ENCAPSULATION

1. Competence

1. Constructor
2. Access Modifier
3. Attributes / methods in the class
4. Intensation of attributes / methods
5. Setter and getter
6. Understanding the notation in UML Class Diagrams

2.1 Encapsulation

Encapsulation is also called **information hiding**. In interacting with objects, often we do not need to know the complexity that is in it. This will be easier to understand if we imagine or analyze objects that are around us, for example bicycle objects, when we change gears on a bicycle, we just need to press the gear lever in the handlebar grip. We don't need to know how to move gear technically.

Examples of other objects such as vacuum cleaner (vacum cleaner), when we plug in the vacuum cleaner cable and turn on the switch, the machine is ready to be used to suck dust. In that process we do not know the complicated process that occurs when converting electricity into power from a vacuum cleaner. In the example above the vacuum cleaner and bicycle have implemented encapsulation or also called **information-hiding** or hiding data because they hide the details of the process of an object from the user.

2.2 Constructor

The constructor is similar to the method of the declaration method but has no return type. And the constructor is executed when the instance of the object is created. So every time an object is created with the new keyword () the constructor will be executed. The way to create a constructor is as follows:

1. The constructor name must be the same as the class name
2. The constructor does not have a data return type
3. Constructors may not use abstract, static, final, and synchronized modifiers

Note: In java we can have constructors with modifiers private, protected, public or default.

2.3 Modifier Access

There are 2 types of modifiers in java, namely: access modifier and non-access modifier.
In this case, we will focus on the access modifier, which is useful for managing access to methods, classes, and constructors. There are 4 access modifiers, namely:

1. private - can only be accessed in the same class
2. default - can only be accessed in the same package
3. protected - can be accessed outside the package using subclasses (creating inheritance)
4. public - can be accessed from anywhere

Modifier access details can be seen in Table 1.1.

Tabel 1. 1 Modifier Access

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

2.4 Getter and Setter

- ☐ Getters are public methods and have a data return type, which functions to get the value of the private attribute.
- ☐ Setter is a public method that does not have a data return type, which functions to manipulate the value of the private attribute.
- ☐ The difference between the method setter and the getter lies in the return value, parameters, and contents of the method.

2.5 UML Class Diagram Notation

In general, the form of UML class diagrams is as shown in Figure 1.1.

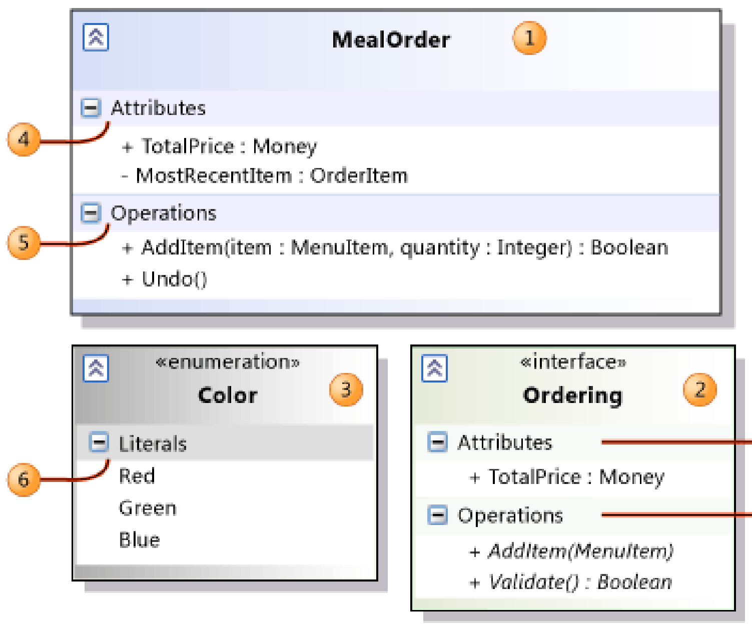


Figure 1. 1 Class Diagram

Description :

1. Class
2. Interface
3. Enumeration
4. Atrributes
5. Method
6. Literals

Modifier access notation in UML class diagram :

1. Plus sign (+) for public
2. The hash sign (#) for protected
3. Minus sign (-) for private
4. For default, it is not given a notation

3. Experiment

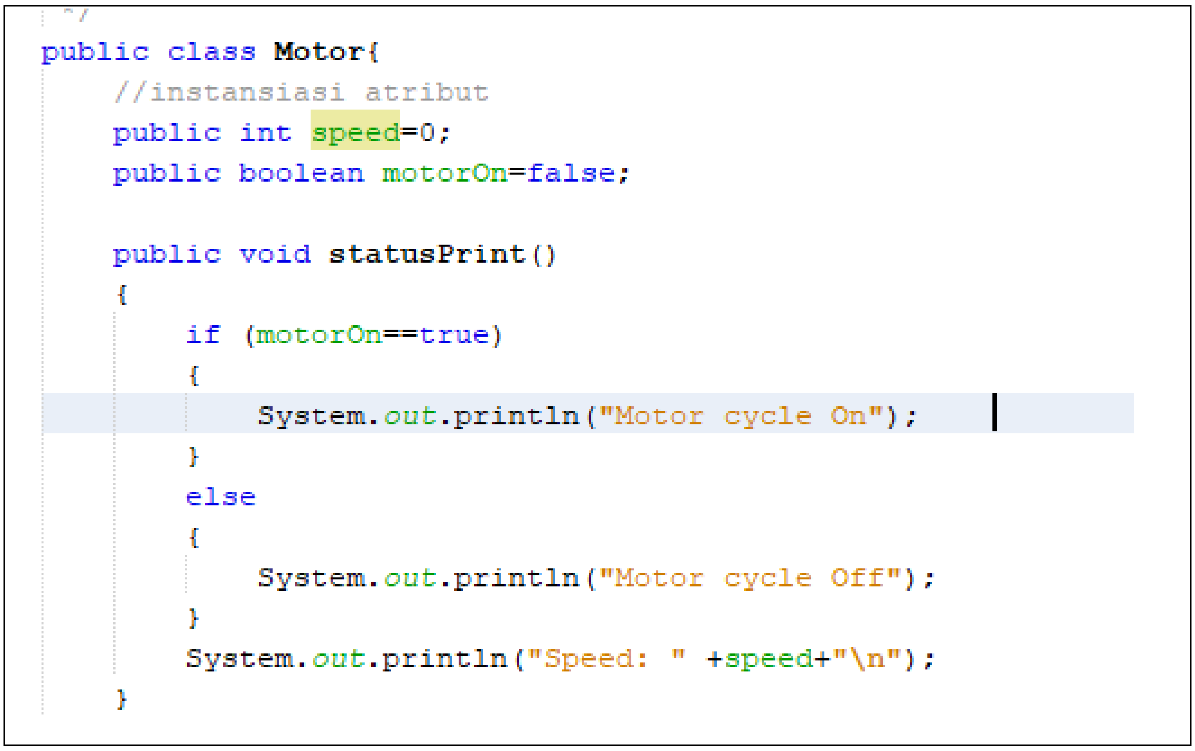
3.1 Experiment 1 - Encapsulation

In the encapsulation experiment, create a Motor class that has the speed and contact attributes, and has the printStatus () method to display the motor status. As follows

1. Open Netbeans, create a MotorEncapsulation project.

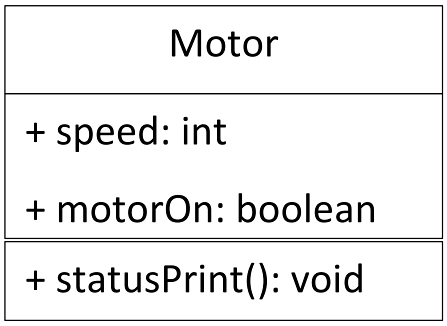
2. Create a Motor class. Right-click on the motorencapsulation package - New - Java Class.

3. Type the Motor class code below.

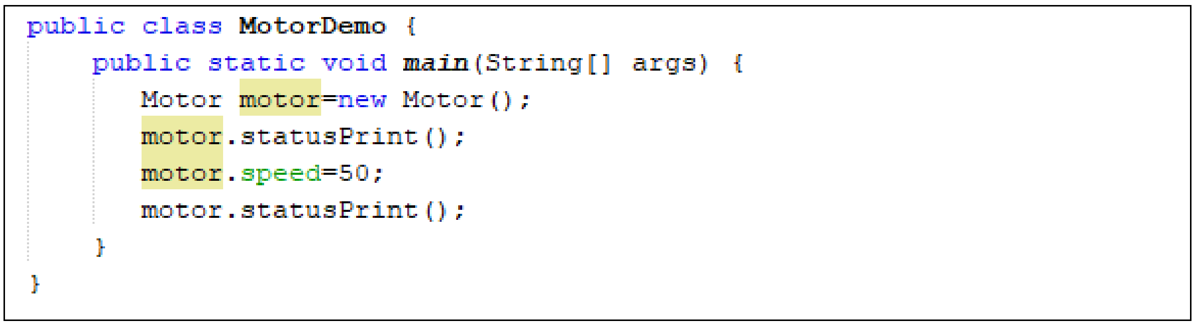


```
*/
public class Motor{
    //instansiasi atribut
    public int speed=0;
    public boolean motorOn=false;

    public void statusPrint()
    {
        if (motorOn==true)
        {
            System.out.println("Motor cycle On");
        }
        else
        {
            System.out.println("Motor cycle Off");
        }
        System.out.println("Speed: " +speed+"\n");
    }
```

Class diagram:

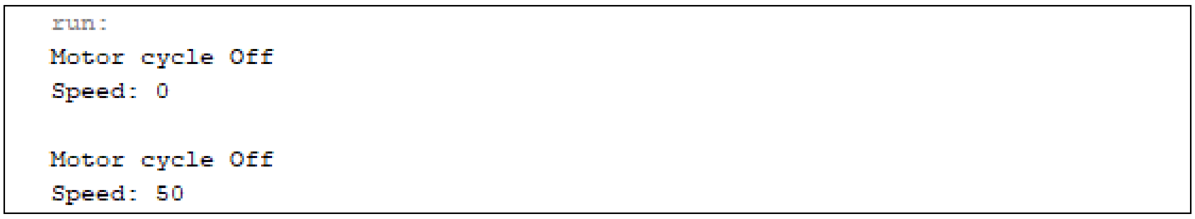


1. Create class MotorDemo,



```
public class MotorDemo {
    public static void main(String[] args) {
        Motor motor=new Motor();
        motor.statusPrint();
        motor.speed=50;
        motor.statusPrint();
    }
}
```

2. The result is:



```
run:
Motor cycle Off
Speed: 0

Motor cycle Off
Speed: 50
```

From experiment 1 - encapsulation, in your opinion, is there anything strange?

That is, the speed of the motor suddenly changes from 0 to 50. Even more awkward, the motor contact position is still in the OFF condition. How is it possible for a motor to be blinked from zero to 50, and even then the ignition is OFF?

Now in this case, access to motor attributes is apparently not controlled. In fact, objects in the real world always have limits and mechanisms for how these objects can be used. Then, how can we improve the Motor class above so that it can be used properly? We can consider the following:

1. Hiding internal attributes (speed, motorOn) from users (other classes)

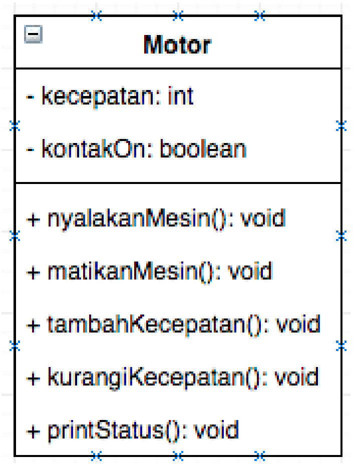
2. Provides a special method for accessing attributes.

For that, let's continue the next experiment about Access Modifier.

3.2 Experiment 2 – Modifier Access

In this experiment the access modifier will be used to improve the workings of the Motor class in the first experiment.

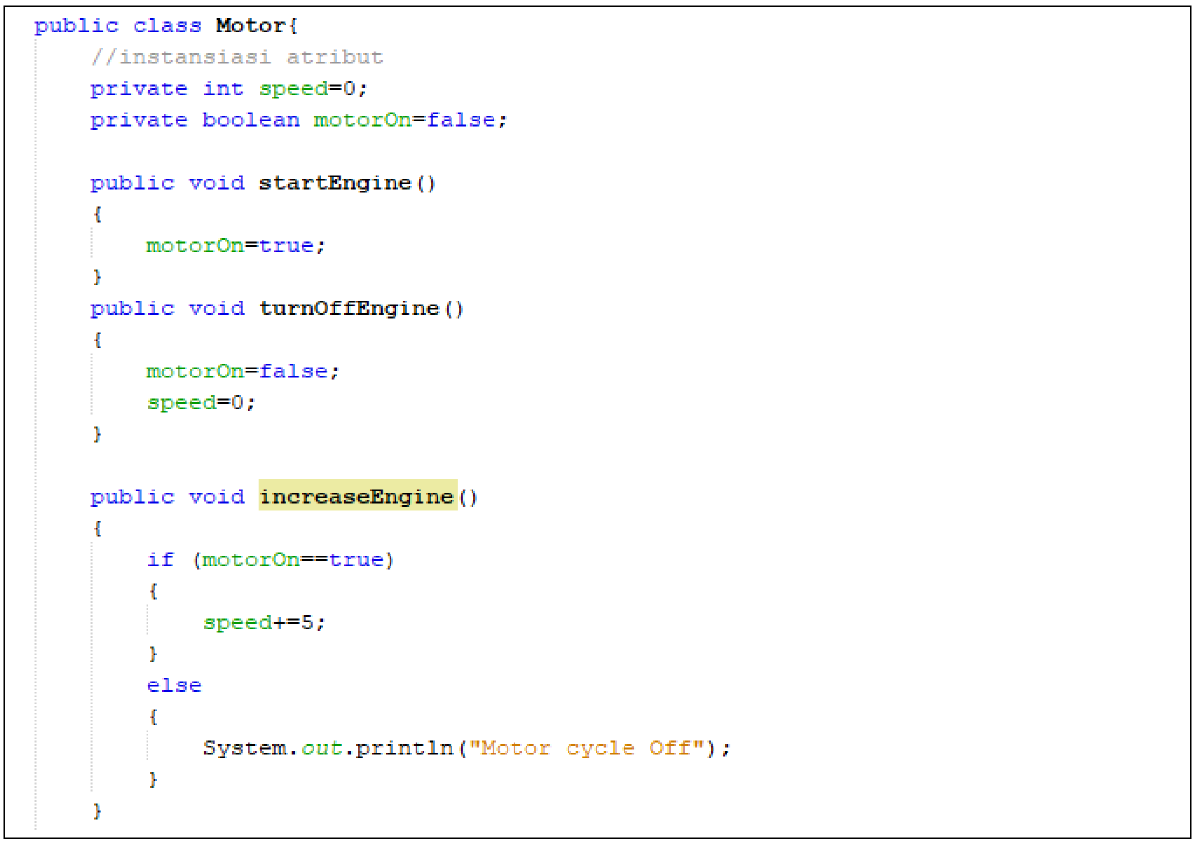
1. Change how the motor class works according to the following UML class diagram.



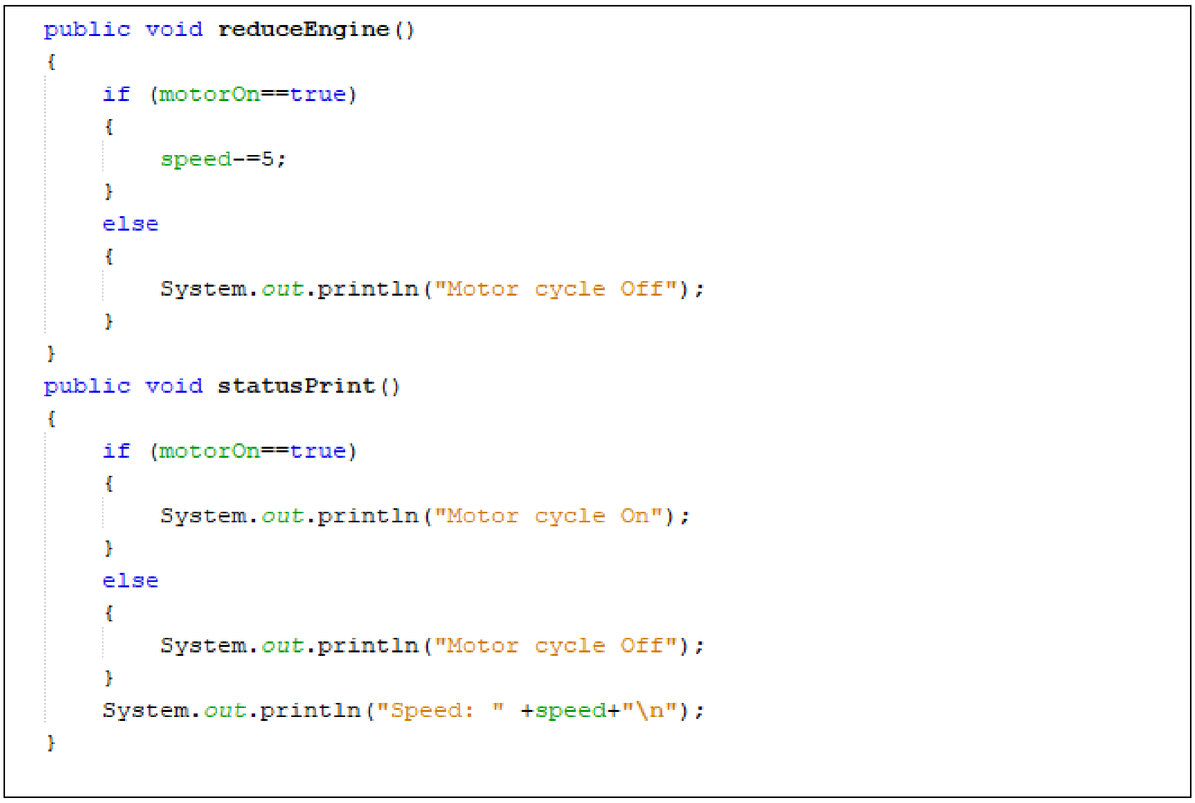
2. Based on the UML class diagram, there are changes in the Motor class, namely:

a. Change the access modifier kecepatan and motorOn to private

b. Add method startEngine, turnOffEngine, increaseEngine, reduceEngine.

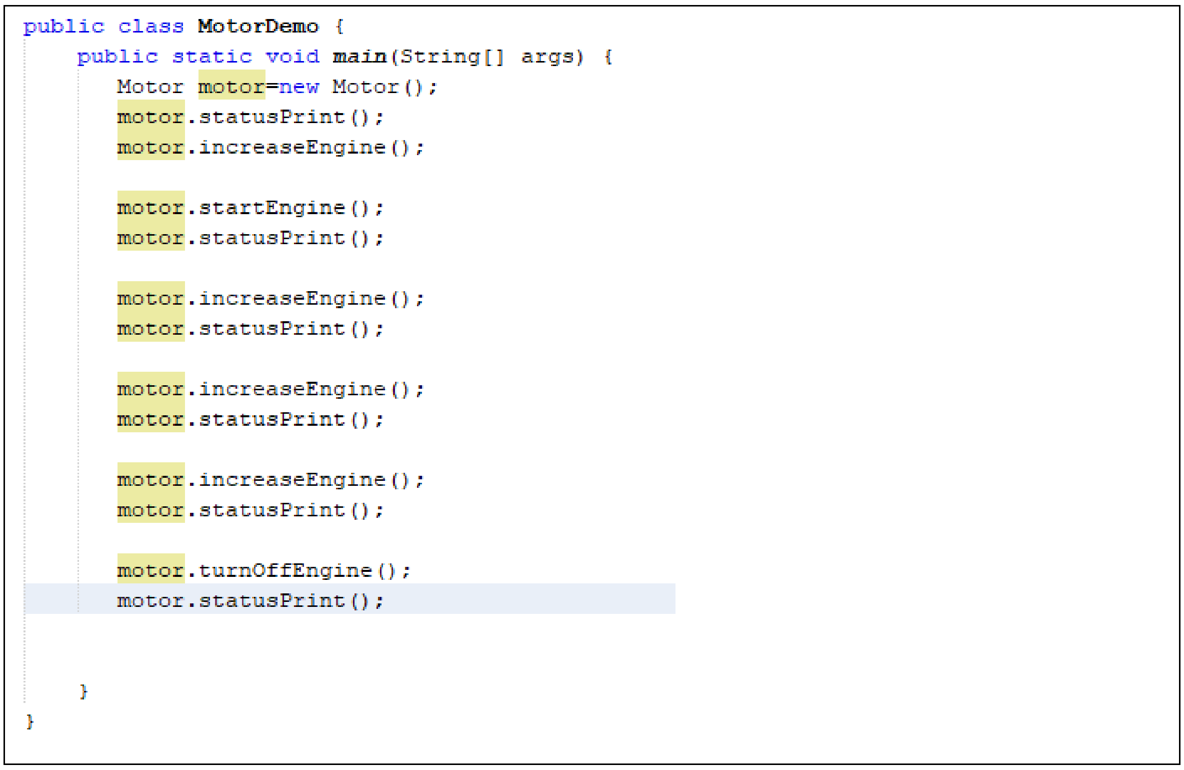


```java
public class Motor{
    //instansiasi atribut
    private int speed=0;
    private boolean motorOn=false;

    public void startEngine()
    {
        motorOn=true;
    }
    public void turnOffEngine()
    {
        motorOn=false;
        speed=0;
    }

    public void increaseEngine()
    {
        if (motorOn==true)
        {
            speed+=5;
        }
        else
        {
            System.out.println("Motor cycle Off");
        }
    }
```



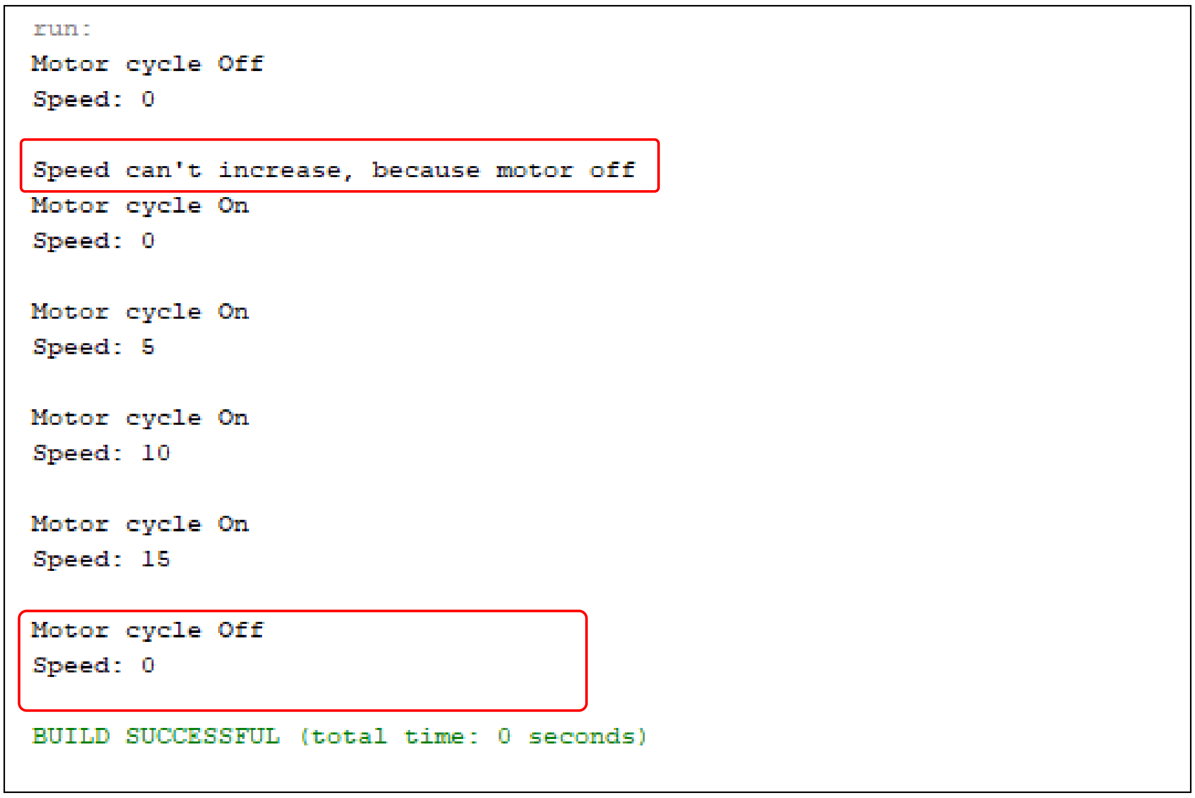
```java
public void reduceEngine()
{
    if (motorOn==true)
    {
        speed-=5;
    }
    else
    {
        System.out.println("Motor cycle Off");
    }
}
public void statusPrint()
{
    if (motorOn==true)
    {
        System.out.println("Motor cycle On");
    }
    else
    {
        System.out.println("Motor cycle Off");
    }
    System.out.println("Speed: " +speed+"\n");
}
```

3. Change class MotorDemo:



```java
public class MotorDemo {
    public static void main(String[] args) {
        Motor motor=new Motor();
        motor.statusPrint();
        motor.increaseEngine();

        motor.startEngine();
        motor.statusPrint();

        motor.increaseEngine();
        motor.statusPrint();

        motor.increaseEngine();
        motor.statusPrint();

        motor.increaseEngine();
        motor.statusPrint();

        motor.turnOffEngine();
        motor.statusPrint();


    }
}
```

4. The result is:



```
run:
Motor cycle Off
Speed: 0

Speed can't increase, because motor off
Motor cycle On
Speed: 0

Motor cycle On
Speed: 5

Motor cycle On
Speed: 10

Motor cycle On
Speed: 15

Motor cycle Off
Speed: 0

BUILD SUCCESSFUL (total time: 0 seconds)
```

From the above experiment, we can observe now that the speed attribute cannot be accessed by the user and is arbitrarily replaced. Even when trying to increase speed when the contact position is still OFF, a notification will appear that the engine is OFF. To get the desired speed, it must be done gradually, namely by calling the add Speed () method several times. This is similar to when we ride a motorcycle.

3.3 Questions

1. In the MotorDemo class, when we add speed for the first time, why does the warning "Speed cannot appear because the engine is off!"
2. Can the speed and contact attributes be set private?
3. Change the Motor class so that the maximum speed is 100!

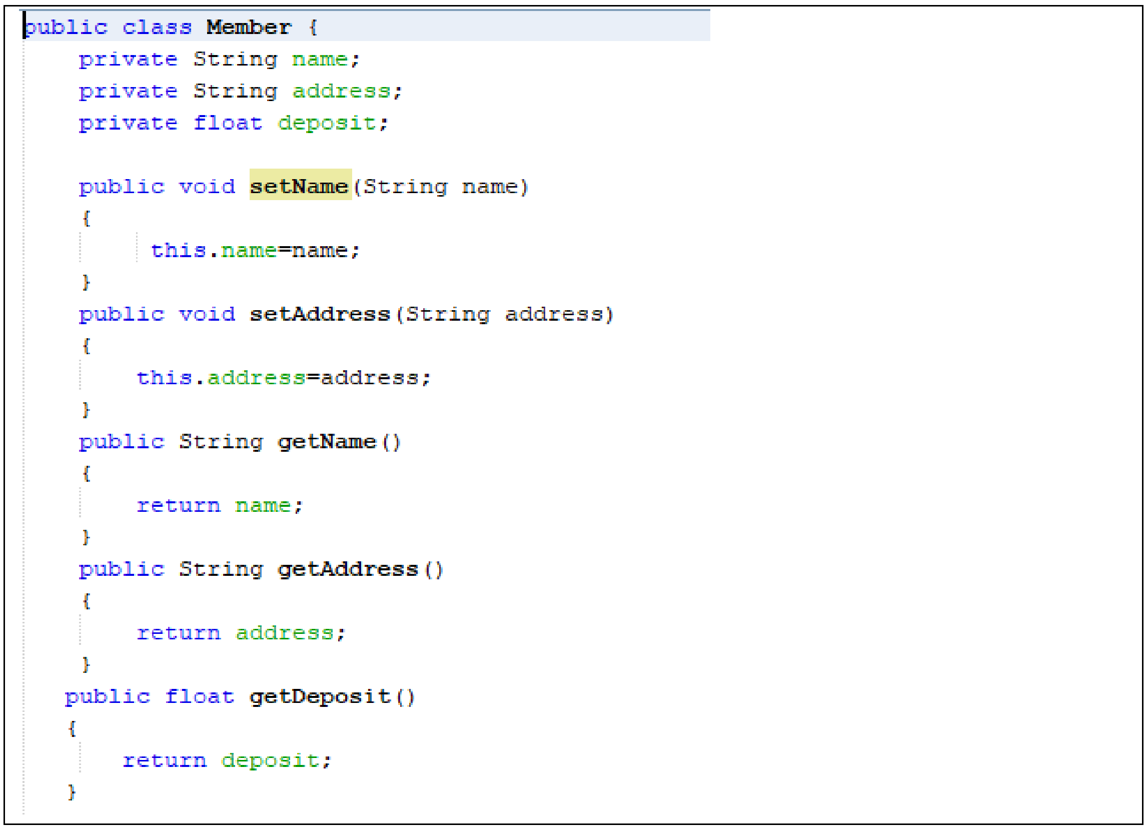
3.4 Experiment 3 - Getter and Setter

For example in a information system, there is a class of Members. Members have name, address and savings attributes, and method setters, getters and deposits and loans. All attributes of the members must not be changed arbitrarily, but can only be changed through the method setter, getter, deposit and pull. Specifically for the deposit attribute there is no setter because deposits will increase when making deposit transactions and will decrease when making loans / withdrawals.

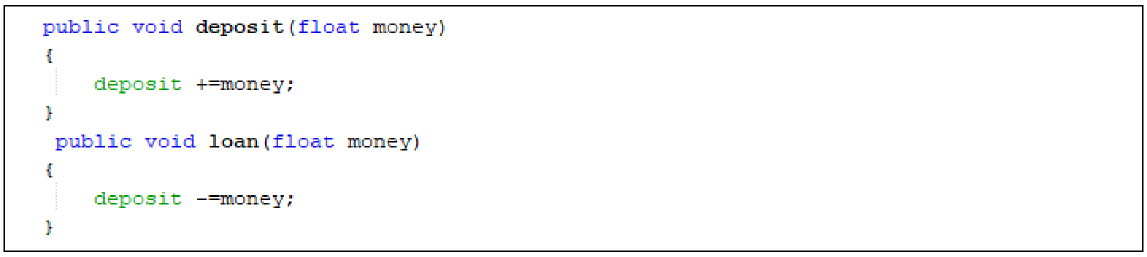
1. Following UML class:

Member
- name: String - address: String - deposit: float
+setName(String): void +setAddress(String): void +getName():String +getAddress():String +getDeposit():float +deposit(float): void +loan(float): void

2. Same with experiment 1 for creating a new project
    a. Open Netbeans, create a KoperasiGetterSetter project.
    b. Create a Anggota class. Right-click on the package package lettersetters - New - Java Class.
    c. Type the Anggota class code below.

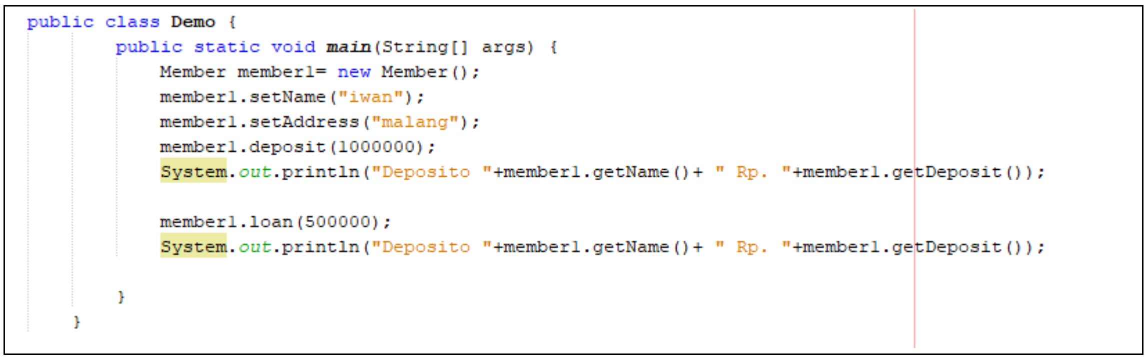


```
public class Member {
    private String name;
    private String address;
    private float deposit;

    public void setName(String name)
    {
            this.name=name;
    }
    public void setAddress(String address)
    {
        this.address=address;
    }
    public String getName()
    {
        return name;
    }
    public String getAddress()
    {
        return address;
    }
   public float getDeposit()
   {
       return deposit;
   }
```



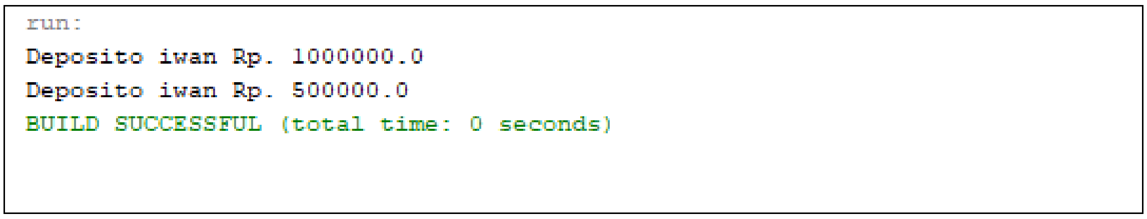
```java
public void deposit(float money)
{
    deposit +=money;
}
 public void loan(float money)
{
    deposit -=money;
}
```

2. Create class Demo



```java
public class Demo {
        public static void main(String[] args) {
            Member member1= new Member();
            member1.setName("iwan");
            member1.setAddress("malang");
            member1.deposit(1000000);
            System.out.println("Deposito "+member1.getName()+ " Rp. "+member1.getDeposit());

            member1.loan(500000);
            System.out.println("Deposito "+member1.getName()+ " Rp. "+member1.getDeposit());

        }
    }
```

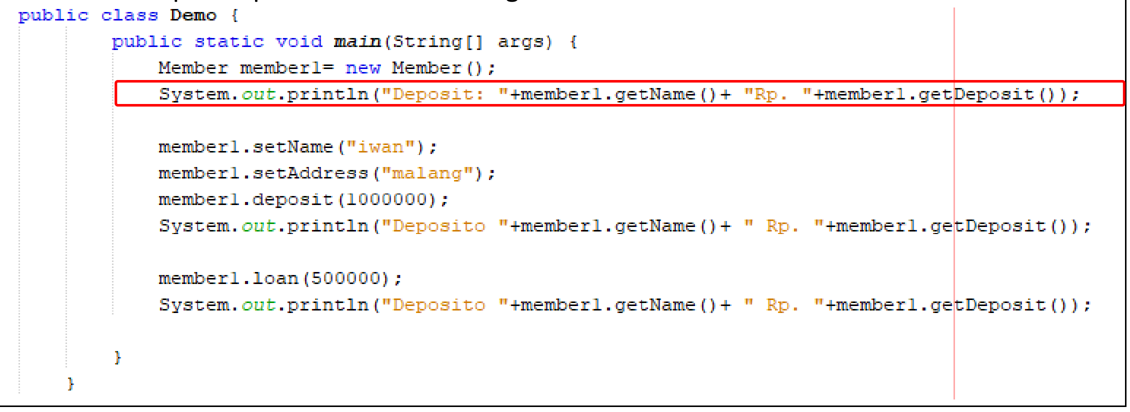
3. The result is:



```
run:
Deposito iwan Rp. 1000000.0
Deposito iwan Rp. 500000.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

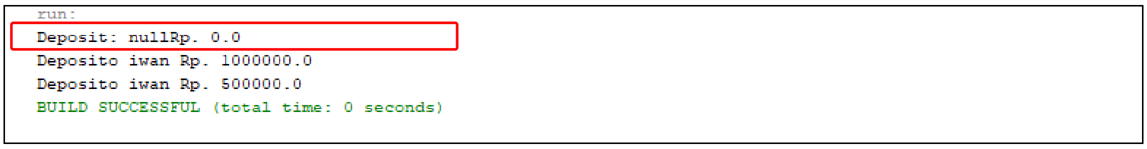
3.5 Percobaan 4 - Construktor, Instantiation

1. The first step of experiment 4 is to change the Demo class as follows



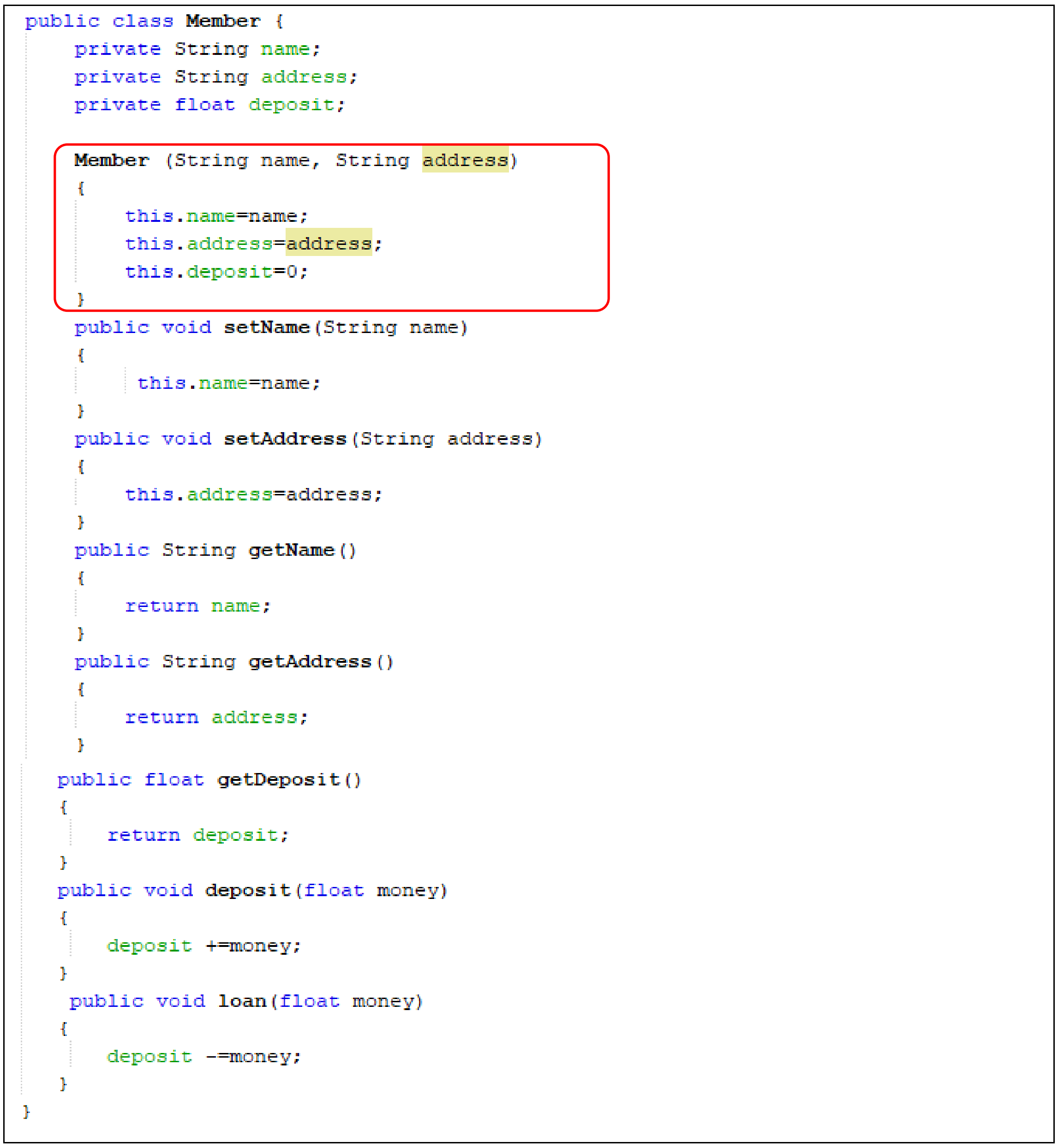
```java
public class Demo {
        public static void main(String[] args) {
            Member member1= new Member();
            System.out.println("Deposit: "+member1.getName()+ "Rp. "+member1.getDeposit());

            member1.setName("iwan");
            member1.setAddress("malang");
            member1.deposit(1000000);
            System.out.println("Deposito "+member1.getName()+ " Rp. "+member1.getDeposit());

            member1.loan(500000);
            System.out.println("Deposito "+member1.getName()+ " Rp. "+member1.getDeposit());

        }
    }
```

2. The result is:



```
run:
Deposit: nullRp. 0.0
Deposito iwan Rp. 1000000.0
Deposito iwan Rp. 500000.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

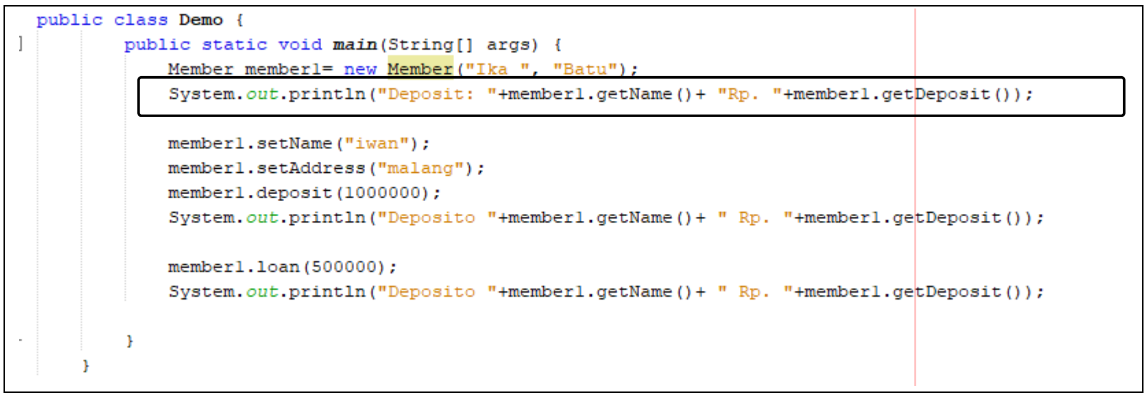
3. Change class Member



```
public class Member {
    private String name;
    private String address;
    private float deposit;

    Member (String name, String address)
    {
        this.name=name;
        this.address=address;
        this.deposit=0;
    }
    public void setName(String name)
    {
        this.name=name;
    }
    public void setAddress(String address)
    {
        this.address=address;
    }
    public String getName()
    {
        return name;
    }
    public String getAddress()
    {
        return address;
    }
   public float getDeposit()
   {
       return deposit;
   }
   public void deposit(float money)
   {
       deposit +=money;
   }
    public void loan(float money)
   {
       deposit -=money;
   }
}
```

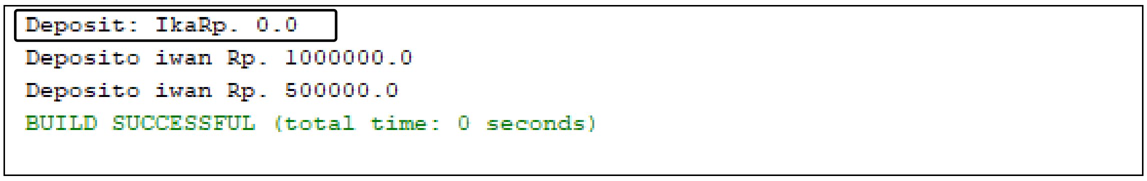
4. In the Anggota class, a constructor is created with a default access modifier that has 2 nama and alamat parameters. And in the constructor, the pinjam value for the first time is Rp. 0

5. Change class Demo



```
public class Demo {
    public static void main(String[] args) {
        Member member1= new Member("Ika ", "Batu");
        System.out.println("Deposit: "+member1.getName()+ "Rp. "+member1.getDeposit());

        member1.setName("iwan");
        member1.setAddress("malang");
        member1.deposit(1000000);
        System.out.println("Deposito "+member1.getName()+ " Rp. "+member1.getDeposit());

        member1.loan(500000);
        System.out.println("Deposito "+member1.getName()+ " Rp. "+member1.getDeposit());

    }
}
```

6. The result:


```
Deposit: IkaRp. 0.0
Deposito iwan Rp. 1000000.0
Deposito iwan Rp. 500000.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

After adding a constructor to the Member class, the member and address attribute must automatically be set first by passing the parameter when doing an instance of the member class. This is usually done for attributes that require specific values. If you don't need specific values in the constructor don't need parameters. For example, simpanan for new members are set to 0, then simpanan do not need to be used as parameters in the constructor.

3.6 Question - Experiments 3 and 4

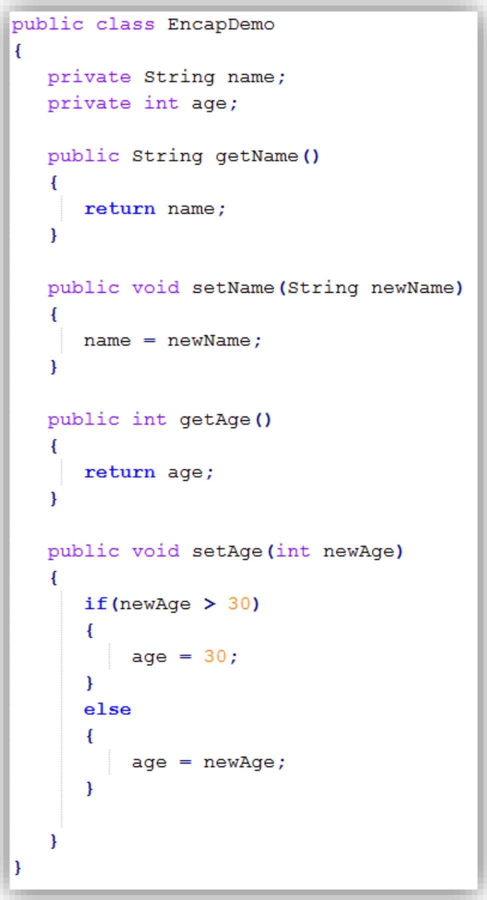
1. What are getters and setters?
2. What is the use of the getDeposit () method?
3. What method is used to add balance?
4. What does the constructor mean?
5. Mention the rules in making a constructor?
6. Can the constructor be private?
7. When to use parameters with passsing parameters?
8. What is the difference between class attribute and instance attribute?
9. What is the difference between the class method and the method instance?

4. Conclusion

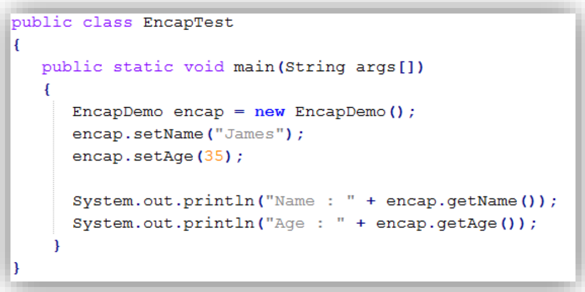
From the above experiments, we have learned the concepts of encapsulation, constructors, access modifiers consisting of 4 types namely public, protected, default and private. The concept of attribute or class methods that exist in the code class block and the concept of attribute instantiation or method. How to use getters and setters along with the functions of getters and setters. And have also learned or understood UML notation

5. Task

1. Try the program below and write the output results



```java
public class EncapDemo
{
    private String name;
    private int age;

    public String getName()
    {
        return name;
    }

    public void setName(String newName)
    {
        name = newName;
    }

    public int getAge()
    {
        return age;
    }

    public void setAge(int newAge)
    {
        if(newAge > 30)
        {
            age = 30;
        }
        else
        {
            age = newAge;
        }

    }
}
```



```java
public class EncapTest
{
    public static void main(String args[])
    {
        EncapDemo encap = new EncapDemo();
        encap.setName("James");
        encap.setAge(35);

        System.out.println("Name : " + encap.getName());
        System.out.println("Age : " + encap.getAge());
    }
}
```

2. In the above program, in the EncapTest class we set age with a value of 35, but when displayed on the screen the value is 30, explain why.

ANSWER:

3. Change the program above so that the age attribute can be given a maximum value of 30 and a minimum of 18.

ANSWER:

4. In a savings and loan information system, there are class Members who have attributes including KTP number, name, loan limit, and loan amount. Members can borrow money within the specified lending limits. Members can also repay loans. When the Member repays the loan, the loan amount will decrease according to the nominal installment. Create a Member class, provide attributes, methods and constructors as needed. Test with the following TestKoperasi to check whether the class of the Member you created is as expected.

```
public class Test
{
    public static void main(String[] args)
    {
        Member donny = new Member("111333444", "Donny", 5000000);

        System.out.println("Member name: " + donny.getName());
        System.out.println("Loan Limit: " + donny.getLoanLimit());

        System.out.println("\n Borrow money 10.000.000...");
        donny.borrow(10000000);
        System.out.println("Current loan amount: " + donny.getLoanAmount());

        System.out.println("\n Borrow money 4.000.000...");
        donny.borrow(4000000);
        System.out.println("Current loan amount " + donny.getLoanAmount ());

        System.out.println("\nPay installments 1.000.000");
        donny.installments(1000000);
        System.out.println("Current loan amount: " + donny.getLoanAmount ());

        System.out.println("\nPay installments3.000.000");
        donny. installments (3000000);
        System.out.println("Current loan amount: " + donny.getLoanAmount ());
    }
}
```

ANSWER:

5. Modification of question no. 4 so that the minimum nominal installment is 10% of the current loan amount. If the installments are less than that, then the warning "Sorry, installments must be 10% of the loan amount".

ANSWER:

6. Modify the TestKoperasi class, so that the loan amount and installments can receive input from the console.

ANSWER: