Jobsheet 04 - Relationships Between Classes

I. Competency

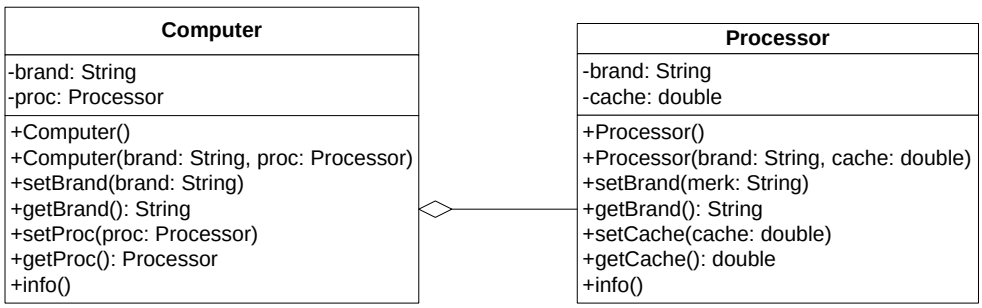
After taking this subject matter, students are able:

1. Understand the concept of relationships between classes;
2. Implement the most common relationships between classes: aggregation (has-a) in code.

II. Introduction

In more complex cases, in a system, more than one class will be found that has an interrelation between one class and another. In previous experiments, the majority of cases that had been worked on only focused on one class. In this jobsheet an experiment will be carried out involving several related classes.

For example there is a class of `Computer` that have attributes in the form of a `brand` and `processor`. If examined in more detail, the `processor` attributes themselves have data in the form of brands, memory cache values, and clock values. That is, there is another class called `Processor` which has `brand`, `cache` and `clock` attributes, and the processor attributes that are in the `Computer` class are objects of the `Proceessor` class. So it is seen between the `Computer` class and the `Processor` class have a relation (has-a).

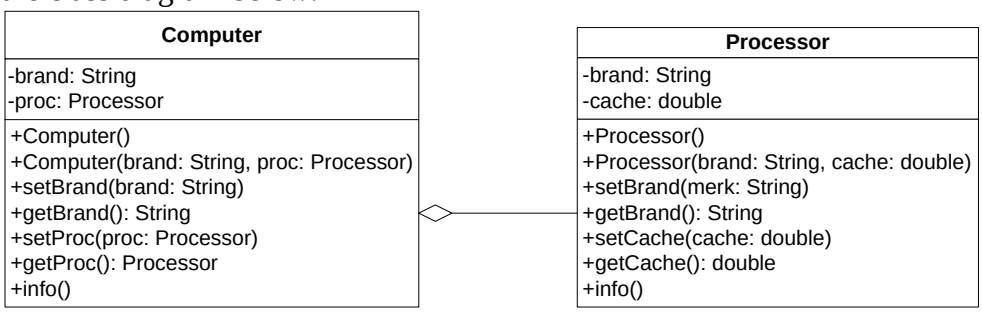


This type of has-a relation will be exemplified in the experiments in this jobsheet. When viewed in more detail, the relation is also called aggregation (has-a). Relationships between other classes are dependencies (uses-a) and inheritance (is-a). It takes an independent initiative from each student to deepen other types of relations, especially those not discussed in this course.

III. Experiments

Experiment 1

a. See the class diagram below:



b. Open *new project* in *Netbeans* and create *package* with the following format: `<identifier>.classrelation.experiment1` (replace `<identifier>` with your identity or other domains), for example: `ac.id.polinema`, `jti.polinema`, and

others).

Note: Naming packages with additional identifiers to avoid the possibility of naming conflicting classes.

c. Create the `Processor` class in your package.

```
public class Processor {

}
```

d. Add `brand` and `cache` attributes to the `Processor` class with `private` modifier access.

```
private String brand;
private double cache;
```

e. Create the default constructor for the `Processor` class.

f. Create a constructor for the `Processor` class with `brand` and `cache` parameters..

g. Implement setters and getters for the `Processor` class.

h. Implement the `info()` method as follows:

```
public void info() {
      System.out.printf("Brand Processor = %s\n", brand);
      System.out.printf("Cache Memory = %.2f\n", cache);
}
```

i. Then create the `Computer` class in the package that you created.

j. Add `brand` attributes with String type and `proc` with Object `Processor` type:

```
private String brand;
private Processor proc;
```

k. Create the default constructor for the `Computer` class.

l. Create a constructor for the `Computer` class with `brand` and `proc` parameters.

m. Then implement the `info()` method on the `Computer` class as follows

```
public void info() {
      System.out.println("Computer Brand = " + brand);
      proc.info();
}
```

n. In the same package, create the `MainExperiment1` class that contains the `main()` method.

o. Declare an Object `Processor` with the name p then instantiate it with the Intel i5 attribute information for `brand` value and 3 for `cache` value.

```
Processor p = new Processor("Intel i5", 3);
```

p. Then declare and instantiate the `Laptop` Object with the name L with information on the Thinkpad attribute and `Processor` Object that has been created.

q. Call the method the `info()` of the object `L`.

```
L.info();
```

r. Add the following line of code

```
Processor p1 = new Processor();
p1.setBrand("Intel i5");
p1.setCache(4);
Computer c1 = new Computer();
c1.setBrand("Thinkpad");
c1.setProc(p1);
c1.info();
```

s. Compile then run the `MainExperiment1` class, you will get the following result:

```
run:
Merk Laptop = Thinkpad
Merk Processor = Intel i5
Cache Memory = 3.00
Merk Laptop = Thinkpad
Merk Processor = Intel i5
Cache Memory = 4.00
BUILD SUCCESSFUL (total time: 0 seconds)
```

Questions

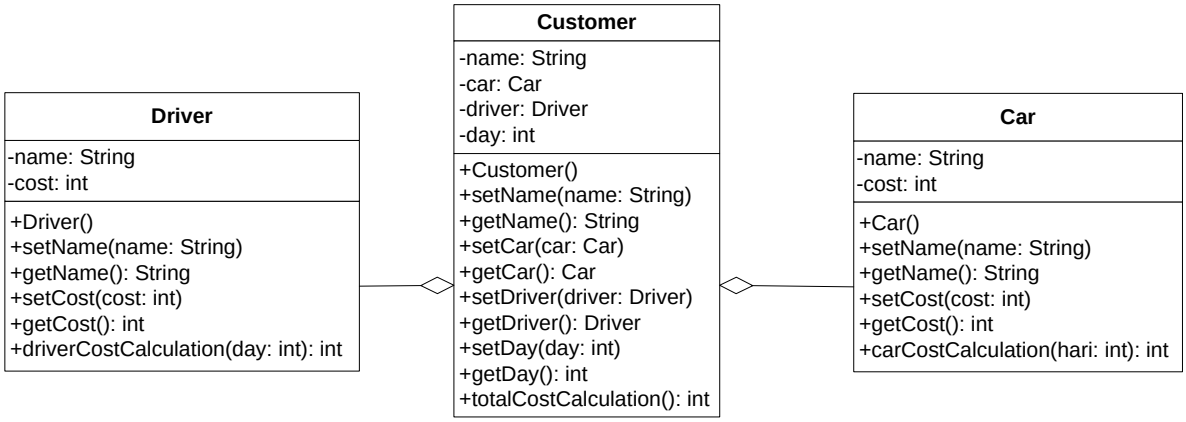
Based on experiment 1, answer the related questions:

1. In the `Processor` class and `Computer` class, there are setter and getter methods for each attribute. What the purpose of method setters and getters are ?
2. Within the `Processor` class and `Computer` class, there are each a default constructor and a parameterized constructor. How is the use of the two types of constructors different ?
3. Consider the `Computer` class, which of the 2 attributes (`brand` and `proc`), which attribute is object type ?
4. Look at the `Computer` class, which line shows that the `Computer` class has a relation with the `Processor` class ?
5. Pay attention to the `Computer` class, What is the syntax of `proc.info()` ?
6. In the `MainExperiment1` class , there is a line of code:
   `Computer c = new Laptop("Thinkpad", p);.`
   What is `p` ?

   And what happens if the line of code is changed to:
   `Computer c = new Laptop("Thinkpad", new Processor("Intel i5", 3));`
   How are the results of the program running, are there any changes ?

Experiment 2

Consider the following class diagram that illustrates the car rental system. Customers can rent a car and driver. Driver and car rental fees are calculated per day.



a. Add package `<identifier>.classrelation.experiment2`.

b. Create a `Car` class in the package.

c. Add `string` type `brand` attributes and `int` type `costs` with `private` modifier access.

d. Add the default constructor, setter, and getter.

e. Implement the `carCostCalculation` method

```
public int carCostCalculation(int days) {
    return cost * days;
}
```

f. Add the `Driver` class with the attribute `name` in `String` type and `cost` in `int` type with `private` modifier access along with the default constructor.

g. Implement the `driverCostCalculation` method

```
public int driverCostCalculation(int days) {
    return cost * days;
}
```

h. Add a `Customer` class with a default constructor.

i. Add the attributes with the following `private` modifier access:

Attribute	Type
Name	String
car	Car
driver	Driver
day	int

j. Implement *setters* and *getters*.

k. Add the `totalCostCalculation` method

```
public int totalCostCalculation() {
    return car.carCostCalculation(day) +
    driver.driverCostCalculation(day);
}
```

l. Create a `MainExperiment2` class that contains the `main()` method. Add the following line of code:

```
Car c = new Car();
c.setName("Avanza");
c.setCost(350000);
Driver d = new Driver();
d.setName("John Doe");
d.setCost(200000);
Customer cust = new Customer();
cust.setName("Jane Doe");
cust.setCar(c);
cust.setDriver(d);
cust.setDay(2);
System.out.println("Total cost = " +
cust.totalCostCalculation());
```

m. Compile and run the `MainExperiment2` class, and see the results!

Questions

1. See the `Customer` Class. In Which program line that shows the `Customer` class has a relation with the `Car` class and `Driver` class ?
2. Pay attention to the method of calculating the Cost of Driver in the `Driver` class, and the method of calculating the Cost of a Car in the `Car` class. Why do you think that method must have a day argument ?
3. Pay attention to the code from the `Customer` class. What do method for `car.carCostCalculation(day)` and `driver.driverCostCalculation(day)` ?
4. See the `MainExperiment2` *class*. What are the code `cust.setCar(c)` and `cust.setDriver(d)` ?
5. See the `MainExperiment2` class. What do method for `cust.totalCostCalculation()` ?
6. See the `MainExperiment2` class, try adding to the last line of the main method and observe the changes as they run!

   ```
   System.out.println(cust.getCar().getBrand());
   ```

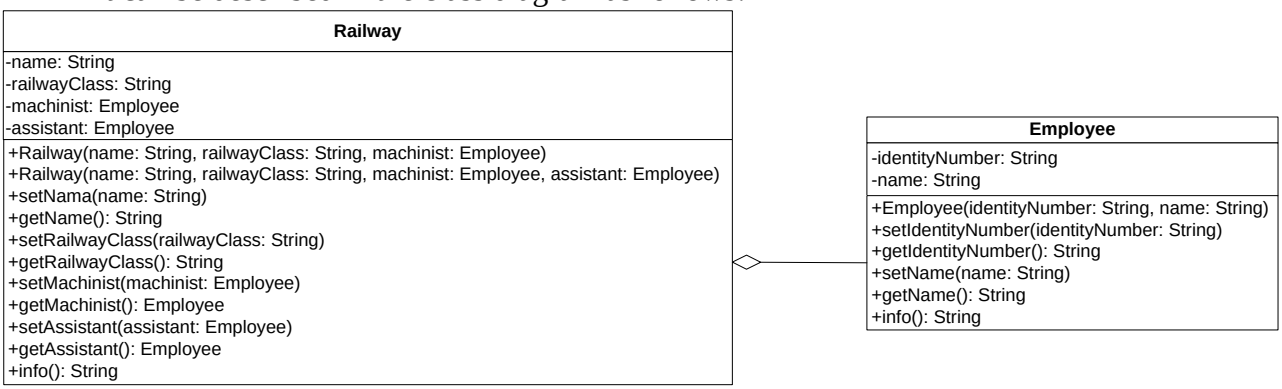
   So what is the syntax for `cust.getCar().getBrand()` in the main method ?

Experiment 3

In previous experiments, relations in class were stated in one-to-one. But sometimes class relations involve more than one. This is called multiplicity. For more detailed relations regarding multiplicity, can be seen in the following table.

Multiplicity	Description
0..1	0 or 1 instance
1	Exactly 1 instance
0..*	0 or more instance
1..*	At least 1 instance
n	Exactly n instance (n replaced with a number)
m..n	At least m instance, but no more than n

a. The Railway operated by engineer and an assistant engineer. Both Machinist and Machinist Assistants are both Employees of PT. Indonesian Railways. From the illustration of the story, it can be described in the class diagram as follows:



b. See and understand the class diagram, then open your IDE!
c. Create a package `<identifier>.classrelation.experiment3`, then add the `Employee` class.

d. Add attributes to the `Employee` class

```
private String identityNumber;
private String name;
```

e. Create a constructor for the `Employee` class with the `identityNumber` and `name` parameters.

f. Add setters and getters for each attribute.

g. Implement the `info()` method by writing the following line of code:

```
public String info() {
    String info = "";
    info += "Identity Number: " + this.identityNumber + "\n";
```

```
        info += "Name: " + this.name + "\n";
        return info;
    }
```

h. Create the `Railway` class based on the class diagram.

i. Add attributes to the `Railway` class in the form of `name`, `railwayClass`, `engineer`, and `assistant`.

```
private String name;
private String railwayClass;
private Employee machinist;
private Employee assistant;
```

j. Add three parameter in the constructor (`name`, `railwayClass`, `machinist`) and four parameter (`name`, `railwayClass`, `machinist`, `assistant`).

k. Add setters and getters for the attributes that exist in the `Railway` class .

l. Then implement the `info()` method

```
public String info() {
    String info = "";
    info += "Name: " + this.name + "\n";
    info += "Railway Class: " + this.railwayClass + "\n";
    info += "Machinist: " + this.machinist.info() + "\n";
    info += "Assistant: " + this.assistant.info() + "\n";
    return info;
}
```

m. Create the `MainExperiment3` class in the same package.

n. Add the `main()` method then write the following line of code.

```
Employee machinist = new Employee("1234", "Spongebob
Squarepants");
Employee assistant = new Employee("4567", "Patrick Star");
Railway railway = new Railway("Gaya Baru", "Bisnis",
machinist, assistant);

System.out.println(railway.info());
```

Questions

1. The `info()` method in the `Railway` class, the line of code `this.machinist.info()` and `this.assistant.info()` is used for what ?
2. Create a new main program with the name `MainQuestion` class in the same package. Add the following code to the `main()` method !

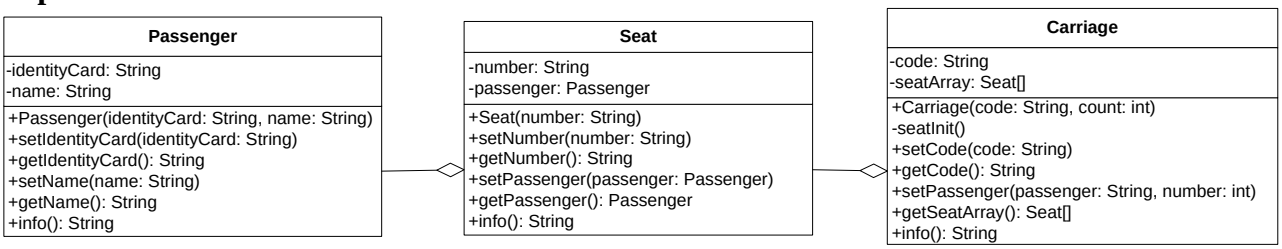
```
Employee machinist = new Employee("1234", "Spongebob
Squarepants");
```

```
Railway railway = new Railway("Gaya Baru", "Bisnis",
machinist);

System.out.println(railway.info());
```

3. What is the output from the main program ? Why did this happen ?
4. Fix the `Railway` class so the program can run !

Experiment 4



a. Pay attention and understand the class diagram.

b. Create each class for `Passenger`, `Seat` dan `Carriage` suit the design on the package `<identifier>.classrelation.experiment4`.

c. Add the `info()` method to the `Passenger` class:

```
public String info() {
    String info = "";
    info += "Identity Card: " + identityCard + "\n";
    info += "Name: " + name + "\n";
    return info;
}
```

d. Add the `info()` method to the `Seat` class:

```
public String info() {
    String info = "";
    info += "Number: " + number + "\n";
    if (this.passenger != null) {
      info += "Passenger: " + passenger.info() + "\n";
    }
    return info;
}
```

e. In the `Carriage` class create a `seatInit()` method with the `private` access.

```
private void seatInit() {
    for (int i = 0; i < seatArray.length; i++) {
       this.seatArray[i] = new Seat(String.valueOf(i + 1));
    }
}
```

f. Call `seatInit()` method in the constructor of `Carriage` so that the line of code into the following:

```
public Carriage(String code, int amount) {
    this.code = code;
    this.seatArray = new Seat[amount];
    this.seatInit();
}
```

g. Add the `info()` method to the `Carriage` class:

```
public String info() {
    String info = "";
    info += "Code: " + code + "\n";
    for (Seat seat : seatArray) {
        info += seat.info();
    }
    return info;
}
```

h. Implement a method to enter the passenger in accordance with the seat number.

```
public void setPassenger(Passenger passenger, int number) {
    this.arraySeat[number - 1].setPassenger(passenger);
}
```

i. Create a `MainExperiment4` class that contains `main()` method. Then add the following line of code!

```
Passenger p = new Passenger("12345", "Mr. Krab");
Carriage carriage = new Carriage("A", 10);
carriage.setPassanger(p, 1);
System.out.println(carriage.info());
```

Questions

1. In the `MainExperiment4` class, what is amount of seats in carriage A ?
2. Pay attention to the code snippet in the `info()` method in the `Seat` class. What does the code mean ?

```
...
if (this.passenger != null) {
    info += "Passenger: " + passenger.info() + "\n";
}
...
```

3. Why the `setPassengers()` method in `Carriage` class, the value of `number` is reduced by the number 1 ?
4. Instantiation of new `budi` object with the `Passenger` type, then insert the new object in the carriage with the `carriage.setPassenger(budi, 1)` method. What's happening ?
5. Modify the program so that it is not allowed to occupy the seat of another passenger !

IV. Assignment

Create a case study, design a class diagram, then implement in the code! The case studies should represent the relation class of experiments have been done on this matter, involving at least 4 classes (the main class does not count).