abstract class without any implementation. It serves as a contract between programmers – certain class implements an interface, if it defines all the methods listed in interface description.
- Functional interface introduced in Java 8 has one method that can be described using functional style.

Keywords “extends” and “implements”

Class hierarchy in Java

```
class A extends B implements C, D { ... }
```

Class A is a subclass of B (inheritance applies) and also it provides all the methods listed in interfaces C and D (no „diamond dilemma“ here).

If extends-part is missing, a class is a subclass of class `Object`

Object Identity and Equality

Object identity in Java is defined by the memory location returned by the new-expression that allocates memory for the object and activates corresponding constructor.

Object is represented by this address, there is no difference between reference and object in Java.

`o1==o2` tests object identity (it means „o1 is the same object as o2“).

To test the object equality, use `o1.equals(o2)` („o1 is equal to o2“).

Sending Messages

Messages to objects are sent using dot-operator:

```
// create o1
Object o1 = new Object();
// send message toString() to the object o1
String s1 = o1.toString();
```

Class methods use the same syntax, but message is sent to the class:

```
double s2 = Math.sqrt (2.0);
```

Receiver of the Message

- In method body the receiver object is referred as `this`.
- If message is sent without indicating the receiver, `this` object (in case of instance method) or current class (in case of class method) is assumed.
- If we need to call a superclass method, we use keyword `super` as the receiver.

Overriding

- Methods are binded to messages during runtime in Java (late binding).
- To change the behaviour of Java framework, a programmer can redefine (override) methods in subclass.
- If an object receives a message, system has to choose appropriate method to be executed. Search is performed bottom-up, starting from the most specific class (runtime class of the object), then its superclass, etc. up to the root class `Object`.

Important Object Methods

Important methods to override:

`toString` – textual representation of an object,

`equals` – predicate to decide, whether two objects are equal, important with testing frameworks like JUnit

`clone` – create a clone of an object (different identity, but equal content)
Deep clone vs. shallow clone.

`hashCode` – influences behaviour of certain collections (like HashMap)

Interfaces

Generic built-in methods will work for our class, if we implement corresponding interface.

Example.

```
public int compareTo (Object o)
```

is a method in Comparable interface. Making a class Comparable (by implementing compareTo) gives a lot of API methods for free, e.g. sort, max, min, ...

o1.compareTo(o2) < 0 , if o1 is less than o2

== 0 , if o1 equals to o2

> 0 , if o1 is greater than o2

Examples

- Pets.java
- Phones.java
- Num.java
- Complex.java