Refactoring

We refactor code to improve the readability and maintainability of code. In this lab we will refactor some smelly code.

Smelly Code

Write the following code in a package name problem.

```
enum DegreeType {
    RegularThesis, RegularNonThesis
}
```

```
public class MastersProgram {
    private DegreeType dt;
    private String major;
    public MastersProgram(DegreeType type, String major){
        this.dt = type;
        this.major = major;
    }
    public boolean isEligible(float cgpa, String major){
        if(dt == DegreeType.RegularThesis){
            return this.major.equals(major) && cgpa >= 3.8;
        } else {
            return this.major.equals(major);
        }
    }
    public int getMarks(int theoryM, int thesisM, int pm){
        if(dt == DegreeType.RegularNonThesis){
            return theoryM + pm/2;
        } else {
            return theoryM + thesisM;
        }
    }
}
```

Now, write test code in MastersProramTest.java to test the above source code.

```
MastersProgramTest {
    @Test
    public void isEligibleThesis(){
        MastersProgram mp = new MastersProgram(DegreeType.RegularThesis, "swe");
        boolean isEligible = mp.isEligible(2, "swe");
        assertFalse(isEligible);
        isEligible = mp.isEligible(3.9f, "swe");
        assertTrue(isEligible);
    }

    @Test
    public void getMarksThesis(){
        MastersProgram mp = new MastersProgram(DegreeType.RegularNonThesis, "swe");
        int marks = mp.getMarks(80,60, 40);
        assertEquals(100, marks);
    }

    @Test
    public void isEligibleThesisDifferentMajor(){
        MastersProgram mp = new MastersProgram(DegreeType.RegularThesis, "cse");
        boolean isEligible = mp.isEligible(3.9f, "swe");
        assertFalse(isEligible);
    }
```

Three test cases are given. You have to write the remaining ones.

As you can see MastersProgram class is not open to future. Because it has casecading change effects to all conditional code in case of new requirement changes.

For example a new requirement Executive master program is introduced. Which has the following properties for eligibility.

```
(cgpa >= 3.75 || this.major.equals(major))
```

For marking, it only considers theoryMarks

```
theoryMarks
```

Refactored Code

Previously, we refactored the code using interface or abstract class. The thing is, we did not consider the test code. We were free to implement interface and abstract class. In this case, we have test code associated with actual code. From the definition of refactoring, it actually improves the current situation without breaking other code. Now, if you only employ previous technique it will break existing test code.

We have to implement **factory method pattern** in this scenario.

- In Factory pattern, we create object without exposing the creation logic to the client and refer to newly created object using a common interface.
- It promotes the loose-coupling

```
public class MastersProgram {
    private IMastersProgram type;

    public MastersProgram(DegreeType type, String major) {
        this.type = chooseMasterprogram (type,major);
    }

    public boolean isEligible(float cgpa, String major) {
        return type.isEligible(cgpa,major) ;
    }

    public int getMarks(int theoryMarks, int thesisMarks, int projectMarks) {
        return type.getMarks(theoryMarks, thesisMarks, projectMarks);
    }

    public IMastersProgram chooseMasterprogram (DegreeType type,String major) {
        IMastersProgram master;
        if(type.equals(DegreeType.RegularThesis)) {
            master = new RegularThesis(major);
        } else if (type.equals(DegreeType.RegularNonThesis)) {
            master = new RegularNonThesis(major);
        } else master = new Executive(major);

        return master;
    }
}
```

Now, add test code for executive master program

```
    @Test
    public void isEligibleExecutive(){
        MastersProgram mp = new MastersProgram(DegreeType.Executive, "swe");
        boolean isEligible = mp.isEligible(2, "cse");
        assertFalse(isEligible);
        isEligible = mp.isEligible(3.76f, "swe");
        assertTrue(isEligible);
    }
```

2nd problem

Below a smelly code is given.

```
enum VehicleType {
    SEDAN, MOTOR_BIKE, SEVEN_SEATER
}

public class Trip {
    private  VehicleType vehicleType;
    private  int distanceKM;
    private  int timeMinutes;
```

```
    private  int timeMinutes;
    private  int numberOfPassengers;

    public Trip(VehicleType vehicleType,
                int distanceKM,
                int timeMinutes,
                int numberOfPassengers) {
        this.vehicleType = vehicleType;
        this.distanceKM = distanceKM;
        this.timeMinutes = timeMinutes;
        this.numberOfPassengers = numberOfPassengers;
    }

    public int perHeadFare()
    {
        int fare = -1;
        switch (vehicleType) {
            case SEDAN:
                fare = (50 + distanceKM * 30 + timeMinutes * 2) / numberOfPassengers;
                break;
            case MOTOR_BIKE:
                fare = Math.max(25, distanceKM * 20) / numberOfPassengers;
                break;
            default:
                if (distanceKM < 10)
                    fare = 300 / numberOfPassengers;
                else
                    fare = distanceKM * 30 / numberOfPassengers;
                break;
        }

        return fare - (fare % 5);
    }

    public boolean canTakeTrip()
    {
        if (numberOfPassengers < 1)
            return false;

        switch (vehicleType)
        {
            case SEDAN:
                return numberOfPassengers <= 4 && distanceKM <= 25;
            case SEVEN_SEATER:
                return numberOfPassengers <= 7 && distanceKM >= 10;
            default:
                return numberOfPassengers <= 1 && distanceKM <= 10;
        }
    }
}
```

Task - write test code for the smelly code. - Refactor the given code without breaking the test code (i.e., applying factory method pattern) - Write additional test code for refactored code.