Java course

Jean-Paul COSAL

September 9, 2018

Introduction

Who I am

Who you are

- Current project
- Previous experience with software development
- Expectations about this course?
- ...

Course Objectives

- Java Basics.
- Object Oriented Programming Basics
- Insights of work and processes in the Industry
- **Develop interesting skills**

Course Objectives

- Java Basics.
- Object Oriented Programming Basics
- Insights of work and processes in the Industry
- **Develop interesting skills**
    - Autonomy
    - Curiosity
    - *Discipline*
    - *Go further*
    - *Defend your cause*
    - ...
- *Be prepared to integrate the Industry*

Day 1: Reminders

My first program

My first program

HelloWorld.java

My first program

HelloWorld.java

```java
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

Java main features

- Object oriented language
- Portable
- Memory managed
- Comes with a rich development environment
- ...

Java basics

Variables and primitive types

byte, short, int, long, float, double, char, boolean

```
int myNumber; # Variable declaration
myNumber = 5; # Variable assignment

String s = "My String"; # Declaration and assignment
```

Conditionals

- Any expression that evaluates to `true` or `false`

```
boolean result;
result = 1 == 3; // a equal to b - false
result = 1 < 3; // true
result = 1 > 3; // false
```

- Compose expressions with || and &&
  - Important notice: **Lazy** evaluation
- Use () to enforce **precedence**

```
int a,b;
if (a > 1 && (a < 9 && b != 3)) /*do something*/;
```

Collections

A collection represents a group of objects

- *Vector* (1D array)

```
int[] a = new int[5];
a[0] = 5;
int[] b = {1, 2, 3};
```

- *Matrix* (nD array)

```
int[][] a = new int[2][2];
a[0][0] = 5;
int[][] b = { {1, 1}, {2, 2}};
```

Collections(cont.)

- Set and List
  Main interfaces of standard collections

```
boolean add(E e);
int size();
void clear();
Iterator<E> iterator();
...
```

Control structures

- if/else if/else
- switch
- for
- while
- do while
- for(each)

Functions

```
public int sum(int a, int b) {
    // Do something here
}
```

Mind:

- **The return value**
- **The arguments**

Excercises

Introducing `Git`

- `SCM` main features
  - Backup
  - Keep track of code changes
  - Collaborative work
- Configuration Management main features

Excercises

Introducing `Git`

- `SCM` main features
  - Backup
  - Keep track of code changes
  - Collaborative work
- Configuration Management main features
  - Manage the lifecycle of a product
  - Version delivery management
- Git main feature
  - **distributed** `SCM`

Git cheat sheet

- git clone <remote_repo>
  - Get a working copy (*clone*) of a repository
- git pull
  - Synchronize, i.e. get changes, with remote repository
- git push
  - Publish your changes to remote repository
- git branch <my_branch>
  - Create branch my_branch
- git checkout <my_branch>
  - Switch to branch my_branch

Raw Java

- Mean function

$$\mathrm{moy}\colon (x_1, x_2, \ldots, x_n) \mapsto \frac{1}{n}\sum_{i=1}^{n} x_i$$

  - Code
  - Compile (write a script)
  - Execute & test (write a script)

Introducing `Eclipse`

- Mean function
  - Create a project
  - Build & run
  - Unit test
- Matrix multiplication
  - `int[][] A = new int[m][n]`
  - `int[][] B = new int[n][o]`
  - `int[][] C = multiply(A, B)`

$$c_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j}, \qquad i \in [1, m],\ j \in [1, o]$$

  - Implement and test

Using the Debugger

Main objective: **Inspect**

- Breakpoint
- Stepping
- Variables inspection

Day 2: Object Oriented Programing

Before Object...

FIFO

Without using objects, implement a FIFO with the following properties:

- `int` elements
- pop function that return **and** remove the first element of the FIFO
- `push` function that add an element to the back (i.e. as last element) of the FIFO

Test the FIFO:

- Create a FIFO
- Add elements {`1,..,1000`} to the FIFO
- Remove the first 100 elements and check content

Modularity & Encapsulation

Modularity & Encapsulation

- Decompose a system into simpler subsystems to reduce overall complexity
- Module: subsystem **loosely** coupled to other subsystems.
- Encapsulation
  - technique used to favor subsystems' modularity
  - separate *interface* and *implementation*
  - data protection: data are not accessed directly, modules communicates via messages (methods)
  - responsibilities

Object

Object

An object is a module

An entity with:

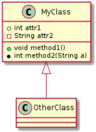
- data (state)
- functions/procedures (behaviour)

Class

Class

Type/structure (of an object) with:

- attributes
- methods



Visibility: public, protected, private

Constructor: initializes an object (instance)

Destructor: `finalize()`. cannot be called directly in `Java`

Class attributes/methods

Inheritance

Inheritance

A child class is a **specialization** of parent class.

- Access to parent members
- *enhancement* through new attributes and overrides
- Constructors
- Prevent class specialization with `final` keyword

Polymorphism

Polymorphism

- An instance belongs to its direct defining class and all its parent class

Abstract class and interface

- An *abstract class* is a class with (at least) one abstract (unimplemented) method.
- An *interface* lists a collection of *useable* methods (i.e contracts). A class **implements** an interface if it defines **all** the methods listed in the interface definition.

Object in a nutshell

Objects are used to **model** a problem with **concepts** relevant to what is to solve.

(Some) Golden rules:

- Analyse the problem to solve.
  - Understand what is at stake
  - Understand the domain
- Model the system with **relevant** concepts
  - e.g. Don't consider element of the *real world* irrelevant to the problem.
- Try to anticipate future changes. . .

Excercices

My first models with objects

- Cars
- Application handling Courses

Excercices

My first models with objects

- Cars
- Application handling Courses
- **HearthStone**

FIFO w/ objects

- Implement a FIFO using objects.
- **Test** your implementation.

Collections using interfaces

We want to be able to use FIFO, LIFO, LILO, ...

- Define an interface suiting the need above.
- Implement a FIFO, LIFO using implementing interface defined above.
- Test.

UML

Unified Modeling **Language**

13 diagrams (**structure** & **behavior**)

Class diagram

- Analysis and design of the static view of an application
- Describe responsibilities of a system
- The only diagram that maps directly mith oo languages
- Highlights
    - Meaningful class name
    - Relationships (e.g specialization, assoication, multiplicity. . .)
    - Favor clarity and keep only useful properties
    - Use notes when needed

Sequence diagram

- Used to visualize the sequence of calls in a system
- Highlights
  - Used to described the *workflow* of a complex functionality

State machine diagram

- To model the dynamic aspect of a system.
- To model the life time of a reactive system.
- To describe different states of an object during its life time.
- Define a state machine to model the states of an object.

Use case diagram

- Used to gather the requirements of a system.
- Used to get an outside view of a system.
- Identify the external and internal factors influencing the system.
- Show the interaction among the requirements and actors.

Case study: Bank agency

An bank agency needs an application to manage its customers' bank accounts.

- Define/precise the needs using a use case diagram.
- Model the application using a class diagram.
- Implement and test.

Day 3: Reminders (cont.)

Generics

Why

- Factor/Re-use code with different input
- Benefit from type checking

Usage

- Declaring a generic type

```
class name<T1, T2, ..., Tn> { /* ... */ }
}
```

- Invoking and Instantiating a Generic Type

```
List<Integer> integerList;
```

Generic Methods

```java
public class Util {
    public static <K, V> boolean compare(K p1, V p2) {
        // do compare
    }
}
```

Bounded Type Parameters

```java
// A, B and C can be classes or interfaces
public class<U extends A & B & C> {
}
```

Example

Container of updatable elements

Threads and concurrency

Using threads

Threads and concurrency

Using threads

- Sub-classing `Thread`

Threads and concurrency

Using threads

- Sub-classing `Thread`
- Implementing `Runnable`

Concurrency

Concurrency

- `synchronized` methods

Exercice: FIFOs

Concurrency

- `synchronized` methods
- `synchronized` blocks

Exercice: FIFOs

Concurrency

- `synchronized` methods
- `synchronized` blocks
- `wait()` and `notifyAll()`

Exercice: FIFOs

Swing

Introduction

Swing

Introduction

- Toolkits

Swing

Introduction

- Toolkits
- Base components (e.g. widgets, container, layouts)

Swing

Introduction

- Toolkits
- Base components (e.g. widgets, container, layouts)
- Event handlers

Case study: Calculator

Day 4: Exam1

Day 5: OOP (cont.)

The facts

- Requirements are **bound** to change and they **will**.
  - Taking into account **unforeseen** changes may be costly.
  - Software architecture and design must *strive* to **anticipate** changes
- *Design object-oriented software is hard.*
  - OOP is a *philosophy/framework*: no methods, some guidelines.
  - *Spaghetti* effect.
  - Complexity grows with size. Maintainability decreases with complexity.

An answer

Design patterns aggregate all the experience from developpers who have been working on a **common** problem and offer a conceptual design solution.

Design patterns

Types

- Creational
- Structural
- Behavioral

Creational Patterns

Simple Factory

Creational Patterns

Simple Factory

Factory Method

Creational Patterns

Simple Factory

Factory Method

Singleton

Structural Patterns

Adapter

Structural Patterns

Adapter

Bridge

Structural Patterns

Adapter

Bridge

Facade

Behavioral Patterns

Observer

Behavioral Patterns

Observer

Visitor

Exercices

Calculator

Exercices

Calculator

PhoneBook

Exercices

Calculator

PhoneBook

PhoneBook (yet again)

Exercices

Calculator

PhoneBook

PhoneBook (yet again)

Widget: Multi-view