Problem #1

Your task is to implement classes and interfaces from the figure below the task. The classes are used to manage salary payments and they are done using already implemented class *BankManager*.

```
public class BankManager {
  public void payment(Worker worker, double amount) {
    System.out.println(worker.getName() + " - " + worker.getBankNumber() + ": " +
      amount);
  }
}
```

Class Worker is a base class for workers and it enables calculation of workers salary and creating payments orders (defined with `SalaryCalculator` interface).

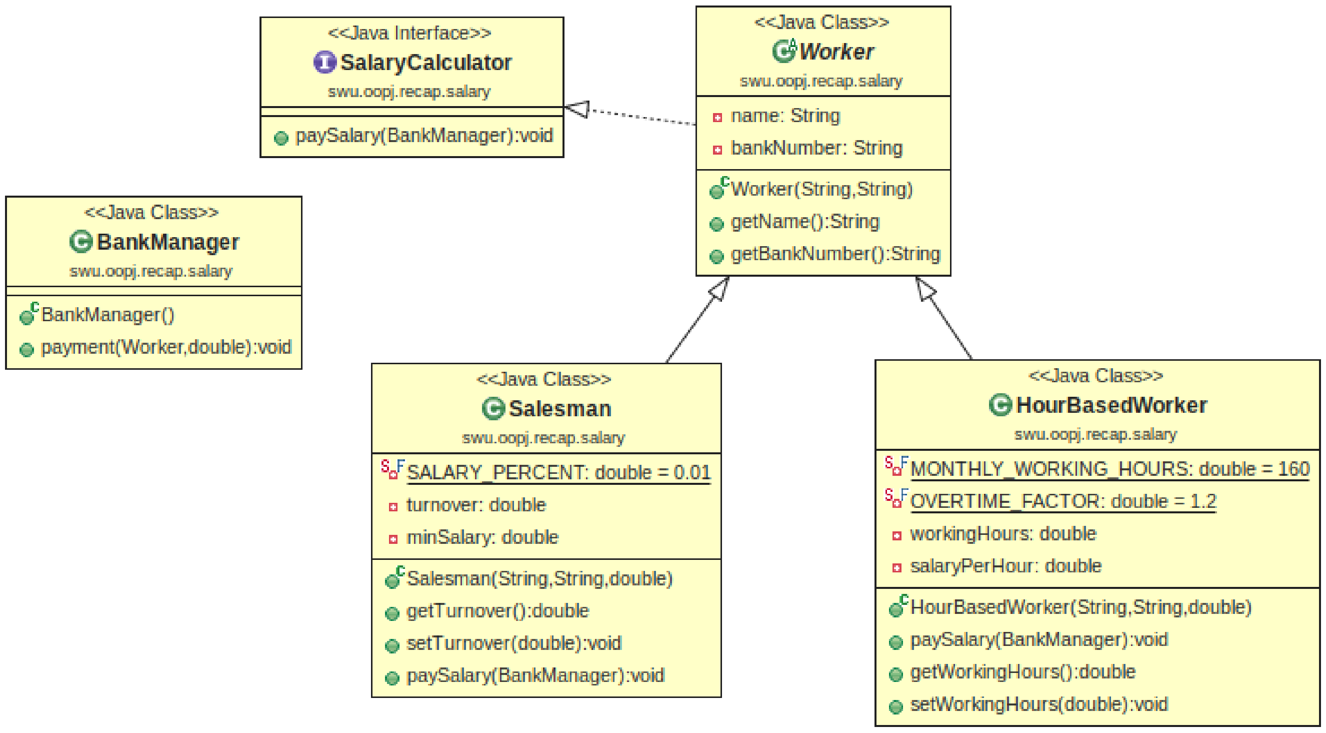
There are two types of workers. Class `Salesman` defines workers that has guaranteed minimal salary (`minSalary`) increased for a percentage (constant `SALARY_PERCENT` with value 1%) of turnover. Class `HourBasedWorker` defines workers payed per hour (`salaryPerHour`) for the first 160 hours (constant `MONTHLY_WORKING_HOUR`). For every overtime hour, salary per hour is increased for overtime factor(constant `OVERTIME_FACTOR` = 1,2).

Classes BankManager and Main are already implemented. Code of the main class is:

```
public class Main {
  public static void main(String[] args) {
    BankManager bankManager = new BankManager();

    Worker employeeList[] = new Worker[3];

    Salesman salesman = new Salesman("s1", "s1b", 3000);
    salesman.setTurnover(10000);
    employeeList[0] = salesman;

    HourBasedWorker hourWorker1 = new HourBasedWorker("h1_no", "h1", 50);
    hourWorker1.setWorkingHours(100);
    employeeList[1] = hourWorker1;

    HourBasedWorker hourWorker2 = new HourBasedWorker("h2_overtime", "h2", 50);
    hourWorker2.setWorkingHours(200);
    employeeList[2] = hourWorker2;

    for (SalaryCalculator salaryCalculator : employeeList) {
      salaryCalculator.paySalary(bankManager);
    }
  }
}
```

Program output:

```
s1 - s1b: 3100.0
h1_no - h1: 5000.0
h2_overtime - h2: 10400.0
```



Problem #2:

The task is to implement classes and interfaces used in a logistics application.

Cargo has properties that define its mass, volume, and a unique identifier.

Cargo objects can be stored in and removed from CargoHolder objects (model the current cargo in the holder using the array and set the array size to 100 that should be enough for the example). Cargo holders need a method for getting the total mass they're holding, called getWeight(). They also need to have a limited maximal weight and volume they can hold.

There are three subtypes of cargo holders: Container, BoxedCargoTruck, and ContainerShip.

A container has a defined weight, volume, and maximum volume (maxCargoVolume) and a unique identifier. Assume that the volume of a holder is always big enough to hold the maximal volume of cargo in it.

A truck can only carry a specific type of cargo -- BoxedCargo. It defines its weight and maximal allowed cargo weight (maxCargoWeight).

A ship carries cargo in the form of a concrete number of containers (maxContainers), and defines its weight and the maximal weight of cargo it can hold.

Example of a main program:

```
Cargo boxedCargo1 = new BoxedCargo(32.5, 56.7, 0);
Cargo boxedCargo2 = new BoxedCargo(18.9, 23.5, 1);

Container container = new Container(10, 100, 80, 3);
System.out.println(container.add(boxedCargo1));//true
System.out.println(container.add(boxedCargo2));//false
System.out.println(container.getWeight());//10 + 32.5 -> 42.5

System.out.println(container.remove(boxedCargo2));//false
System.out.println(container.remove(boxedCargo1));//true
System.out.println(container.getWeight());//10

CargoHolder truck = new BoxedCargoTruck(8.2, 60);
System.out.println(truck.add(boxedCargo1));//true
System.out.println(truck.add(boxedCargo2));//true
System.out.println(truck.getWeight());//8.2 + 32.5 + 18.9 -> 59.6

CargoHolder ship = new ContainerShip(120.3, 567.2, 60);
System.out.println(ship.add(container));//true
System.out.println(ship.remove(container));//true

System.out.println(truck.add(container));//false
System.out.println(ship.add(boxedCargo2));//false
```