Assignment 2

Handout: **Friday, 10 November 2023**
Due: **23:59:59, Monday, 20 November 2023**

Goals:

- To practice the use of Java data types and control structures.
- To understand the importance of information hiding.
- To design and implement Java classes.
- To get used to the IntelliJ Idea IDE.

NOTE

You MUST NOT change the names of the Java files or the signatures/return types of the methods in the assignment project. We use tests to grade your assignments, and the tests refer to the classes and methods defined in the given Java files. Failed tests due to changes to the file names and/or method signatures/return types will be treated the same as failed tests due to incorrect implementations.

1. Employee and Manager (10 points)

Read files `SalaryLevel.java`, `Employee.java`, and `Manager.java` in the assignment project, then add the missing code so that 1) the tests in EmployeeTest.java and ManagerTest.java will all pass and 2) the classes meet all the requirements specified in the Java files.

What to Do:

[Task 1] Add necessary code in `SalaryLevel.java`, `Employee.java`, and `Manager.java`.

2. Polymorphism and Dynamic Binding (12 points)

Suppose we have three classes `Hero`, `Warrior`, and `Healer` as defined below.

```
abstract class Hero{
  private String name;
  private int level;
  private int health;

  public Hero(String s){
    name = s;
    level = 1;
    health = 100;
  }

  public String getName() { return name;  }

  public int getLevel() {  return level;  }

  public int getHealth() {  return health;  }

```

```
    public void setHealth(int health){
        this.health = health;
        if(health == 0){
            System.out.println(name + " is dead.");
        }
    }

    public abstract void doAction(Hero other);

    public void levelUp(){
        level++;
        setHealth(100);
    }
}

class Warrior extends Hero{
    public Warrior(String name){   super(name);  }

    public void doAction(Hero other){
        int damage = Math.min(getLevel() * 5, other.getHealth());
        other.setHealth(other.getHealth() - damage);
        System.out.println(getName() + " attacks " + other.getName() + ". Does "
                            + damage + " damage.");
    }

    public void levelUp(){
        super.levelUp();
        System.out.println(getName() + " is now a level " + getLevel() + " warrior.");
    }
}

class Healer extends Hero{

    private int mana;

    public Healer(String name){
        super(name);
        mana = 100;
    }

    public void doAction(Hero other){
        if(mana >= 10){
            int h = Math.min(getLevel() * 10, 100 - other.getHealth());
            other.setHealth(other.getHealth() + h);
            mana -= 10;
            System.out.println(getName() + " heals " + other.getName() + " by "
                                + h + " points.");
        }
    }

    public void levelUp(){
        super.levelUp();
        mana = 100;
        System.out.println(getName() + " is now a level " + getLevel() + " healer.");
    }
}
```

Given the following variable declarations:

```
Hero hero;
Warrior warrior;
Healer healer;
```

For each of the code fragment below,
1) indicate whether it compiles, and
2) explain why the code fragment does not compile, if that's the case, or
3) specify the output of the code fragment, if it compiles and is executed.

Note this is a pen-and-paper exercise. You should NOT use a compiler or a computer to help you figure out the answers.

[Example]

```
warrior = new Warrior();
warrior.levelUp();
```

This code does not compile, because no default constructor is defined for class `Warrior`.

[Task 2]

```
warrior = new Warrior("Thor");
warrior.levelUp();
```

[Task 3]

```
hero = new Hero("Althea");
hero.levelUp();
```

[Task 4]

```
warrior = new Warrior("Thor");
```

```
healer = new Healer("Althea");
warrior.doAction(healer);
```

[Task 5]

```
warrior = new Healer("Diana");
warrior.levelUp();
```

[Task 6]

```
hero = new Warrior("Thor");
hero.doAction(hero);
hero = new Healer("Althea");
hero.doAction(hero);
```

[Task 7]

```
hero = new Warrior("Thor");
warrior = hero;
warrior.doAction(hero);
```

3. **Generic Set** (20 points)

Class `CompSet` implements a `HashSet` as described on page 7 of `LEC07 Object-Oriented Programming (II).pdf`.

What to Do:

[Task 8] Read and complete the class so that all requirements expressed in both `CompSetTest.java` and the code comments are satisfied. Note that you are not allowed to add new fields to class `CompSet`.

4. **BankAccount Using Monitor Operations** (10 points)

In this task, you will need to use synchronized methods/statements and operations like `wait`, `notify`, and `notifyAll` to coordinate the `withdraw` and `deposite` operations on a shared bank account.

What to Do:

[Task 9] Complete class `BankAccount` so that the test in `BankAccountTest.java` always executes successfully.

What to hand in:

The whole **Assignment2** folder, including the completed methods, compressed into a ZIP file.