

# Lab 3: Inheritance, Hash, Design Pattern and Big Number

## Objectives

- Learn how to use inheritance concept in Java programming
- Learn how to use interface concept in Java programming
- Learn the importance of `equals` and `hashCode`
- Get familiar with Java design patterns
- Learn how to calculate with big numbers in Java
- Full mark: 25 points

## Source files

- [Printable.java](#)
- [SalaryRaisable.java](#)
- [JavaDPExample.java](#)

## 1 Introduction

### 1.1 Inheritance in Java

Different kinds of objects often have a certain amount in common with each other. Employees, customers, and managers, for example, all share the characteristics of a person (name, etc.). Yet each also defines additional features that make it different. Object-oriented programming allows such different classes to inherit commonly used state and behavior from other classes. In this example, `Person` is the superclass of `Employee` and `Customer`; `Employee` is the super class of `SwEngineer` and `HwEngineer`; `SwEngineer` is the superclass of `ProjectManager`.

In Java, each class is allowed to **have and only have one direct superclass**, and each superclass has the potential for an unlimited number of subclasses. A subclass contains all the methods (functions) and attributes (variables) defined in the superclass plus the subclass's own methods and variables.

Java implements inheritance through the `extends` keyword.

Example: a software engineer has all the attributes of an employee plus attributes picked up from a person, which means the software engineer is an employee, and an employee is a person, and therefore the software engineer is also a person. In Java we would represent this as:

```
class SwEngineer extends Employee
class Employee extends Person
```

By doing so, `SwEngineer` gets all the attributes and methods defined in: `SwEngineer`, `Employee` and `Person`.

NOTE: A method is usually `public`/`private`. An attribute is usually `private`, meaning that it cannot be directly accessed outside of the class. A "getter" and "setter" method enables such access. For example:

```
Person person = new Person();
person.setName("John Doe");
int age = person.getAge();
```

## 1.2 Interface

Interfaces define a standardized set of commands that a class will obey. The commands are a set of methods. The interface definition states: the names of the methods; the return types; and the signatures (argument lists). There is no executable body for any method. The body is left to each class to implement.

A class can implement **multiple interfaces**. Once a class implements an interface, you are able to invoke the corresponding method. That is, implementing an interface enables a class to be "plugged in" to any situation that requires a specific behavior (manifested through the set of methods).

Java implements interface through the `implements` keyword.

```
interface SalaryRaisable {
    public double RaiseSalary();
}

class Employee implements SalaryRaisable {
    double baseSalary;
    @Override
    public double RaiseSalary() {
        return baseSalary *= 1.2;
    }
}
```

## 2 Deliverable 1 -- Inheritance and Interfaces

In this part you will write a program to represent people who are involved in a software production company (name, department, related projects...), as mentioned at the beginning of this lab.

*Step 1: create classes*

Please refer to the following [UML class diagram](#). Your program will **only** have the classes and interfaces mentioned in the diagram (you can put the main entry `public static void main(String[] args)` and test cases anywhere you like), where the two interfaces `Printable` and `SalaryRaisable` are already provided. Also, your classes will **only** have the attributes and methods mentioned in the diagram.

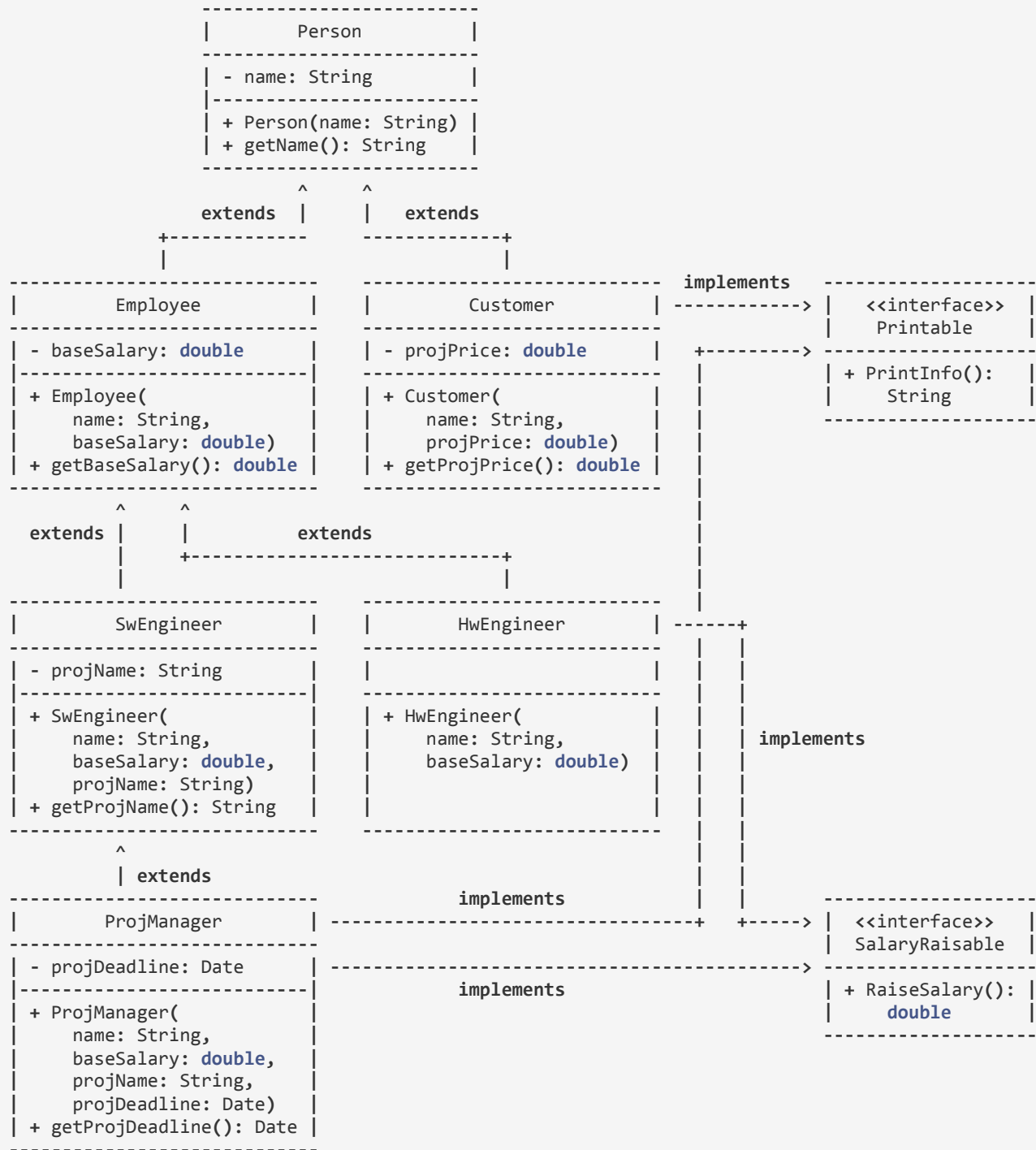
**HINT:** You can use `super` to call the constructor of the super class.

*Step 2: implement `SalaryRaisable`*

The `RaiseSalary` method raises the salary in a certain rate and returns the raised salary amount:

Role	Rate
HwEngineer	0.18
ProjManager	0.24

Create an instance of `HwEngineer` with base salary of \$3000, and an instance of `ProjManager` with base salary of \$6000. Display their final (raised) salaries -- they should be different.



Step 3: implement `PrintInfo`

The printed information shall be:

- `Customer`: name + project price;
- `ProjManager`: name + project name + final salary + project deadline.

**DEMO this deliverable to the lab instructor (10 points).**

### 3 Deliverable 2 -- `equals` and `hashCode`

Comparison is a common activity, hence nearly every class has its own definition of `equals` and `hashCode`.

Override the two methods for `SwEngineer` and `ProjManager`. Your implementations must follow the best practice as discussed in the lecture.

**Note 1:** even if you only need a customized `equals`, you still need to override `hashCode` -- this is important.

**Note 2:** if you override `equals` and `hashCode` of a subclass, you may need to override the superclass, too. This is because the hash code of the subclass relies on that of its superclass.

**DEMO this deliverable to the lab instructor (5 points).**

### 4 Deliverable 3 -- Java Design Pattern

Consider the code from *JavaDPExample.java* and provide answers to these questions (google useful clues if you need):

1. Why do we use a static method in this situation?
2. The code implements a class-level (involving multiple classes) programming "good practice", commonly these practices are called design patterns in Java. Which design pattern is implemented?
3. Explain why this is considered a good practice.

**DISCUSS this deliverable with the lab instructor (5 points).**

### 5 Deliverable 4 -- Big Numbers

Consider the following piece of C code. What is it doing? Convert it into Java codes.

```
uint64_t fnv(void *b, int c) {
    unsigned char *p = b;
    uint64_t h = 14695981039346656037;
    int i;
    for (i = 0; i < c; i++)
        h = (h * 1099511628211) ^ p[i];
    return h;
}
```

**Hint 1:** `h` is too big even for `long`, because Java doesn't have unsigned version of `int/long`. So you may need `BigInteger`.

**Hint 2:** `^` in C is "bitwise XOR", not `Math.pow`.

**Hint 3:** Don't forget the overflow of higher bits, or you may get incredibly large number. Such number is incorrect, and what's more, it may crash Eclipse.

**Demo and discuss this deliverable with the lab instructor (5 points).**