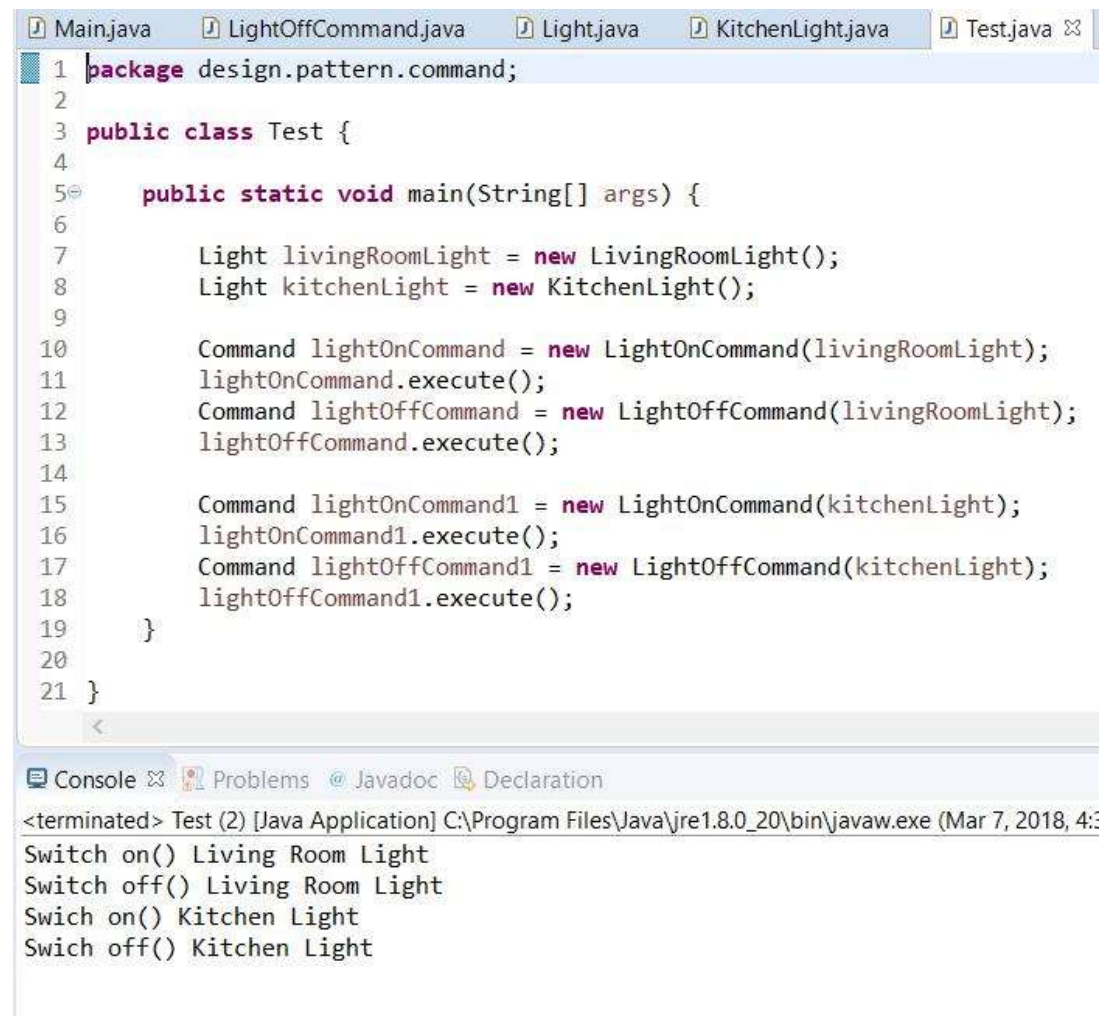## SE3030 – Software Architecture (SA)

Exercise 01 – Command Pattern

1. Create an Interface called **Command** with a method signature **execute()**
2. Create an Interface called **Light** with method signatures **on()** and **off()**
3. Implement 2 concrete classes named (**KitchenLight**, **LivingRoomLight**) and use **Light** interface with overriding methods **on()** and **off()** in each class.
4. Similarly use **Command interface** and implement 2 concrete classes named (**LightOnCommand**, **LightOffCommand**) and override **execute()** methods in each class.
5. Create **Test** class as below to check each light **on** and **off** commands with respect to the provided location.
6. Method **execute()** will run the given object for command class as below. Follow all above steps and execute relevant method.
7. Run this **Test** class and check the output should be as below.

```
Main.java    LightOffCommand.java    Light.java    KitchenLight.java    Test.java

 1  package design.pattern.command;
 2
 3  public class Test {
 4
 5      public static void main(String[] args) {
 6
 7          Light livingRoomLight = new LivingRoomLight();
 8          Light kitchenLight = new KitchenLight();
 9
10          Command lightOnCommand = new LightOnCommand(livingRoomLight);
11          lightOnCommand.execute();
12          Command lightOffCommand = new LightOffCommand(livingRoomLight);
13          lightOffCommand.execute();
14
15          Command lightOnCommand1 = new LightOnCommand(kitchenLight);
16          lightOnCommand1.execute();
17          Command lightOffCommand1 = new LightOffCommand(kitchenLight);
18          lightOffCommand1.execute();
19      }
20
21  }
```

```
Console    Problems  @ Javadoc  Declaration
<terminated> Test (2) [Java Application] C:\Program Files\Java\jre1.8.0_20\bin\javaw.exe (Mar 7, 2018, 4:3
Switch on() Living Room Light
Switch off() Living Room Light
Swich on() Kitchen Light
Swich off() Kitchen Light
```

SE3030 – Software Architecture (SA)

Exercise 02 – Template-method pattern

1. Create an abstract class called **Beverages** and extends that class using **Tea** and **Coffee** concrete classes
2. Within **Tea** and **Coffee** classes override both abstract methods **addCondiments()** and **brew()**
3. Now modify the **Beverage** class to implement **boilWater()** and **pourInCup()** methods. As per the below.

```java
abstract void brew();

abstract void addCondiments();

void boilWater(){
    System.out.println("Boiling water.");
}

void pourInCup(){
    System.out.println("Pour into cup.");
}
```

4. Now you should impose the order of execution of these methods as below. This order of execution **should not be changed** implicitly or explicitly in any of these sub classes and it should work as life cycle methods.
5. Your modification should satisfy **step 4**
6. Now Implement **Test class** to test **above template method pattern** and you should be able to display the output below.

```java
 3  public class TestTemplateMethod {
 4
 5      static Beverage beverage = null;
 6
 7      public static void main(String[] args) {
 8
 9          Beverage tea = new Tea();
10          tea.prepareRecepie();
11          System.out.println("===============================");
12          Beverage coffie = new Coffie();
13          coffie.prepareRecepie();
14      }
15  }
16
```

```
Console ☒  Problems  @ Javadoc  Declaration
<terminated> TestTemplateMethod [Java Application] C:\Program Files\Java\jre1.8.0_20\b
Boiling water.
Steeping the Tea.
Adding Lemon.
Pour into cup.
===============================
Boiling water.
Stripping coffie through filter.
Add suger and milk.
Pour into cup.
```

SE3030 – Software Architecture (SA)

Exercise 03 – Builder Pattern

1. Create a **Query** class with SELECT, FROM WHERE and ORDER BY as properties
   a. Implement a method to print the complete query

2. Create a **QueryBuilder** class with a property to hold a **Query** object
   a. Create the Query object inside the constructor of **QueryBuilder**
   b. Implement methods to set SELECT, FROM WHERE and ORDER to the Query object
   c. Each method should return a **QueryBuilder** object

3. Implement a method called **build()** in **QueryBuilder** that returns the **Query** object
   a. Check if the Query contains at least SELECT and FROM properties, if not it is not a valid query and prevent building the query by throwing an exception

4. Create a Test class to test out the pattern functionality

```java
Query query1 = new QueryBuilder().select("name").from("student").build();
System.out.println(query1.toString()); // A valid query will be constructed

Query query2 = new QueryBuilder().select("name").from("student").where("name =
'Name1'").build();
    System.out.println(query2.toString()); // A valid query will be constructed

Query query3 = new QueryBuilder().select("name").where("name = 'Name1'").build();
System.out.println(query3.toString()); // Will throw an exception
```

5. You should display the following outputs. If you missed key word of the query, you should throw an exception as below.

```java
3 public class Main {
4
5⊖   public static void main(String[] args) {
6
7       Query query1 = new QueryBuilder().select("name").from("student").build();
8       System.out.println(query1.toString()); // A valid query will be constructed
9
10      Query query2 = new QueryBuilder().select("name").from("student").where("name = 'Name1'").build();
11      System.out.println(query2.toString()); // A valid query will be constructed
12
13      Query query3 = new QueryBuilder().select("name").where("name = 'Name1'").build();
14      System.out.println(query3.toString()); // Will throw an exception
15  }
16 }
```

```
🖳 Console ⊠  📋 Problems  @ Javadoc  🔍 Declaration
<terminated> Main (2) [Java Application] C:\Program Files\Java\jre1.8.0_20\bin\javaw.exe (Mar 7, 2018, 5:31:38 PM)
SELECT name FROM student
SELECT name FROM student WHERE name = 'Name1'
Exception in thread "main" java.lang.IllegalStateException: Query must have a FROM
        at design.pattern.builder.QueryBuilder.build(QueryBuilder.java:41)
        at design.pattern.builder.Main.main(Main.java:15)
```

SE3030 – Software Architecture (SA)

Exercise 04 – Abstract Factory Pattern

1. Create an Interface called **Shape** with a method signature **draw()**
2. Implement 3-4 Concrete Classes of Shape
   a. Create classes for **Square**, **Circle**, **Triangle**, **Rectangle**, etc.
   b. Implement the **draw()** method
      e.g. Print the name of the shape inside the draw method of each class

3. Create a **ShapeFactory** class
   a. Add a method called **getShape()** that accepts a **String** as a parameter and returns a **Shape**
   b. Implement **getShape()** method to create the concrete shapes
      e.g. Check if the parameter is "SQUARE" and create an instance of Square class and return it
4. Create a Test class to test out the pattern functionality

```
ShapeFactory shapeFactory = new ShapeFactory();
// get an object of Circle and call its draw method.
Shape shape1 = shapeFactory.getShape("SQUARE");
// call draw method of Circle
shape1.draw();
```

5. Complete the rest of the parts in design You should display the following output.
6. Create a Test class to test out the pattern functionality as below

```java
3  public class Main {
4
5      public static void main(String[] args) {
6
7          ShapeFactory shapeFactory = new ShapeFactory();
8          // get an object of Circle and call its draw method.
9          Shape circle = shapeFactory.getShape("CIRCLE");
10         // call draw method of Circle
11         circle.draw();
12         // get an object of Rectangle and call its draw method.
13         Shape rectangle = shapeFactory.getShape("RECTANGLE");
14         // call draw method of Rectangle
15         rectangle.draw();
16         // get an object of Square and call its draw method.
17         Shape square = shapeFactory.getShape("SQUARE");
18         // call draw method of circle
19         square.draw();
20     }
21 }
```

SE3030 – Software Architecture (SA)

```
Console ⊠  Problems  Javadoc  Declaration
<terminated> Main (1) [Java Application] C:\Program Files\Jav
Inside Circle::draw() method.
    O
Inside Rectangle::draw() method.
##################
#                #
#                #
#                #
##################
Inside Square::draw() method.
#########
#       #
#       #
#       #
#########
```

============================END OF THE LAB===================================