Practical Software Engineering I.

Introduction

Dr. Rudolf Szendrei

ELTE IK

2018.

Information

- Prerequisite (strong):
  - Programming
- Credits: 5
  - Lecture – 3
  - Practice (consultation) – 2
- Goal
  - Implement Object Oriented GUI applications in JAVA

Information

Final Grade

- An average of the grades of the three assignments you have to do during the semester
- Assignment (graded between 1 and 5)
  - It has to be submitted it on canvas and defended personally
  - It has to solve the given task, run without errors, and it has to be documented and tested
  - It should be your OWN work!
  - Each week delay results in -1 penalty point
- Each assignment needs to be graded for at least 2

Information

Contacts:

- Website:
  - https://swap.web.elte.hu/
- E-mail:
  - swap@inf.elte.hu
- Personally:
  - Southern building, Room 2.602

Object Oriented programming

Remembrance

- It focuses on the data and the participants of the task, instead of focusing on the functions or activities.
- We have to identify and group the participants, and explore their relationships and responsibilities. This is, how objects and classes are made.
- The functionality of the system is given by the set of the cooperating objects. An object is responsible for only one well defined task.
- An object can store data, but it is also liable to manage it. The object can hide its data from the outside. We can use the objects in a standard way.

Exercise

- Create a class to store information about employees.
- An employee has the following properties:
  - First name,
  - Last name,
  - Job,
  - Salary.
- Besided the necessary getter methods, provide an option to raise the salary of the employee.

Solution

```
package company;
public class Employee {
    private String firstName, lastName;
    private String job;
    private int    salary;
    ...
}
```

- Visibility of fields to other classes:
  - `public` visible for everybody
  - `protected` visible only in the inherited classes
  - `private` visible only in its own class

Solution

```
package company;
public class Employee {
    private String firstName, lastName;
    private String job;
    private int    salary;
    ...
}
```

- Visibility of class to classes of other packages
  - `public` can be seen from other packages
  - `-` can be used only in its own package (package private)

Solution

```
package company;
public class Employee {
    ...
    public  String getFirstName(){ return firstName; }
    public  String getLastName(){  return lastName; }
    public  int    getSalary(){    return salary; }
    public  String getJob(){       return job; }
    ...
}
```

- Name of a `boolean` Getter method: **`is`** + field name
- Name of other Getter methods: **`get`** + field name
- Name of the Setter method: **`set`** + field name

Solution

```
public class Employee {
    ...
    public Employee(String firstName, String lastName,
                    int salary,       String job) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.salary = salary;
        this.job = job;
    }
}
```

- There are no pointers in Java, but references.
  An object can refer to itself with the `this` keyword.

Solution

```
public class Employee {
    ...
    public void raiseSalary(int percent){
        salary = (int)(salary*(1.0 + percent / 100.0));
    }
    ...
}
```

- If an implicit type conversion may lead to information loss, then we have to use explicit type cast.

Java project structure

- Java project hierarchy:
  - Packages
  - Classes, interfaces, enumerations
- Codes of other packages can be reached using the `import` statement
- The name of the class and its file should be the same (except for nested classes)
- We define the visibility in front of each declared data type, field and method
- Fields are usually hidden (private), and they can be reached through their setter/getter methods

Exercise

- Write a main program, which reads the employee properties from the console, and instantiates the Employee class with these properties.
- Also, print out the properties of the employees to the console.

Solution

```
package company;
import java.util.Scanner;
public class EmployeeTester {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("First name: ");
        String firstName = sc.nextLine();
        System.out.print("Last name:  ");
        String lastName  = sc.nextLine();
        System.out.print("Job:        ");
        String job       = sc.nextLine();
        System.out.print("Salary:     ");
        int    salary    = sc.nextInt();
        Employee e = new Employee(firstName, lastName, salary, job);
        ...
    }
}
```

Solution

- The entry point of a program can be one of its static main method:

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

- Static methods belong to the class,
  so they can reach only static methods and fields.
- In Java language:
  - Console input: `System.in`
  - Console output: `System.out`
- Print to console: Using the print(…) methods
- Read from console: Calling the right next(…) method of the Scanner object.

Solution – improved version

- Our solution contains some code repetition.
  Repeated code parts can be put into methods.
  Define the following two methods to avoid code repetition.

```
public static String readString(Scanner sc, String msg){
  System.out.print(msg);
  return sc.nextLine();
}

public static int readInt(Scanner sc, String msg){
  System.out.print(msg);
  int i = sc.nextInt(); // It leaves the ENTER in the buffer
  sc.nextLine();        // Remove ENTER from the buffer
  return i;
}
```

Solution with methods

```
public static void main(String[] args) {
   Scanner sc = new Scanner(System.in);
   String firstName = readString(sc, "First name: ");
   String lastName  = readString(sc, "Last name:  ");
   String job       = readString(sc, "Job:        ");
   int    salary    = readInt(sc,    "Salary:     ");
   int    raise     = readInt(sc,    "Raise:      ");
   Employee e = new Employee(firstName, lastName,
                             salary, job);
   System.out.println(e.getFirstName() + " " +
                      e.getLastName()  + " job: " +
                      e.getJob()       + ", salary: " +
                      e.getSalary());
   e.raiseSalary(raise);
   System.out.println("Raised salary: ");
   System.out.println(e); // uses toString() implicitly
}
```

toString

- Producing a textual form of the information contained in an object is often complicated, so we want to avoid defining this procedure repeatedly.
- Creating a method which does this procedure is a smart move, and Java has the proper way to do this.
- Implement the following method of the class which has to be printed out:

```
@Override
public String toString() {
   return firstName + " " + lastName +
          "'s job: " + job +
          ", salary: " + salary;
}
```

toString

- It gives the String representation of simple data types (boolean, char, int, float, double, String etc.).
- In case of objects, it returns the reference of them.
- We can override this method to define what this will return. It greatly simplifies the code which prints out to the console.
- We can override all the non private methods of a super class. This ovveriding is annotated with the `@Override` annotation.
- What is the super class of **`Employee`**?
  - In Java, **`Object`** is a super class of every other classes

Exercise

- Use the previous solution
- Create a container, in which we can put employees
- Employees are read from the console
- Ask the job and the amount of raise from user to raise the salaries of our employees who have the given job
- Print out all the information about the employees to the console
- Print out which employee has the biggest salary and what is his/her job

Solution

- In Java, there are plenty of data structures we can choose from. We decide mostly by the way of usage/storage about which one to use:
  - Indexable:
    - ArrayList, ArrayLinkedList, Vector, Stack…
  - Linked list based:
    - Queue, DeQueue, PriorityQueue, LinkedList…
  - Tree data structure based:
    - TreeSet, TreeMap…
  - Hash function/table based:
    - HashSet, LinkedHashSet, HashTable, HashMap…
- We choose now the ArrayList

Solution

- Read the Employees in a loop and store them
- Reading employees can be put into a method
- Read the parameters of the modification
- Iterate over the employees, and raise the salary of the corresponding employees
- Run a maximum search on the employees to find most paid one
- Print out the Employee who earns the most

Solution – Read an employee

```
public static Employee readEmployee(Scanner sc){
  String firstName = readString(sc, "First name: ");
  String lastName  = readString(sc, "Last name:  ");
  String job       = readString(sc, "Job:        ");
  int    salary    = readInt(sc,    "Salary:     ");
  return new Employee(firstName, lastName,
                      salary, job);
}
```

Solution – Read and store employees

```
public static void main(String[] args){
  Scanner sc = new Scanner(System.in);
  ArrayList<Employee> employees = new ArrayList<>();

  for (int i = 0; i < 3; i++) {
    employees.add(readEmployee(sc));
    System.out.println(employees.get(i));
  }
  ...
}
```

Solution – Raising the salary

```
public static void main(String[] args){
   ...
   int    raise = readInt(sc,    "Raise: ");
   String job   = readString(sc, "Job:   ");

   for (Employee e : employees) {
     if (e.getJob().equals(job)) e.raiseSalary(raise);
     System.out.println(e);
   }
   ...
}
```

Comparison of objects

- We used the `equals()` method to compare the jobs, because == means the comparison of references in case of objects
- In case of Strings the `equals()` and `equalsIgnoreCase()` methods can be used as regular string comparisons
- We can define the way of comparison in our classes by implementing the `equals()` and `hashCode()` methods
  (these methods can be generated automatically in most Java developer environments).

Solution – Maximum search

```
public static void main(String[] args){
  ...
  Employee richMan = employees.get(0);
  for (Employee e : employees)
    if (e.getSalary() > richMan.getSalary())
      richMan = e;

  System.out.println("Employee with biggest salary: "
                     + richMan);
}
```

Lacks of the solution

- User can type anything in the console, e.g.: string instead of a number → Validation is required
- Test data should be typed manually during the debugging
- `equals` and `hashCode` methods of class **`Employee`** are not implemented, so we cannot (mustn't) use data structures based on the hash function
- Solutions to these problems will be discussed later

JAVA developer tools and environments

Frequently used tools

- JAVA Development Kit (JDK)
- JAVA Runtime Environment (JRE)
- JAVA documentation
  https://docs.oracle.com/javase/8/docs/

- NetBeans http://netbeans.org/
- Eclipse http://www.eclipse.org/
- IntelliJ IDEA http://www.jetbrains.com/idea/

Netbeans HotKeys / Shortcuts

Run:	F6
Run current file:	Shift + F6
Code completion:	Ctrl + Space
Code generation:	Alt + Insert
Code formatting:	Alt + Shift + F
Tips to solve the errors	Alt + Enter
Renaming:	Ctrl + R
Introduce variable from expression:	Alt + Shift + V
Introduce attribute from expression:	Alt + Shift + E
Create method from code :	Alt + Shift + M