JUNIA Grande école d'ingénieurs
HEI·ISEN·ISA

Part 5

OOP in practice

Exam question samples

- For which reason files with .class extension have been created?
- List wrapper classes and explain their use.
- What are the differences between using =, == and calling the *equals ( )* method?
- Give 2 advantages of using a variable of Integer data type instead of an int ?
- Explain the JDK, JRE and JVM.
- Explain the meaning of each word in the method definition
  public static void main (String args [])
- According to you, is Java 100% Object-Oriented ? Explain your answer.

Command line arguments

- Command line arguments are defined as a parameter for the main method

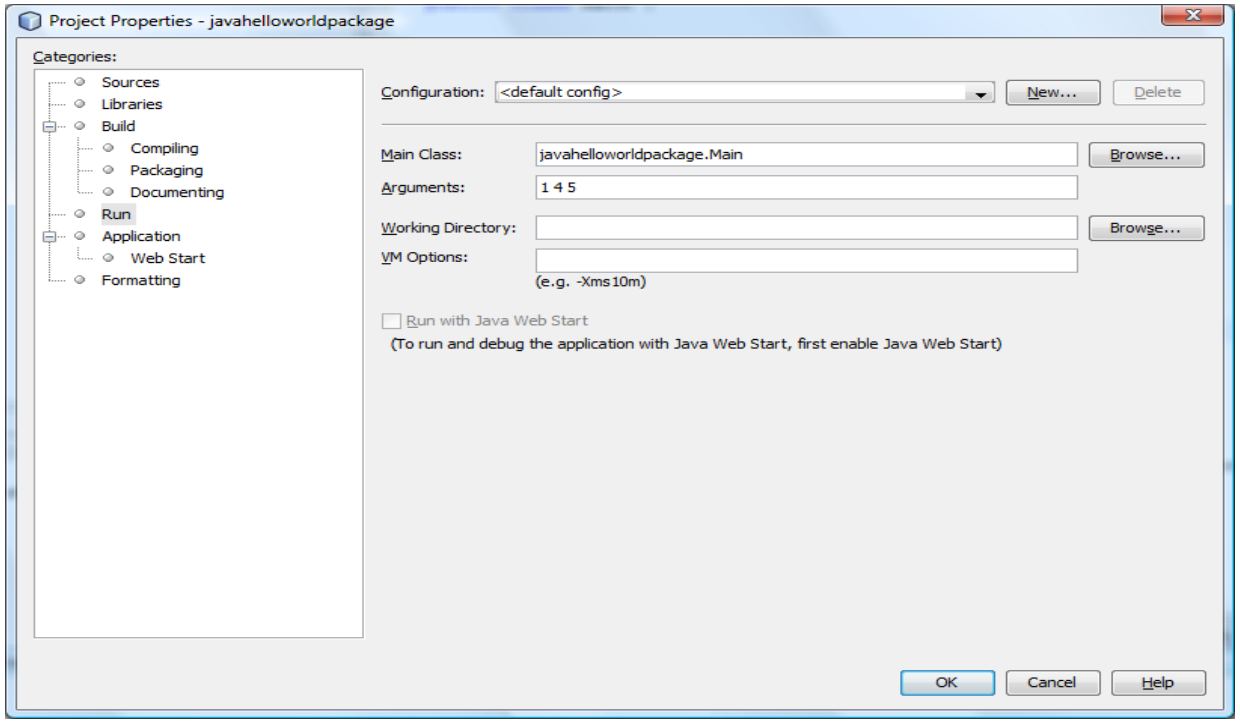
- To use arguments:
- int args.length: Returns the number of arguments as an integer
- String args[index]: Returns the arguments number “index” as an instance of class String

- The index of first argument is always equal to 0

```
8   public class HelloWorld {
9
10      /**
11       * @param args the command line arguments
12       */
13      public static void main(String[] args) {
14          // TODO code application logic here
15          System.out.println("Hello World!");
16      }
17  }
```

Command line arguments

- Entering command line arguments with NetBeans:
- Choose the option “Run” in the menu
- Then select “Set Project Configuration” and “Customize”
- Then enter your arguments (separated by a space) in the “Arguments” text box in the “Run” category.
- Click “OK” to validate



Input and Output

- Class Scanner allows to manipulate easily the input stream

- A few methods for the class Scanner:
- boolean hasNext ( ): Returns true if the calling scanner instance has another token in its input
- boolean hasNextInt ( ): Returns true if the next token in the calling scanner instance input can be interpreted as an int value
- String next ( ): Finds and returns the next complete token from the calling scanner instance
- int nextInt ( ): Scans the next token of the calling scanner instance as an int
- String nextLine ( ): Advances the calling scanner instance past the current line and returns the input that was skipped

- Could not be used if you does not include the package java.util.Scanner in your program

Input and Output

```java
package javaapplication1;
import java.util.Scanner;

public class ScannerTest {

    public static void main(String[] args) {

        Scanner keyboard = new Scanner (System.in);
        String name = "";

        System.out.println("Enter your name");
        System.out.print("--> ");

        name = keyboard.nextLine();
        System.out.println("\nHello " + name);
    }
}
```

```
run:
Enter your name
--> Albert

Hello Albert
BUILD SUCCESSFUL (total time: 6 seconds)
```

Input and Output

- With Scanner multiple inputs must be separated by whitespace and read by multiple invocations of the appropriate method (example : nextInt())
- nextLine reads the remainder of a line of text starting wherever the last keyboard reading left off
- This can cause problems when combining it with different methods for reading from the keyboard such as nextInt
- Need of import java.util.Scanner;

Input and Output

Given the code :

```
Scanner keyboard = new Scanner (System.in);
int number = keyboard.nextInt();
String string1 = keyboard.nextLine();
String string2 = keyboard.nextLine();
```

and the input,

```
2
Heads are better than
1 head.
```

Then :

```
number = 2
string1 =
string2 = Heads are better than
```

If the following results were desired instead
number equal to "2", string1 equal to "heads are better than", and string2 equal to "1 head."

then an extra invocation of nextLine would be needed to get rid of the end of line character ('\n')

JUNIA Grande école d'ingénieurs
HEI·ISEN·ISA

A word about Polymorphism

static polymorphism

Overloading : **compile-time polymorphism**

```
public static void main(String[] args) {

    System.out.println("Welcome to first course survivors");

    System.out.println('c');

    System.out.println(4.2f);

}
```

void java.io.PrintStream.println(String x)

void java.io.PrintStream.println(char x)

void java.io.PrintStream.println(float x)

static polymorphism

```java
public static void methodName (int i,long l){
    System.out.println("methodName(int, long)");
}
public static void methodName (long i,int l){
    System.out.println("methodName(long, int)");
}


public static void main(String[] args) {

    int i = 1;
    long l = 1L;
    methodName(i,l);
    methodName(l, i);


}
```

static polymorphism

```
public static void methodName (int i,long l){
    System.out.println("methodName(int, long)");
}
public static void methodName (long i,int l){
    System.out.println("methodName(long, int)");
}


public static void main(String[] args) {

    int i = 1;
    long l = 1L;
    methodName(i,l);
    methodName(l, i);
    methodName(i, i);

}
```

static polymorphism

Overloading ambiguity :

The compiler can not know which one to choose

```
public static void methodName (int i,long l){
    System.out.println("methodName(int, long)");
}
public static void methodName (long i,int l){
    System.out.println("methodName(long, int)");
}

public static void main(String[] args) {

    int i = 1
    long l =
    methodNam
    methodNam
    methodName(i, i);

}
```

reference to methodName is ambiguous
both method methodName(int,long) in HelloWorld and method methodName(long,int) in HelloWorld match
---
(Alt-Enter shows hints)

PART 6

Using classes in Java (Part 1)

JUNIA Grande école d'ingénieurs
HEI·ISEN·ISA

What is a Class : definition

- Java is an **Object Oriented Programming** language
- **Classes** are the **most important** language **features**
- A Java program: instances from **various classes** interacting with one another
- You have already used classes (String and Scanner for example)
- In Java: the name of a class should **always** begin with an **uppercase**
- In Java: you could have only **one class per file** and the name of the file should be equal to the name of the class.

What is a Class : definition

- A Class is a **dataType**
- You can declare variables of a **class type**
- A value of a class type is an **instance (or an object)**
- An instance has both **data** and **actions**
- Actions are called **methods**

Understanding Class vs object

Each instance of a class have the same types of data and the same method

```
public Car myIdealCar = new Car( ) ;
```

"*myIdealCar* is an instance of Car",
"*myIdealCar* is of type Car" and
"*myIdealCar* is an object of the class Car"
mean the same thing

```
public Car myFirstCar = new Car( )
```

Each instance of a class could have different data

Each instance of a class have the same actions

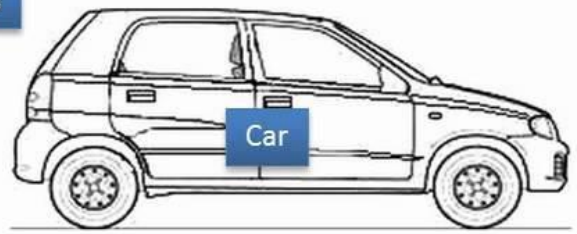
Understanding Class vs object

Classes,
Fields or attributes,
methods,
constructors,
are the building blocks of object-based Java applications.

What is a Class?

A *class* is the blueprint from which individual objects are created.

Class




```
public class Car
{
    private String brand = null;
    private String model = null;
    private String color = null;

    public String getBrand()
    {
        return brand;
    }

    public void setBrand(String brand)
    {
        this.brand = brand;
    }

    public String getModel()
    {
        return model;
    }

    public void setModel(String model)
    {
        this.model = model;
    }

    public String getColor()
    {
        return color;
    }

    public void setColor(String color)
    {
        this.color = color;
    }
}
```

Objects

```
Car maruthiAltok10 = new Car();

maruthiAltok10.setBrand("Maruthi Alto");
maruthiAltok10.setModel("k10");
maruthiAltok10.setColor("Orange");
```

brand = Maruthi Alto
model = K10
color = Orange

```
Car swift = new Car();

swift.setBrand("Swift");
swift.setModel("ZDI");
swift.setColor("Red");
```

brand = Swift
model = ZDI
color = Red

```
Car maruthiAlto800 = new Car();

maruthiAlto800.setBrand("Maruthi Alto");
maruthiAlto800.setModel("800");
maruthiAlto800.setColor("Blue");
```

brand = Maruthi Alto
model = 800
color = Blue

Example

Example of a class definition:

```
package simplePackage;

public class SimpleDate {
    public int day;
    public int month;
    public int year;

    public void writeSimpleDate(){
        System.out.println(month + "/" + day + "/" + year);
    }
}
```

- To create an instance of a class, you must use the **new** operator
- Syntax:
  Class_Name class_Variable = new Class_Name();
- Example:
  SimpleDate date1 = new SimpleDate();

Example


```java
package simpleDatePackage;

import simpleDate.SimpleDate;

public class SimpleDatedemo {

    public static void main(String[] args) {
        SimpleDate date1 = new SimpleDate();

        date1.day = 20;
        date1.month = 9;
        date1.year = 2023;

        date1.writeSimpleDate();
    }
}
```

ifications | Search Results | Output - Course2 (run) ×

```
run:
9/20/2023
BUILD SUCCESSFUL (total time: 0 seconds)
```

Class content

```
[visibility] [final] class NameOfTheClass {

        [visibility] [static] [final] dataType variable1;
        [visibility] [static] [final] dataType variable2;
        ....
        public NameOfTheClass() { .... }
        [visibility] NameOfTheClass( parameterList) { .... }
        [visibility] NameOfTheClass( differentParameterList) { .... }
        ....
        [visibility] [static] [final] dataTypeReturned nameOfMethod1 (parameterList){ .... }
        [visibility] [static] [final] dataTypeReturned nameOfMethod2 (parameterList){ .... }
        ....
}
```

Encapsulation

Encapsulation is a way to protect your program and your data.

There exists 4 different **[visibility]** status :

- **public**
- **friendly (no modifier)**
- **protected**
- **private**

Classes are almost always public

methods are generally

public : if giving access to functionnalities (e.g. accessors)

private or protected when dealing with inside workout of the class

Variables are private or protected (unless it is a constant)

Encapsulation

	Same Class Same Package	Other Class Same Package	SubClass Other Package	Other Class Other Package
public	✓	✓	✓	✓
protected	✓	✓	✓	
default	✓	✓		
private	✓			

Encapsulation

- To access instance variables, it's recommended to use accessor and mutator methods

- **Accessor** methods allow to obtain the value of data in a class instance
  General syntax for an accessor method:
  ```
  public dataType getVariableName( )
  ```

- **Mutator** methods allow to change data in a class instance
  General syntax for a mutator method:
  ```
  public void setVariableName(dataType variableName)
  ```

Encapsulation

- To access instance variables, it's recommended to use accessor and mutator methods
- **Accessor** methods allow to retrieve the value of data in a class instance
- **Mutator** methods allow to change data in a class instance

```java
public class PlayerExample {
    private String name = "Albert";
    private int age = 18;
    private double money = 45.23;

    /**
     * Accessor of the age variable
     * @return the age of the person
     */
    public int getAge(){
        return this.age;
    }
    /**
     * Accessor of the name variable
     * @return the name of the person
     */
    public String getName(){
        return this.name;
    }
    /**
     * Accessor of the money variable
     * @return the amount of money the player has
     */
    public double getMoney(){
        return this.money;
    }
```

```java
    /**
     * Mutator of the age variable
     * @param age the new age value has to be between 0 and 144 years old.
     */
    public void setAge(int age){
        if(age>0 && age<114)
            this.age = age;
        else
            System.out.println("The given age value is incorrect");
    }
    /**
     * Mutator of the name variable
     * @param name the nes player name
     */
    public void setName(String name) {
        if(name != null && (name.length()>4))
            this.name = name;
        else
            System.out.println("The given name value is incorrect");
    }
}
```

Class content

```
[visibility] [final] class NameOfTheClass {

        [visibility] [static] [final] dataType variable1;
        [visibility] [static] [final] dataType variable2;
        ....
        public NameOfTheClass() { .... }
        [visibility] NameOfTheClass( parameterList) { .... }
        [visibility] NameOfTheClass( differentParameterList) { .... }
        ....
        [visibility] [static] [final] dataTypeReturned nameOfMethod1 (parameterList){ .... }
        [visibility] [static] [final] dataTypeReturned nameOfMethod2 (parameterList){ .... }
        ....
}
```

static keyword

[static] **means « Something that belongs to the class »**

Not to a single objects/instance

[static] **means « Shared and accessible »**

[static] can be used with :

- methods (instance methods ≠ class methods)

static method (class method)

Syntax:

public static ReturnDatatype methodName(Parameter_List)

- A static method is a method that can be used without a calling instance
- To do that, we should use the class name in place of a calling instance
- A static method could never refer to an instance variable

- Example of a definition of a static method in class SimpleDate:

```
public static boolean isDateOk(int day, int month) {
    return (((day>0)&&(day<31))&&((month>0)&&(month<13)));
}
```

- Example of use in the main method :

```
if(SimpleDate.isDateOk(day,month))
    date1 = new SimpleDate(month,day,year);
else
    date1 = new SimpleDate(1,1,2019);
```

static method (class method)

```
public static void main(String[] args) {
    SimpleDate date1 = new SimpleDate();

    date1.day = 20;
    date1.month = 9;
    date1.year = 2021;

    if (isDateOK(date1.day, date1.month))
        date1.writeSimpleDate();
}
```

non-static method isDateOK(int,int) cannot be referenced from a static context

incompatible types: void cannot be converted to boolean

(Alt-Enter shows hints)

static method (class method)

```
public static void main(String[] args) {

    String string1 = "Something to say.";
    String string2 = "18092017";
    double valueOfString2=0;
    double result=0;

    System.out.println(string1);

    valueOfString2 = Double.parseDouble(string2);

    result = Math.sqrt(valueOfString2);


}
```

static method (class method)

- A static method cannot refer to an instance variable of the class, and it cannot invoke a non-static method of the class

- A static method has no **this**, so it cannot use an instance variable or method that has an implicit or explicit **this** for a calling object

- A static method can invoke another static method, however

static keyword

[static] means « Something that belongs to the class »

Not to a single objects/instance

[static] means « Shared and accessible »

[static] can be used with :

- **methods (instance methods ≠ class methods)**
- **variables (instance variables ≠ class variables)**

static variable (class variable)

- Syntax of a static variable:
  private static **dataType variableName ;**
- A static variable is a variable that belongs to the class and not only to one instance *(same value for all instances)*
- A static variable is generally used to allow communication between instances
- A defined constant is a static variable which value could not be changed
- Syntax of a defined constant:
  public static final **dataType variableName = value;**

static variable (class variable)

Example of a definition and use of a static variables in simple counter of registered players

```java
public class PlayerExample {

    public static int counter = 0;

    private String name = "Albert";
    private int age = 18;
    private double money = 45.23;

    public PlayerExample(){
        this.counter++;
    }
}
```

```java
public class PlayerExample {

    private static int counter = 0;

    private String name = "Albert";
    private int age = 18;
    private double money = 45.23;

    public PlayerExample(){
        this.counter++;
    }
    /**
     * Accessor of the counter variable
     * @return the number of objects created since the beginning of the program
     */
    public int getNumberOfObjectsCreated(){
        return this.counter;
    }
}
```

static keyword

[static] **means « Something that belongs to the class »**

Not to a single objects/instance

[static] **means « Shared and accessible »**

[static] can be used with :

- **methods (instance methods ≠ class methods)**
- **variables (instance variables ≠ class variables)**
- **bloc of code (max 1 per class)**
- **classes**

static initializing code

- Only situation where you can write code outside from method.

```
public class StaticDemo {

    public static int numberOfInstances;

    public static char[] alphabet = new char[26];

    static {

        numberOfInstances=0;

        int index=0;
        for (char i='a';i<='z';i++)
            alphabet[index++] = i;

    }
}
```

- Not possible to use instance variable inside

JUNIA Grande école d'ingénieurs
HEI·ISEN·ISA

PART 7

Using classes in Java (Part 2)

instance methods

- Keyword **this** can be used as a name for the calling instance
- Useful if you want to have the parameters equal to the instance variables
- Example with our SimpleDate class:

```
public void setDate(int day, int month, int year) {
    this.day = day;
    this.month = month;
    this.year = year;
}
```

this gives you access to :

variables (instance, class)
methods (instance, class, overloaded, …)
inherited things

other methods

- Java expects certain methods to be in almost all classes
- Why? Because Java libraries have software that assumes such methods are defined
- Method **equals** returns a boolean which is true when the two instances compared are equals **(data inside the objects ≠ references)**
- Syntax:
  public boolean equals(Class_Name Parameter_Name)
- Method **toString** return a String containing the data in the instance
- Syntax:
  public String toString()

```
@Override
public boolean equals(Object obj) {
    if (this == obj) {
        return true;
    }
    if (obj == null) {
        return false;
    }
    if (getClass() != obj.getClass()) {
        return false;
    }
    PlayerExample other = (PlayerExample) obj;
    if (this.age != other.age) {
        return false;
    }
    if (!this.name.equals(other.name)) {
        return false;
    }
    return true;
}
```

JUNIA Grande école d'ingénieurs
HEI·ISEN·ISA

```java
package simplePackage;
/**
 * @version 1.3
 * @author mikael.morelle
 */
public class SimpleDate {
    private int day = 16;
    private int month = 9;
    private int year = 2019;

    /**
     * Accessor of the day field
     * @return the value of the day
     */
    public int getDay() {
        return day;
    }

    /**
     * Accessor of the month field
     * @return the value of the month
     */
    public int getMonth() {
        return month;
    }

    /**
     * Accessor of the year field
     * @return the value of the year
     */
    public int getYear() {
        return year;
    }

    /**
     * Mutator of the day field
     * @param day the new value
     */
    public void setDay(int day) {
        int [] daysPerMonth = {31,28,31,30,31,30,31,31,30,31,30,31};
        if((day>=1) && (day <= daysPerMonth[this.month-1]))
            this.day = day;
    }

    /**
     * Mutator of the month field
     * @param month the new value
     */
    public void setMonth(int month) {
        if((month >= 1) && (month <= 12))
        this.month = month;
    }

    /**
     * Mutator of the year field
     * @param year the new value
     */
    public void setYear(int year) {
        this.year = year;
    }


    public boolean equals(SimpleDate other) {

        if (this.day != other.day) {
            return false;
        }
        if (this.month != other.month) {
            return false;
        }
        if (this.year != other.year) {
            return false;
        }

        return true;
    }
}
```

```
    public void printDate(){
        System.out.println(this);
    }

    public static void main(String ... args){

        SimpleDate date = new SimpleDate();

        date.printDate();
    }
}
```

simplePackage.SimpleDate

ut - SimpleDate (run)

```
run:
simplePackage.SimpleDate@15db9742
```

The system prints the object reference

The system prints the String returned by toString()

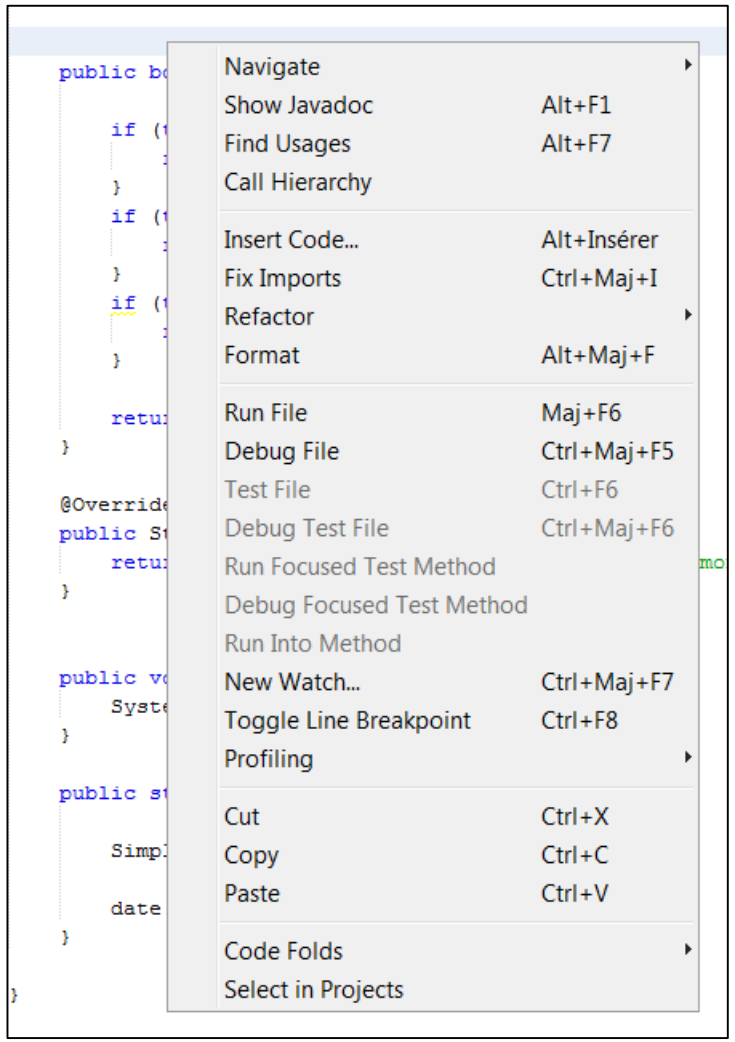
```
    public void printDate(){
        System.out.println(this);
    }

    @Override
    public String toString() {
        return "The date is : " + month + "/" + day + "/" + year;
    }

    public static void main(String ... args){

        SimpleDate date = new SimpleDate();

        date.printDate();
    }
}
```

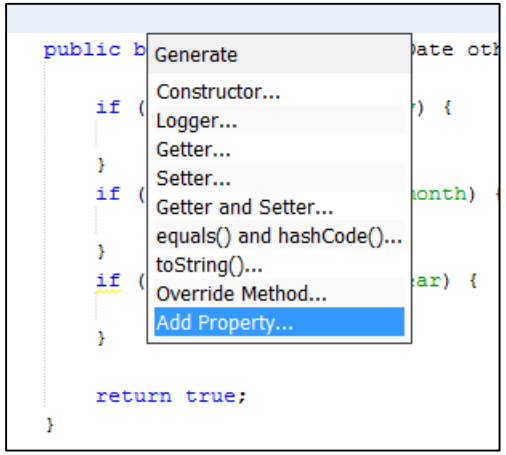
simplePackage.SimpleDate

ut - SimpleDate (run)

```
run:
The date is : 9/16/2019
BUILD SUCCESSFUL (total time: 0 seconds)
```

Your IDE can help you writing code :





However, you need to modify and adapt the generated code to your needs.

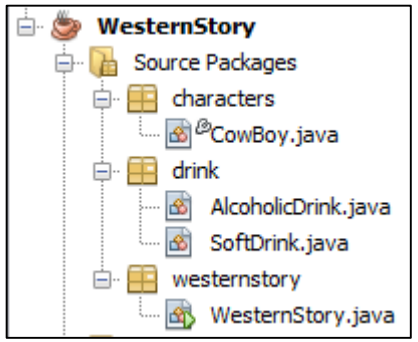
Constructor

- A constructor is a specific method that is designed to perform the initialization of an instance
- A constructor has always the **same name as the class name**
- A constructor does not return any data type
- You could use **overloading** on constructor
- Each class always has **an implicit constructor** with no parameters (initialization of all the instance variables to their default value) when no constructor is implemented.
- To use a new object, you must call a constructor with the instruction **new**

```
SimpleDate currentDate;
currentDate = new SimpleDate();
```

- You **can't use a constructor to reinitialize** the value of an already initialized instance (use mutators instead)

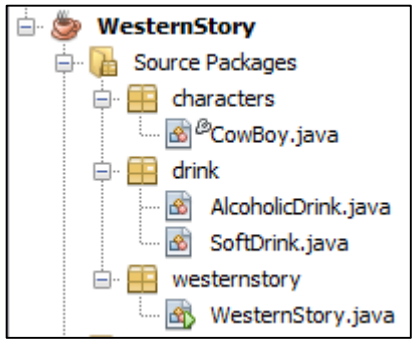
```java
package characters;
import drink.*;

public class CowBoy {
    private String name;
    private String adjective;
    private SoftDrink favouriteDrink;
    private int numberOfRobberArrested;

}
```




```java
package westernstory;
import characters.*;

public class WesternStory {

    public static void main(String[] args) {

        CowBoy luke = new CowBoy();
    }
}
```

Implicit constructor

```
compile:
run:
BUILD SUCCESSFUL (total time: 1 second)
```




```java
package characters;
import drink.*;

public class CowBoy {
    private String name;
    private String adjective;
    private SoftDrink favouriteDrink;
    private int numberOfRobberArrested;


    @Override
    public String toString() {
        return "CowBoy{" + "name=" + name + ", adjective=" + adjective +
                ", favouriteDrink=" + favouriteDrink + ", numberOfRobberArrested="
                + numberOfRobberArrested + '}';
    }
}
```

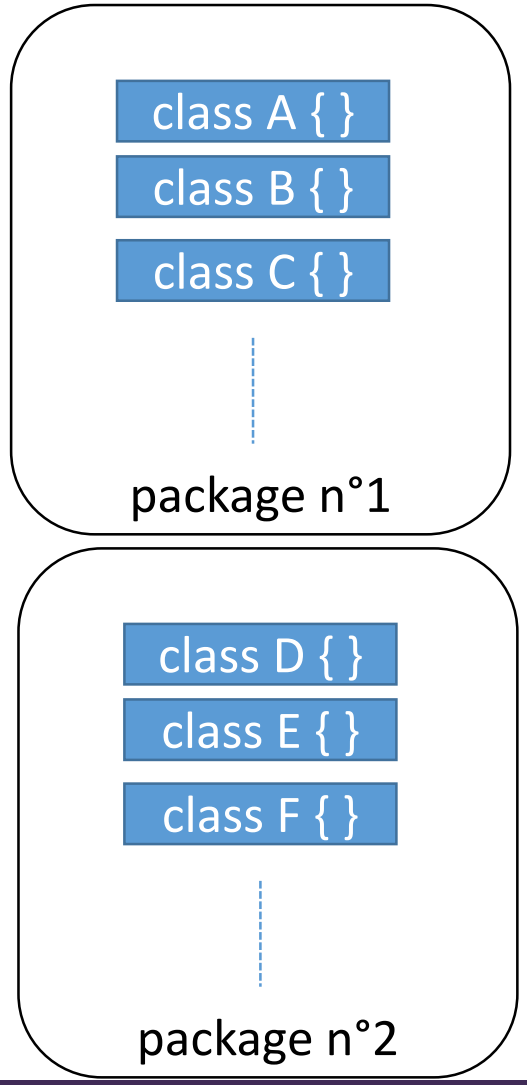
```java
package westernstory;
import characters.*;

public class WesternStory {

    public static void main(String[] args) {

        CowBoy luke = new CowBoy();
        System.out.println(luke);
    }
}
```

Implicit constructor

instance variables initialized to 0 or null

```
run:
CowBoy{name=null, adjective=null, favouriteDrink=null, numberOfRobberArrested=0}
BUILD SUCCESSFUL (total time: 1 second)
```

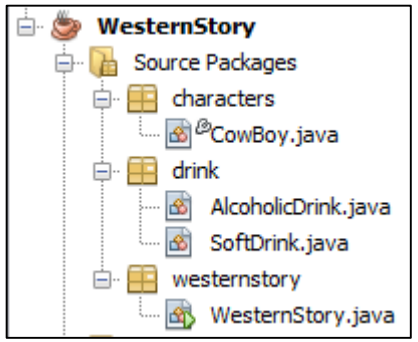
Program structure



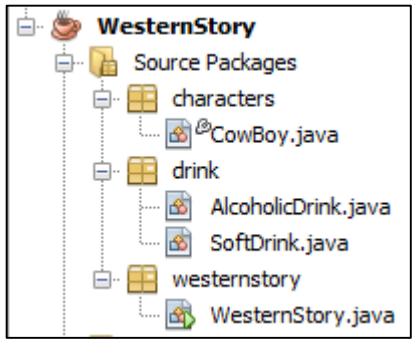
Unlike C programs, Object Oriented Programming allows to :

- Easily represents complex things of real life
- Structure things correctly
- Manage rights (visibility, modifications, …)

```
import package 1;
import package 2;
import others.libraries;


class Program {
// mainMethod (){}
}
```




```java
package drink;

public class SoftDrink {
    private String name;
    private int size;
    private float price;

    public static final SoftDrink WATER = new SoftDrink("Water", 25, 0.0f);
    public static final SoftDrink TEA = new SoftDrink("Tea", 25, 2.50f);

    public SoftDrink(String name, int size, float price){
        this.name = name;
        this.size = size;
        this.price = price;
    }

    public String getName(){
        return this.name;
    }
    public int getsize(){
        return this.size;
    }
    public float getPrice(){
        return this.price;
    }
    protected void setPrice(float price){
        if(price>0)
            this.price = price;
        else
            System.out.println("The price value is incorrect");
    }
}
```




```java
package drink;

public class AlcoholicDrink {
    private String name;
    private int size;
    private float price;
    public int alcoholLevel;

    public static final AlcoholicDrink BEER = new AlcoholicDrink("Beer", 25, 3.0f, 6);

    public AlcoholicDrink(String name, int size, float price, int alcoholLevel){
        this.name = name;
        this.size = size;
        this.price = price;
        this.alcoholLevel = alcoholLevel;
    }

    public String getName(){
        return this.name;
    }
    public int getsize(){
        return this.size;
    }
    public float getPrice(){
        return this.price;
    }
    public int getAlcoholLevel(){
        return this.alcoholLevel;
    }
    protected void setPrice(float price){
        if(price>0)
            this.price = price;
        else
            System.out.println("The price value is incorrect");
    }
}
```

JUNIA Grande école d'ingénieurs
HEI·ISEN·ISA

```java
package characters;
import drink.*;

public class CowBoy {
    private String name;
    private String adjective;
    private SoftDrink favouriteDrink;
    private int numberOfRobberArrested;

    public CowBoy(String name, String adjective,SoftDrink favouriteDrink ){
        this(name,adjective);
        this.favouriteDrink = favouriteDrink;
    }

    public CowBoy(String name, String adjective){
        this.name = name;
        this.adjective = adjective;

        this.favouriteDrink = SoftDrink.WATER;
        introduceYourself();
    }

    public void introduceYourself(){
        talk("Hello, I am the newest citizen.");
        talk("People call me " + this.name + " the " + this.adjective);
        talk("Meet me at the Saloon, I am sure you want to buy " + this.favouriteDrink);
    }

    public void talk(String say){
        System.out.println(say);
    }
}
```

```java
package westernstory;
import characters.*;

public class WesternStory {

    public static void main(String

        CowBoy luke = new CowBoy();
    }
}
```

```
no suitable constructor found for CowBoy(no arguments)
    constructor CowBoy.CowBoy(String,String,SoftDrink) is not applicable
      (actual and formal argument lists differ in length)
    constructor CowBoy.CowBoy(String,String) is not applicable
      (actual and formal argument lists differ in length)
----
(Alt-Enter shows hints)
```

```java
package westernstory;
import characters.*;

public class WesternStory {

    public static void main(String[] args) {

        CowBoy luke = new CowBoy("Lucky Luke", "Brave");
    }
}
```

```
run:
Hello, I am the newest citizen.
People call me Lucky Luke the Brave
Meet me at the Saloon, I am sure you want to buy me a good Water
BUILD SUCCESSFUL (total time: 1 second)
```

```java
public class CowBoy {
    private String name;
    private String adjective;
    private SoftDrink favouriteDrink;
    private int numberOfRobberArrested;

    private SimpleDate birthDate;

    public SimpleDate getBirthDate(){
        return this.birthDate;
    }

    public CowBoy(String name, String adjective,SimpleDate birthDate ){
        this(name,adjective);
        this.birthDate = birthDate;
    }
```

```java
package westernstory;

public class SimpleDate {
    public int month, day, year;

    public SimpleDate(int month, int day, int year){
        this.month = month;
        this.day = day;
        this.year = year;
    }

    @Override
    public String toString() {
        return this.month + "/" + this.day + "/" + this.year;
    }
}
```

```java
public static void main(String[] args) {

    CowBoy luke = new CowBoy("Lucky Luke", "Brave", new SimpleDate(3,22,2022));
    luke.talk(luke.getBirthDate().toString());
}
```

```
run:
Hello, I am the newest citizen.
People call me Lucky Luke the Brave
Meet me at the Saloon, I am sure you want to buy Water
3/22/2022
BUILD SUCCESSFUL (total time: 0 seconds)
```

```java
public class CowBoy {
    private String name;
    private String adjective;
    private SoftDrink favouriteDrink;
    private int numberOfRobberArrested;

    private SimpleDate birthDate;

    public SimpleDate getBirthDate(){
        return this.birthDate;
    }

    public CowBoy(String name, String adjective,SimpleDate birthDate ){
        this(name,adjective);
        this.birthDate = birthDate;
    }
```

```java
package westernstory;

public class SimpleDate {
    public int month, day, year;

    public SimpleDate(int month, int day, int year){
        this.month = month;
        this.day = day;
        this.year = year;
    }

    @Override
    public String toString() {
        return this.month + "/" + this.day + "/" + this.year;
    }
}
```

```java
public static void main(String[] args) {

    CowBoy luke = new CowBoy("Lucky Luke", "Brave", new SimpleDate(3,22,2022))

    luke.getBirthDate().month = 1;
    luke.getBirthDate().day = 1;
    luke.getBirthDate().year = 666;

    luke.talk(luke.getBirthDate().toString());
}
```

```
run:
Hello, I am the newest citizen.
People call me Lucky Luke the Brave
Meet me at the Saloon, I am sure you want to buy Water
1/1/666
BUILD SUCCESSFUL (total time: 0 seconds)
```

Privacy leak !!!

Copy Constructor

- A copy constructor is a constructor with a single argument of the same type as the class
- The copy constructor should create an object that is a separate, independent object, but with the instance variables set so that it is an exact copy of the argument object

```java
public class CowBoy {
    private String name;
    private String adjective;
    private SoftDrink favouriteDrink;
    private int numberOfRobberArrested;

    private SimpleDate birthDate;

    public SimpleDate getBirthDate(){
        return new SimpleDate(this.birthDate);
    }

    public CowBoy(String name, String adjective,SimpleDate birthDate ){
        this(name,adjective);
        this.birthDate = birthDate;
    }
```

```java
package westernstory;

public class SimpleDate {
    public int month, day, year;

    public SimpleDate(int month, int day, int year){
        this.month = month;
        this.day = day;
        this.year = year;
    }

    public SimpleDate(SimpleDate other){
        this.month = other.month;
        this.day = other.day;
        this.year = other.year;
    }
    @Override
    public String toString() {
        return this.month + "/" + this.day + "/" + this.year;
    }
}
```

```java
public static void main(String[] args) {

    CowBoy luke = new CowBoy("Lucky Luke", "Brave", new SimpleDate(3,22,2022));

    luke.getBirthDate().month = 1;
    luke.getBirthDate().day = 1;
    luke.getBirthDate().year = 666;

    luke.talk(luke.getBirthDate().toString());
}
```

```
run:
Hello, I am the newest citizen.
People call me Lucky Luke the Brave
Meet me at the Saloon, I am sure you want to buy Water
3/22/2022
BUILD SUCCESSFUL (total time: 1 second)
```

Data are secured !

Making objects interact

```java
public void putInJail(Robber robber){
    this.numberOfRobberArrested++;
    robber.putInJail(this);

    talk("Game over " + robber.whoAreYou() + ", You will never see the light of the sun anymore.");
}

public String whoAreYou(){
    return this.name + " the " + this.adjective;
}
```

In the CowBoy.java file

```java
public void putInJail(CowBoy cowboy){
    talk ("Oh no, " + cowboy.whoAreYou() + " surprised me");
    talk ("That's not fair, i will get me revenge");

    this.rewardValue =0;
}

public String whoAreYou(){
    return this.name + " the " + this.look;
}
```

In the Robber.java file

```
run:
Hello, I am the newest citizen.
People call me  Lucky Luke the Brave
Meet me at the Saloon, I am sure you want to buy Water

Hello, I am the newest citizen !!!
I am Joe Dalton the Wicked don't bother me !!!

Oh no, Lucky Luke the Brave surprised me !!!
That's not fair, i will get me revenge !!!
Game over Joe Dalton the Wicked, You will never see the light of the sun anymore.
BUILD SUCCESSFUL (total time: 1 second)
```

```java
public static void main(String[] args){
    CowBoy luke = new CowBoy("Lucky Luke", "Brave", new SimpleDate(3,22,2022));
    Robber joe = new Robber ("Joe Dalton", "Wicked");

    luke.putInJail(joe);
}
```

Destructor in Java

- A destructor allows to make a cleanup of memory after an instance is no longer used
- In C++ destructor should be used
- In Java there is **no explicit "destructor"**
- Java uses the **garbage collector** instead which is part of the runtime
- An instance is put inside the garbage collector when there is no more reference to it inside the program
- The garbage collector is a **low priority thread** for cleaning memory
- With the garbage collector you will never know when the memory will be cleaned

Destructor in Java

- The method finalize allows you to define statements that will be executed before the instance is deleted from the memory by the garbage collector
- Syntax:

```
@Override
protected void finalize() throws Throwable {
    // Coding instructions to execute before deleting an object
}
```

Destructor in Java

```java
public class Counter {

    public static int numberOfInstances = 0;

    public Counter(int i){
        numberOfInstances++;
        print(i);
    }

    public void print(int i){
        System.out.println("i value is :" + i + "\tnumberOfInstances is :" + numberOfInstances);
    }

    @Override
    protected void finalize()  {
        numberOfInstances--;
    }

    public static void main (String ... args){
        Counter testCounter;
        for(int i =0;i<1000000;i++)
            testCounter = new Counter(i);

    }
}
```

Destructor in Java

```
i value is :194532      numberOfInstances is :33553
i value is :194533      numberOfInstances is :33554
i value is :194534      numberOfInstances is :33555
i value is :194535      numberOfInstances is :33556
i value is :194536      numberOfInstances is :33557
i value is :194537      numberOfInstances is :33558
i value is :194538      numberOfInstances is :33559
i value is :194539      numberOfInstances is :33560
i value is :194540      numberOfInstances is :33561
i value is :194541      numberOfInstances is :33562
i value is :194542      numberOfInstances is :33563
i value is :194543      numberOfInstances is :33564
i value is :194544      numberOfInstances is :33565
i value is :194545      numberOfInstances is :33566
i value is :194546      numberOfInstances is :33567
i value is :194547      numberOfInstances is :33568
i value is :194548      numberOfInstances is :33569
```

```
i value is :378663      numberOfInstances is :44936
i value is :378664      numberOfInstances is :44937
i value is :378665      numberOfInstances is :44938
i value is :378666      numberOfInstances is :44939
i value is :378667      numberOfInstances is :44940
i value is :378668      numberOfInstances is :44941
i value is :378669      numberOfInstances is :44942
i value is :378670      numberOfInstances is :44943
i value is :378671      numberOfInstances is :44944
i value is :378672      numberOfInstances is :44945
i value is :378673      numberOfInstances is :44946
i value is :378674      numberOfInstances is :44947
i value is :378675      numberOfInstances is :44948
i value is :378676      numberOfInstances is :44949
i value is :378677      numberOfInstances is :44950
i value is :378678      numberOfInstances is :44951
i value is :378679      numberOfInstances is :44952
i value is :378680      numberOfInstances is :44953
```

Destructor in Java

```
i value is :504787      numberOfInstances is :54567
i value is :504788      numberOfInstances is :54568
i value is :504789      numberOfInstances is :54569
i value is :504790      numberOfInstances is :54570
i value is :504791      numberOfInstances is :54571
i value is :504792      numberOfInstances is :54572
i value is :504793      numberOfInstances is :54573
i value is :504794      numberOfInstances is :54574
i value is :504795      numberOfInstances is :54575
```

```
i value is :632039      numberOfInstances is :7079
i value is :632040      numberOfInstances is :7080
i value is :632041      numberOfInstances is :7081
i value is :632042      numberOfInstances is :7082
i value is :632043      numberOfInstances is :7083
i value is :632044      numberOfInstances is :7084
i value is :632045      numberOfInstances is :7085
i value is :632046      numberOfInstances is :7086
i value is :632047      numberOfInstances is :7087
i value is :632048      numberOfInstances is :7088
i value is :632049      numberOfInstances is :7089
```