

Lesson 04 Demo 04

Creating and Using Design Patterns Factory and State

Objective: To implement the Factory and State design patterns

Tools required: Eclipse IDE

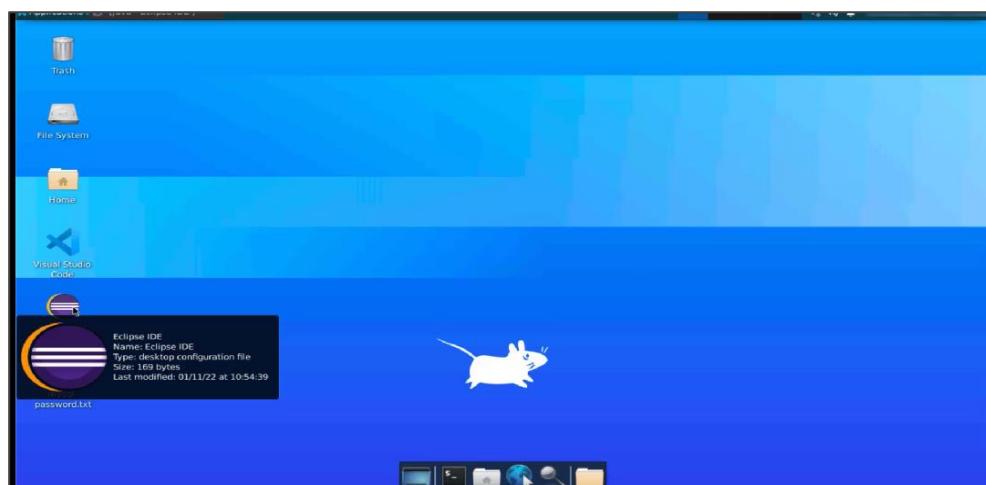
Prerequisite: None

Steps to be followed:

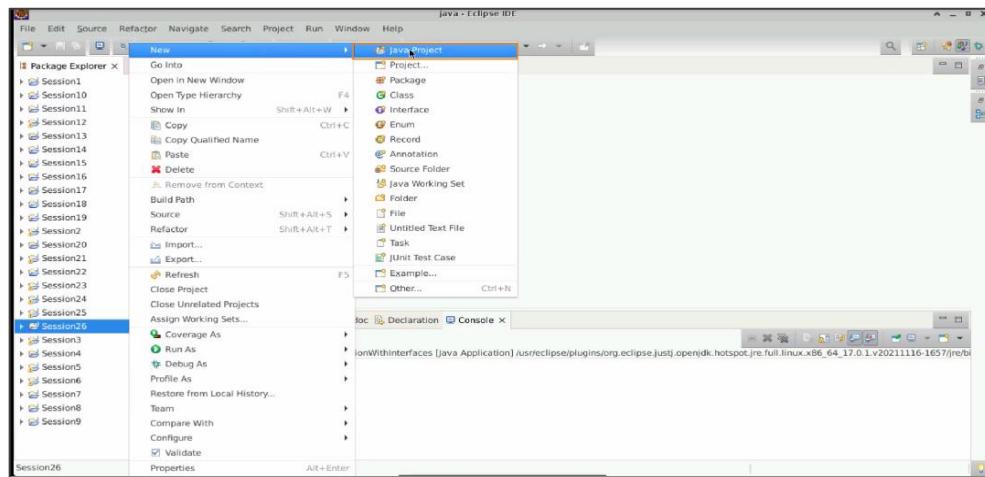
1. Implement and create Factory Design Pattern class and Plan interface
2. Create class Plan3G
3. Create class Plan4G and Plan5G
4. Create class Plan Factory
5. Implement the methods of the interface and classes
6. Create State Design Patterns

Step 1: Implement and create Factory Design Pattern class and Plan interface

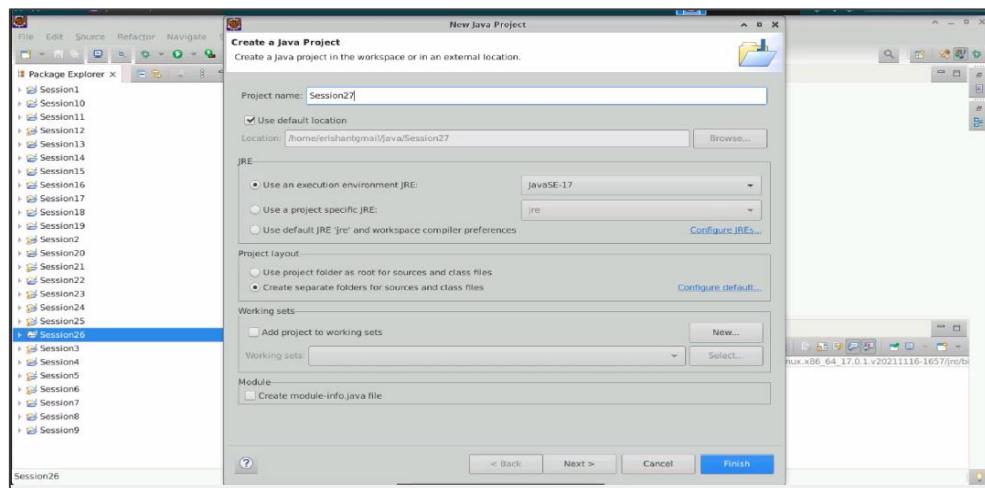
1.1 Open the Eclipse IDE.



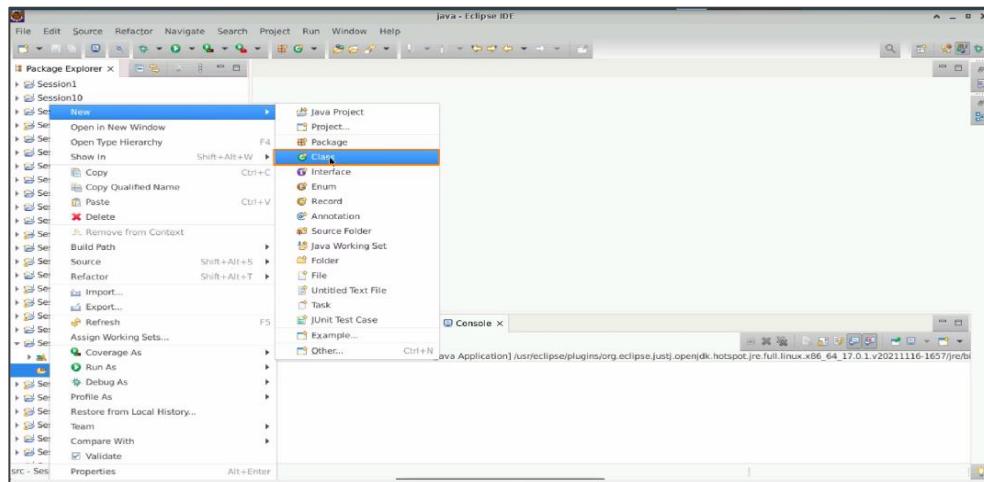
1.2 Select File, then New, and then Java Project.



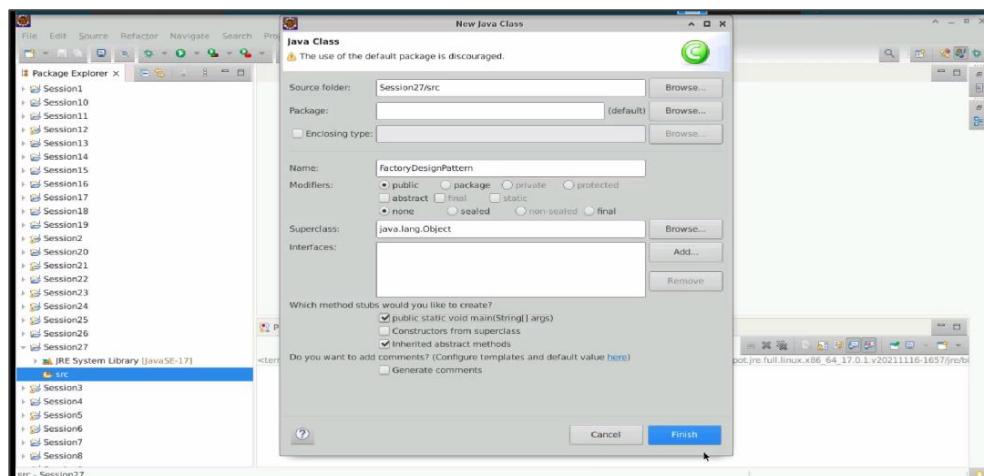
1.3 Name the project as “Session27” and select Finish.



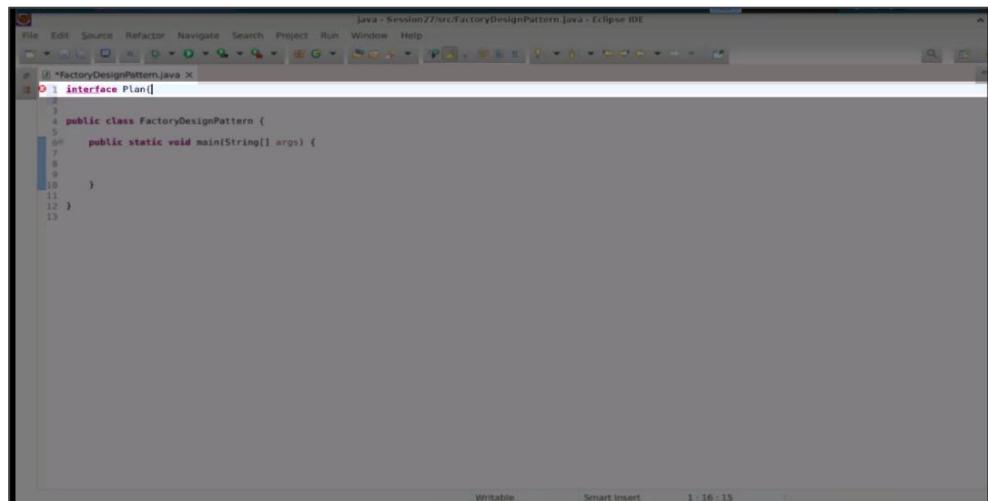
1.4 Right click on the **src** folder under the project name, select **New**, and then select **Class**.



1.5 Name the class as **FactoryDesignPattern** and select the option **public static void main(String[] args)**. Select **Finish**. This will create a java class with a **main()** method.



1.6 In the class **FactoryDesignPattern**, create an interface **Plan** by writing the **interface Plan**.



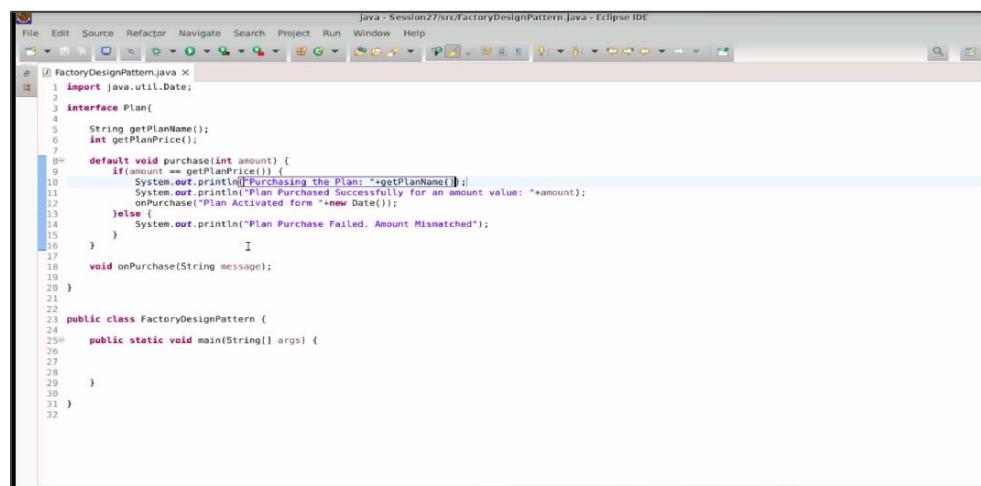
The screenshot shows the Eclipse IDE interface with a Java file named "FactoryDesignPattern.java" open. The code contains the following:

```
1 package com.simplilearn.FactoryDesignPattern;
2
3 public class FactoryDesignPattern {
4     public static void main(String[] args) {
5         System.out.println("Hello World!");
6     }
7 }
```

The interface definition is present but incomplete.

1.7 Create four methods inside the interface. The methods are:

- **String getPlanName()** : This method will return the plan name as a string.
- **int getPlanPrice()** : This method will return the price of the selected plan as an integer value.
- **default void purchase(int amount)** : In the purchasing technique, you will first determine whether the amount is equal to determine the plan price. If your amount equals the plan price only then will you complete a transaction, regardless of the plan price and print the **Plan purchased successfully for the amount** and mention the amount using the **System.out.println()**. Else, print **Plan purchase failed. Amount Mismatched** in the **else** section.
- **void onPurchase(String message)** : This method will be executed in the **purchase()** method and will display a string message **Plan activated from** and the current date and time.



```

File Edit Source Refactor Navigate Search Project Run Window Help
FactoryDesignPattern.java X
1 import java.util.Date;
2
3 interface Plan{
4
5     String getPlanName();
6     int getPlanPrice();
7
8     default void purchase(int amount) {
9         if(amount == getPlanPrice()) {
10             System.out.println("Purchasing the Plan: "+getPlanName());
11             System.out.println("Plan Purchased Successfully for an amount value: "+amount);
12             onPurchase("Plan Activated from "+new Date());
13         } else {
14             System.out.println("Plan Purchase Failed. Amount Mismatched");
15         }
16     }
17     void onPurchase(String message);
18 }
19
20
21
22
23 public class FactoryDesignPattern {
24
25     public static void main(String[] args) {
26
27
28     }
29
30
31 }
32

```

Step 2: Create class Plan3G

- 2.1 Create a new class **Plan3G** which implements **Plan** interface as **class Plan3G implements Plan**.

```

1 *FactoryDesignPattern.java *
2 import java.util.Date;
3
4 interface Plan{
5     String getPlanName();
6     int getPlanPrice();
7
8     default void purchase(int amount) {
9         if(amount == getPlanPrice()) {
10             System.out.println("Purchasing the Plan: "+getPlanName());
11             System.out.println("Plan Purchased Successfully for an amount value: "+amount);
12             onPurchase("Plan Activated form "+new Date());
13         } else {
14             System.out.println("Plan Purchase Failed. Amount Mismatched");
15         }
16     }
17     void onPurchase(String message);
18 }
19
20
21 class Plan3G implements Plan{
22 }
23
24
25
26
27 public class FactoryDesignPattern {
28
29     public static void main(String[] args) {
30
31
32
33
34
35 }
36

```

- 2.2 Click on the error bulb of **Plan3G** and select the option **Add unimplemented methods** to implement the methods of the plan interface.

```

1 *FactoryDesignPattern.java *
2 import java.util.Date;
3
4 interface Plan{
5     String getPlanName();
6     int getPlanPrice();
7
8     default void purchase(int amount) {
9         if(amount == getPlanPrice()) {
10             System.out.println("Purchasing the Plan: "+getPlanName());
11             System.out.println("Plan Purchased Successfully for an amount value: "+amount);
12             onPurchase("Plan Activated from "+new Date());
13         } else {
14             System.out.println("Plan Purchase Failed. Amount Mismatched");
15         }
16     }
17     void onPurchase(String message);
18 }
19
20
21 class Plan3G implements Plan{
22     <!-- Add unimplemented methods -->
23     <!-- Make type 'Plan3G' abstract -->
24     <!-- Rename in file (Ctrl+2 R) -->
25     <!-- Rename in workspace -->
26
27 public
28
29     pu
30
31
32
33
34
35 }
36

```

The type Plan3G must implement the inherited abstract method Plan.getPlanName()

2.3 In getPlanName() method, write return “Unlimited plan3G”.

```

File Edit Source Refactor Navigate Search Project Run Window Help
Java - Session27/src/FactoryDesignPattern.java - Eclipse IDE
1 *FactoryDesignPattern.java *
2  * Author: Gururammeeti
3  * Date: 2023-07-10
4  * Time: 10:57 AM
5  * Version: 1.0
6  * 
7  int getPlanPrice();
8 
9 default void purchase(int amount) {
10    if(amount == getPlanPrice()) {
11      System.out.println("Purchasing the Plan: "+getPlanName());
12      System.out.println("Plan Purchased Successfully for an amount value: "+amount);
13      onPurchase("Plan Activated form "+new Date());
14    } else {
15      System.out.println("Plan Purchase Failed. Amount Mismatched");
16    }
17 }
18 void onPurchase(String message);
19 
20 }
21 
22 class Plan3G implements Plan{
23 
24    @Override
25    public String getPlanName() {
26        return "Unlimited Plan3G";
27    }
28 
29    @Override
30    public int getPlanPrice() {
31        // TODO Auto-generated method stub
32        return 0;
33    }
34 
35    @Override
36    public void onPurchase(String message) {
37        // TODO Auto-generated method stub
38    }
39 }
40 
41 }
42

```

Writable Smart Insert 26 : 27 : 570

2.4 In the getPlanPrice() method, write return 250.

```

File Edit Source Refactor Navigate Search Project Run Window Help
Java - Session27/src/FactoryDesignPattern.java - Eclipse IDE
1 *FactoryDesignPattern.java *
2  * Author: Gururammeeti
3  * Date: 2023-07-10
4  * Time: 10:57 AM
5  * Version: 1.0
6  * 
7  int getPlanPrice();
8 
9 default void purchase(int amount) {
10    if(amount == getPlanPrice()) {
11      System.out.println("Purchasing the Plan: "+getPlanName());
12      System.out.println("Plan Purchased Successfully for an amount value: "+amount);
13      onPurchase("Plan Activated form "+new Date());
14    } else {
15      System.out.println("Plan Purchase Failed. Amount Mismatched");
16    }
17 }
18 void onPurchase(String message);
19 
20 }
21 
22 class Plan3G implements Plan{
23 
24    @Override
25    public String getPlanName() {
26        return "Unlimited Plan3G";
27    }
28 
29    @Override
30    public int getPlanPrice() {
31        return 250;
32    }
33 
34    @Override
35    public void onPurchase(String message) {
36        // TODO Auto-generated method stub
37    }
38 }
39 
40 }
41 
42 public class FactoryDesignPattern {

```

Writable Smart Insert 31 : 19 : 635

2.5 In the `onPurchase()` method, write `System.out.println("[MESSAGE] "+message)`

```

1 *FactoryDesignPattern.java *
2
3 package com.simplilearn;
4
5 public interface Plan {
6     int getPlanPrice();
7
8     default void purchase(int amount) {
9         if(amount == getPlanPrice()) {
10             System.out.println("Purchasing the Plan: "+getPlanName());
11             System.out.println("Plan Purchased Successfully for an amount value: "+amount);
12             onPurchase("Plan Activated form "+new Date());
13         } else {
14             System.out.println("Plan Purchase Failed. Amount Mismatched");
15         }
16     }
17
18     void onPurchase(String message);
19 }
20
21 class Plan3G implements Plan{
22
23     @Override
24     public String getPlanName() {
25         return "Unlimited Plan3G";
26     }
27
28     @Override
29     public int getPlanPrice() {
30         return 250;
31     }
32
33     @Override
34     public void onPurchase(String message) {
35         System.out.println("[MESSAGE] "+message);
36     }
37
38 }
39
40
41 public class FactoryDesignPattern {
42
43 }

```

2.6 Here, we have `Plan3G` implementing one of the plans of the interface.

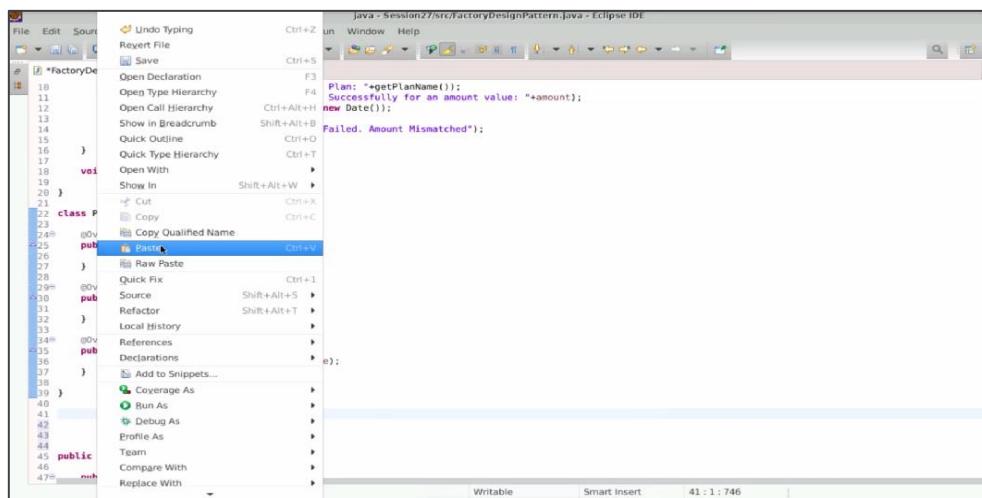
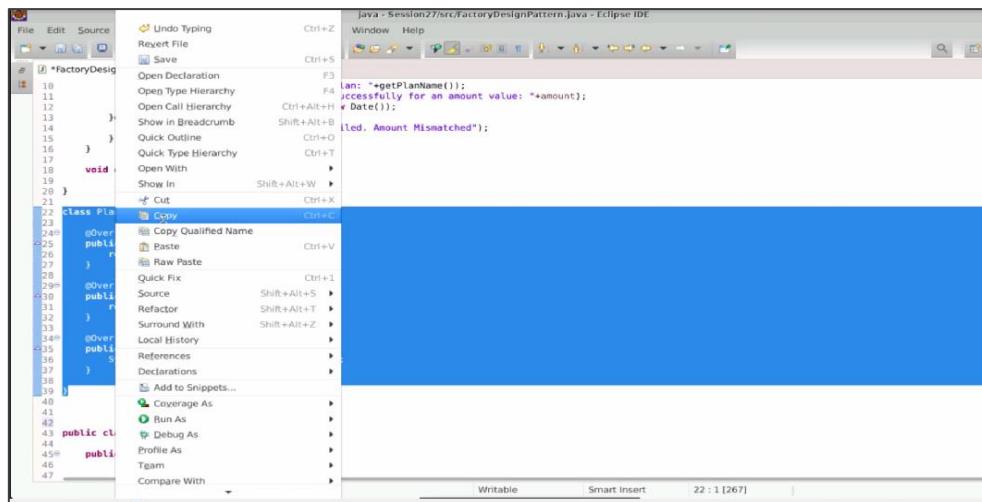
```

10     System.out.println("Purchasing the Plan: "+getPlanName());
11     System.out.println("Plan Purchased Successfully for an amount value: "+amount);
12     onPurchase("Plan Activated form "+new Date());
13 } else {
14     System.out.println("Plan Purchase Failed. Amount Mismatched");
15 }
16
17 void onPurchase(String message);
18 }
19
20
21 class Plan3G implements Plan{
22
23     @Override
24     public String getPlanName() {
25         return "Unlimited Plan3G";
26     }
27
28     @Override
29     public int getPlanPrice() {
30         return 250;
31     }
32
33     @Override
34     public void onPurchase(String message) {
35         System.out.println("[MESSAGE] "+message);
36     }
37
38 }
39
40
41 public class FactoryDesignPattern {
42
43     public static void main(String[] args) {
44
45
46
47 }

```

Step 3: Create class Plan4G and Plan5G

3.1 Create class **Plan4G** by copying and pasting class **Plan3G**.

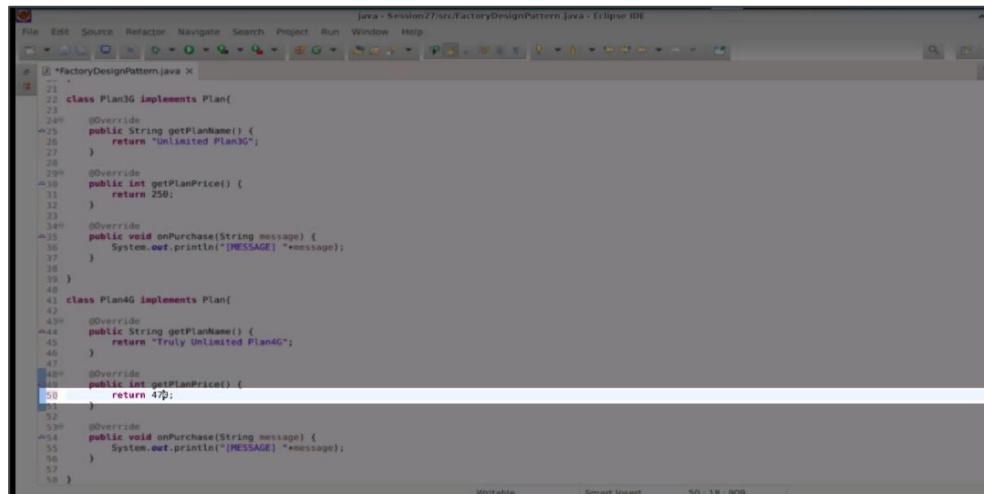


```
# *FactoryDesignPattern.java X
1
2
3 class Plan3G implements Plan{
4
5     @Override
6     public String getPlanName() {
7         return "Unlimited Plan3G";
8     }
9
10    @Override
11    public int getPlanPrice() {
12        return 250;
13    }
14
15    @Override
16    public void onPurchase(String message) {
17        System.out.println("[MESSAGE] "+message);
18    }
19
20 }
21
22
23 class Plan4G implements Plan{
24
25     @Override
26     public String getPlanName() {
27         return "Unlimited Plan4G";
28     }
29
30    @Override
31    public int getPlanPrice() {
32        return 250;
33    }
34
35    @Override
36    public void onPurchase(String message) {
37        System.out.println("[MESSAGE] "+message);
38    }
39
40 }
41
42
43 class Plan5G implements Plan{
44
45     @Override
46     public String getPlanName() {
47         return "Unlimited Plan5G";
48     }
49
50    @Override
51    public int getPlanPrice() {
52        return 250;
53    }
54
55    @Override
56    public void onPurchase(String message) {
57        System.out.println("[MESSAGE] "+message);
58    }
59
60 }
```

3.2 In `getPlanName()` method, write `return "Truly Unlimited plan4G";`

```
File Edit Source Refactor Navigate Search Project Run Window Help
[*] *FactoryDesignPattern.java X
1 package com.factorydesignpattern;
2
3 public class Plan {
4     String name;
5     int price;
6
7     public String getName() {
8         return name;
9     }
10
11     public void setName(String name) {
12         this.name = name;
13     }
14
15     public int getPrice() {
16         return price;
17     }
18
19     public void setPrice(int price) {
20         this.price = price;
21     }
22 }
23
24 class Plan3 implements Plan{
25
26     @Override
27     public String getPlanName() {
28         return "Unlimited Plan3";
29     }
30
31     @Override
32     public int getPlanPrice() {
33         return 250;
34     }
35
36     @Override
37     public void onPurchase(String message) {
38         System.out.println("[MESSAGE] "+message);
39     }
40 }
41
42 class Plan4 implements Plan{
43
44     @Override
45     public String getPlanName() {
46         return "Unlimited Plan4";
47     }
48
49     @Override
50     public int getPlanPrice() {
51         return 250;
52     }
53
54     @Override
55     public void onPurchase(String message) {
56         System.out.println("[MESSAGE] "+message);
57     }
58 }
```

3.3 In the `getPlanPrice()` method, write **return 470**.

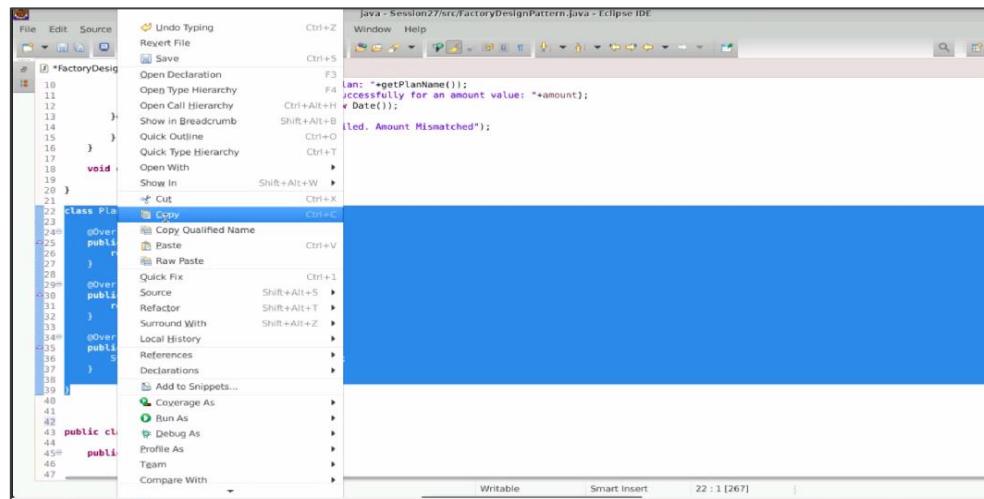


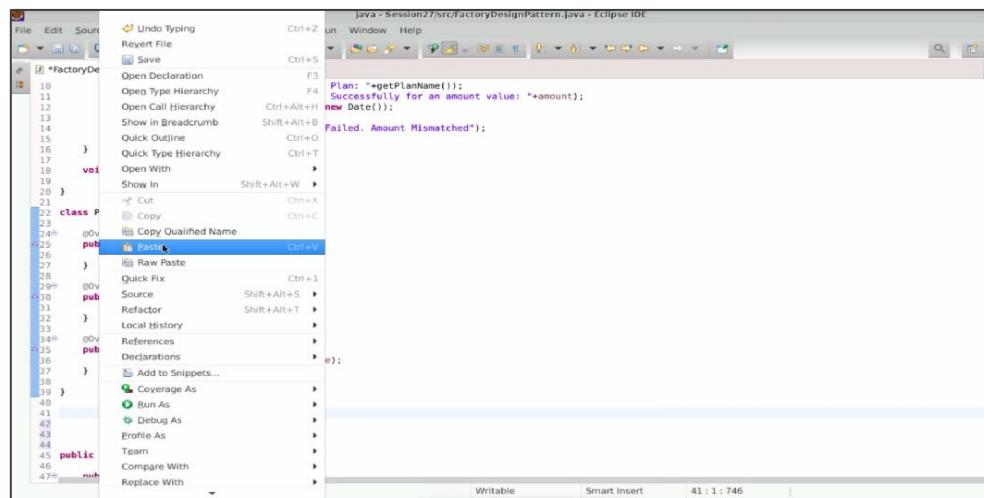
```

1 *FactoryDesignPattern.java *
21
22     class Plan3G implements Plan{
23
24         @Override
25         public String getPlanName() {
26             return "Unlimited Plan3G";
27         }
28
29         @Override
30         public int getPlanPrice() {
31             return 250;
32         }
33
34         @Override
35         public void onPurchase(String message) {
36             System.out.println("[MESSAGE] "+message);
37         }
38
39     }
40
41     class Plan4G implements Plan{
42
43         @Override
44         public String getPlanName() {
45             return "Truly Unlimited Plan4G";
46         }
47
48         @Override
49         public int getPlanPrice() {
50             return 470;
51         }
52
53         @Override
54         public void onPurchase(String message) {
55             System.out.println("[MESSAGE] "+message);
56         }
57
58     }

```

3.4 Create class **Plan5G** by copying and pasting class **Plan3G**.





```
File Edit Source Refactor Navigate Search Project Run Window Help
Java - Session2/src/FactoryDesignPattern.java - Eclipse IDE
[ *FactoryDesignPattern.java X
10
11     class Plan4G implements Plan{
12
13         @Override
14         public String getPlanName() {
15             return "Truly Unlimited Plan4G";
16         }
17
18         @Override
19         public int getPlanPrice() {
20             return 470;
21         }
22
23         @Override
24         public void onPurchase(String message) {
25             System.out.println("[MESSAGE] "+message);
26         }
27
28     }
29
30     class Plan5G implements Plan{
31
32         @Override
33         public String getPlanName() {
34             return "Unlimited Plan3G";
35         }
36
37         @Override
38         public int getPlanPrice() {
39             return 250;
40         }
41
42         @Override
43         public void onPurchase(String message) {
44             System.out.println("[MESSAGE] "+message);
45         }
46
47     }
48 }
```

3.5 In `getPlanName()` method, write `return Nolimits plan5G`.

```
File Edit Source Refactor Navigate Search Project Run Window Help
Java - Session2/src/FactoryDesignPattern.java - Eclipse IDE
[ *FactoryDesignPattern.java X
10
11     class Plan4G implements Plan{
12
13         @Override
14         public String getPlanName() {
15             return "Truly Unlimited Plan4G";
16         }
17
18         @Override
19         public int getPlanPrice() {
20             return 470;
21         }
22
23         @Override
24         public void onPurchase(String message) {
25             System.out.println("[MESSAGE] "+message);
26         }
27
28     }
29
30     class Plan5G implements Plan{
31
32         @Override
33         public String getPlanName() {
34             return "Nolimits plan5G";
35         }
36
37         @Override
38         public int getPlanPrice() {
39             return 250;
40         }
41
42         @Override
43         public void onPurchase(String message) {
44             System.out.println("[MESSAGE] "+message);
45         }
46
47     }
48 }
```

3.6 In the `getPlanPrice()` method, write `return 800`.

```

File Edit Source Refactor Navigate Search Project Run Window Help
Java - Session27/src/FactoryDesignPattern.java - Eclipse IDE
1 *FactoryDesignPattern.java *
2
3 class Plan4G implements Plan{
4
5     @Override
6     public String getPlanName() {
7         return "Truly Unlimited Plan4G";
8     }
9
10    @Override
11    public int getPlanPrice() {
12        return 470;
13    }
14
15    @Override
16    public void onPurchase(String message) {
17        System.out.println("[MESSAGE] "+message);
18    }
19
20
21 class Plan5G implements Plan{
22
23     @Override
24     public String getPlanName() {
25         return "NoLimits Plan5G";
26     }
27
28     @Override
29     public int getPlanPrice() {
30         return 800;
31     }
32
33
34    @Override
35    public void onPurchase(String message) {
36        System.out.println("[MESSAGE] "+message);
37    }
38
39 }
40
41 class Plan4G implements Plan{
42
43     @Override
44     public String getPlanName() {
45         return "Truly Unlimited Plan4G";
46     }
47
48     @Override
49     public int getPlanPrice() {
50         return 470;
51     }
52
53     @Override
54     public void onPurchase(String message) {
55         System.out.println("[MESSAGE] "+message);
56     }
57
58 }
59
60 class PlanFactory{
61
62     class Plan5G implements Plan{
63
64         @Override
65         public String getPlanName() {
66
67
68
69     }
70
71
72 }
73
74
75 }
76
77

```

Step 4: Create class PlanFactory

4.1 Create another class **PlanFactory** by writing the **class PlanFactory** to implement the factory design pattern.

```

File Edit Source Refactor Navigate Search Project Run Window Help
Java - Session27/src/FactoryDesignPattern.java - Eclipse IDE
1 *FactoryDesignPattern.java *
2
3 class Plan4G implements Plan{
4
5     @Override
6     public int getPlanPrice() {
7         return 250;
8     }
9
10
11    @Override
12    public void onPurchase(String message) {
13        System.out.println("[MESSAGE] "+message);
14    }
15
16 }
17
18 class Plan4G implements Plan{
19
20     @Override
21     public String getPlanName() {
22         return "Truly Unlimited Plan4G";
23     }
24
25     @Override
26     public int getPlanPrice() {
27         return 470;
28     }
29
30     @Override
31     public void onPurchase(String message) {
32         System.out.println("[MESSAGE] "+message);
33     }
34
35 }
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
59 class PlanFactory{
60
61
62     class Plan5G implements Plan{
63
64
65
66
67
68
69
69     }
70
71
72
73
74
75
76
77

```

4.2 Create a public method which returns a **plan**. Code it as **public Plan getPlan(int typeOfPlan)** which takes typeOfPlan as input.

The screenshot shows the Eclipse IDE interface with the title bar "java - Session27/src/factorydesignpattern.java - Eclipse IDE". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help. The toolbar has icons for New, Open, Save, Cut, Copy, Paste, Find, and others. The code editor displays the following Java code:

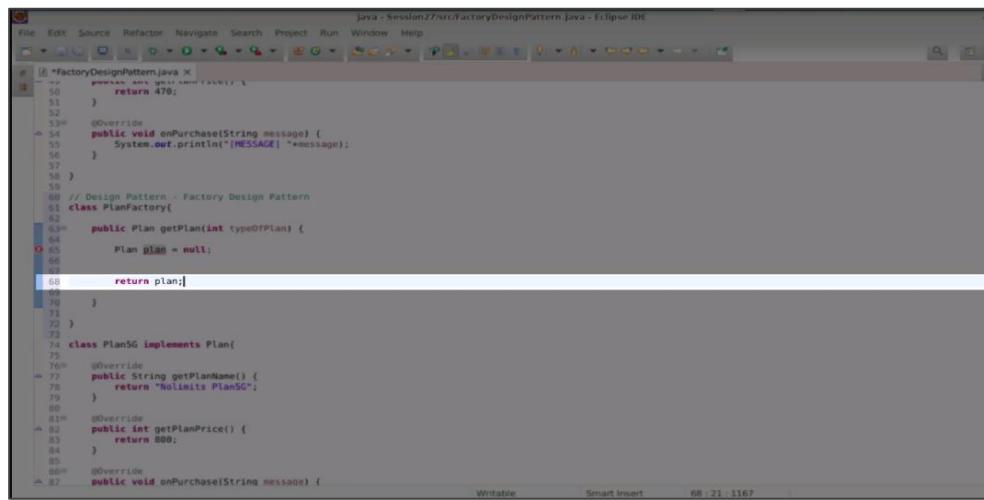
```
* *FactoryDesignPattern.java *
50     return 470;
51 }
52
53 @Override
54 public void onPurchase(String message) {
55     System.out.println("[MESSAGE] "+message);
56 }
57
58 // Design Pattern - Factory Design Pattern
59 class PlanFactory{
60
61     public Plan getPlan(int typeOfPlan){
62
63     }
64
65     class PlanSG implements Plan{
66
67         @Override
68         public String getPlanName(){
69             return "NoLimits PlanSG";
70         }
71
72         @Override
73         public int getPlanPrice() {
74             return 800;
75         }
76
77         @Override
78         public void onPurchase(String message) {
79             System.out.println("[MESSAGE] "+message);
80         }
81
82     }
83
84 }
85
86
87
```

A red box highlights the closing brace at line 87. A status bar at the bottom left says "Syntax error on token "}", { expected after this token". The status bar also shows "Writable" and "Smart Insert" with a timestamp of "63 : 39 : 1120".

4.3 Create a reference variable of plan which is by default **null**.

```
File Edit Source Refactor Navigate Search Project Run Window Help
# *FactoryDesignPattern.java X
  powers and generate taxes
  50     return 478;
  51   }
  52 }
  53
  54 @Override
  55 public void onPurchase(String message) {
  56     System.out.println("[MESSAGE] "+message);
  57 }
  58 }
  59
  60 // Design Pattern - Factory Design Pattern
  61 class PlanFactory{
  62
  63     public Plan getPlan(int typeOfPlan) {
  64
  65         Plan plan = null;
  66
  67     }
  68 }
  69
  70 class PlanSG implements Plan{
  71
  72     @Override
  73     public String getPlanName() {
  74         return "NoLimits PlanSG";
  75     }
  76
  77     @Override
  78     public int getPlanPrice() {
  79         return 800;
  80     }
  81
  82     @Override
  83     public void onPurchase(String message) {
  84         System.out.println("[MESSAGE] "+message);
  85     }
  86 }
```

4.4 Return the reference variable plan.

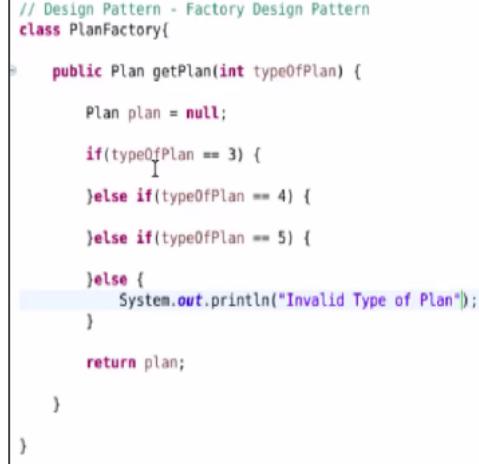


The screenshot shows the Eclipse IDE interface with a Java file named 'FactoryDesignPattern.java' open. The code implements the Factory Design Pattern. It includes a PlanFactory class that returns a Plan object based on its type (3G, 4G, or 5G). Each plan type has its own specific implementation (Plan3G, Plan4G, Plan5G) that overrides the getPlanName() and getPlanPrice() methods. The PlanFactory class also overrides the onPurchase() method to print a message to the console.

```

 50     return 470;
 51   }
 52 
 53   @Override
 54   public void onPurchase(String message) {
 55     System.out.println("MESSAGE");
 56   }
 57 }
 58 
 59 // Design Pattern - Factory Design Pattern
 60 class PlanFactory{
 61 
 62   public Plan getPlan(int typeOfPlan) {
 63     Plan plan = null;
 64 
 65     return plan;
 66   }
 67 }
 68 
 69 class PlanSG implements Plan{
 70 
 71   @Override
 72   public String getPlanName() {
 73     return "NoLimits PlanSG";
 74   }
 75 
 76   @Override
 77   public int getPlanPrice() {
 78     return 800;
 79   }
 80 
 81   @Override
 82   public void onPurchase(String message) {
 83 
 84   }
 85 }
 86 
 87 public void onPurchase(String message) {
 88 
 89   System.out.println("MESSAGE");
 90 }
 91 }
 92 
```

4.5 If the type of plan is equivalent to 3, you are attempting to make reference to a 3G plan. And, "4" refers to the 4G plan. The type of plan is equivalent to five then denotes a 5G plan, and in any other situation, you can inform the user that their plan is invalid.



The code editor displays the same Java code as the previous screenshot, but with a different section highlighted. A conditional block within the getPlan() method checks the typeOfPlan variable. If it equals 3, it prints '3G'. If it equals 4, it prints '4G'. If it equals 5, it prints '5G'. For any other value, it prints 'Invalid Type of Plan'.

```

// Design Pattern - Factory Design Pattern
class PlanFactory{

  public Plan getPlan(int typeOfPlan) {

    Plan plan = null;

    if(typeOfPlan == 3) {
      System.out.println("3G");
    } else if(typeOfPlan == 4) {
      System.out.println("4G");
    } else if(typeOfPlan == 5) {
      System.out.println("5G");
    } else {
      System.out.println("Invalid Type of Plan");
    }

    return plan;
  }
}
  
```

4.6 According to the selected plan, create the reference objects of respective plans.

This is called runtime polymorphism; the same reference variable plan can refer to any plan object. This is the polymorphic behavior with the interfaces.

```
// Design Pattern - Factory Design Pattern
class PlanFactory{

    public Plan getPlan(int typeOfPlan) {

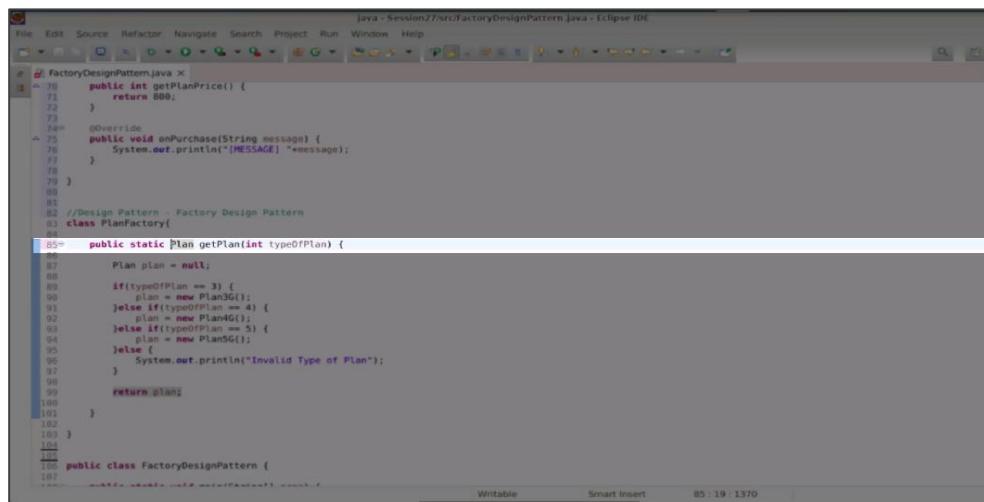
        Plan plan = null;

        if(typeOfPlan == 3) {
            plan = new Plan3G();
        }else if(typeOfPlan == 4) {
            plan = new Plan4G();
        }else if(typeOfPlan == 5) {
            plan = new Plan5G();
        }else {
            System.out.println("Invalid Type of Plan");
        }

        return plan;
    }
}
```

Step 5: Implement the methods of the interface and classes

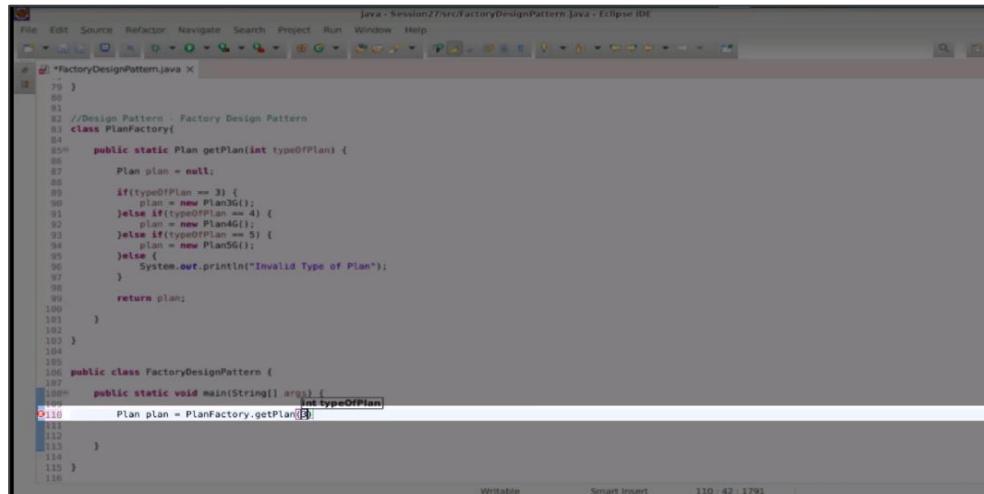
5.1 Make the **getPlan** method of class **PlanFactory** as **static** so that we need not create the object to implement this method.



```
File Edit Source Refactor Navigate Search Project Run Window Help
Java - Session27/src/FactoryDesignPattern.java - Eclipse IDE

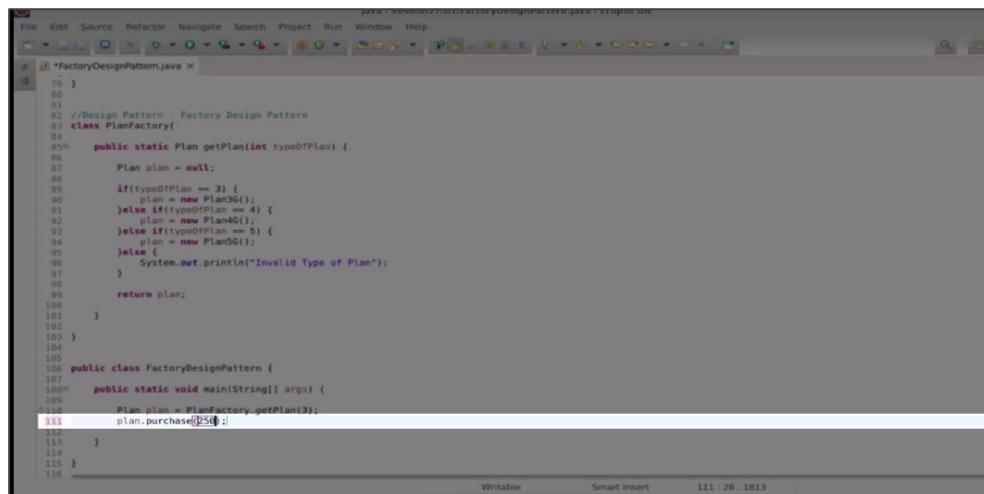
70     public int getPlanPrice() {
71         return 800;
72     }
73
74     @Override
75     public void onPurchase(String message) {
76         System.out.println("[MESSAGE] "+message);
77     }
78 }
79
80
81 //Design Pattern - Factory Design Pattern
82 class PlanFactory{
83
84     public static Plan getPlan(int typeOfPlan) {
85
86         Plan plan = null;
87
88         if(typeOfPlan == 3) {
89             plan = new Plan3G();
90         }else if(typeOfPlan == 4) {
91             plan = new Plan4G();
92         }else if(typeOfPlan == 5) {
93             plan = new Plan5G();
94         }else {
95             System.out.println("Invalid Type of Plan");
96         }
97
98         return plan;
99     }
100 }
101
102 public class FactoryDesignPattern {
103 }
```

5.2 Now under the main of the class **FactoryDesignPattern**, call the getPlan method as **Plan plan = PlanFactory.getPlan()**. Pass the value 3 inside the function call.



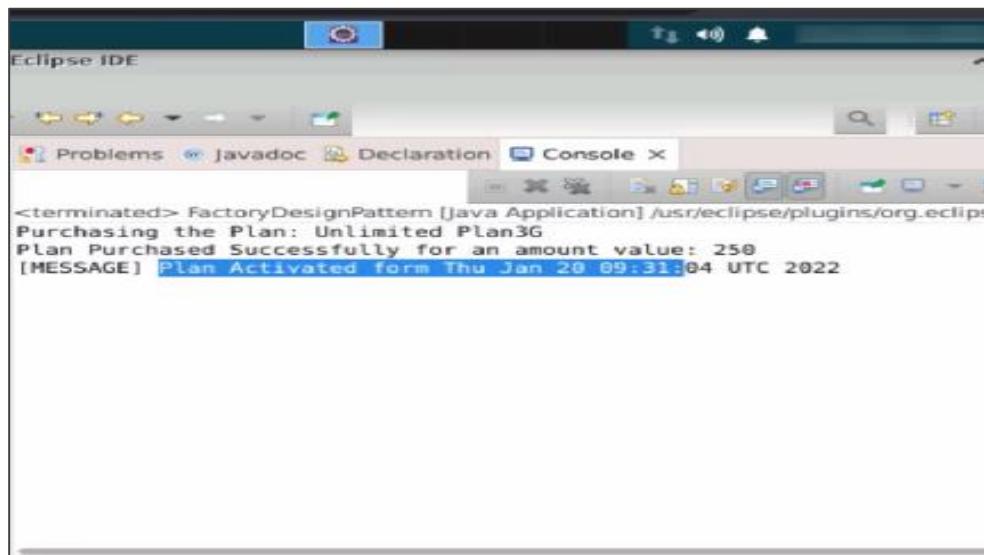
```
79 }
80
81 //Design Pattern - Factory Design Pattern
82 class PlanFactory{
83
84
85    public static Plan getPlan(int typeOfPlan) {
86        Plan plan = null;
87
88        if(typeOfPlan == 3) {
89            plan = new Plan3();
90        }else if(typeOfPlan == 4) {
91            plan = new Plan4();
92        }else if(typeOfPlan == 5) {
93            plan = new Plan5();
94        }else {
95            System.out.println("Invalid Type of Plan");
96        }
97
98        return plan;
99    }
100
101 }
102
103
104
105
106 public class FactoryDesignPattern {
107
108     public static void main(String[] args) {
109         Plan plan = PlanFactory.getPlan(3);
110
111         System.out.println(plan);
112
113     }
114
115 }
```

5.3 Call the **purchase()** method as **plan.purchase()** and then enter the amount. This amount should match the type of plan amount.



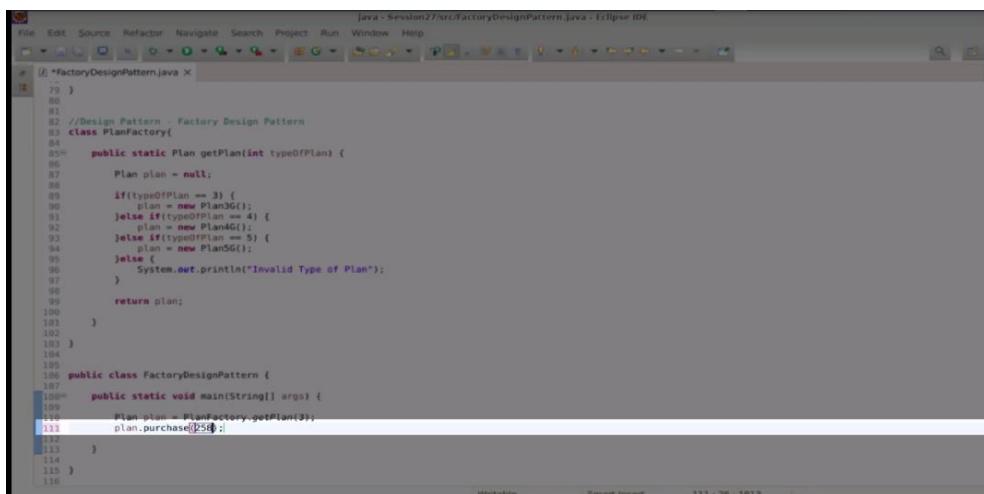
```
79 }
80
81 //Design Pattern - Factory Design Pattern
82 class PlanFactory{
83
84
85    public static Plan getPlan(int typeOfPlan) {
86        Plan plan = null;
87
88        if(typeOfPlan == 3) {
89            plan = new Plan3();
90        }else if(typeOfPlan == 4) {
91            plan = new Plan4();
92        }else if(typeOfPlan == 5) {
93            plan = new Plan5();
94        }else {
95            System.out.println("Invalid Type of Plan");
96        }
97
98        return plan;
99    }
100
101 }
102
103
104
105
106 public class FactoryDesignPattern {
107
108     public static void main(String[] args) {
109         Plan plan = PlanFactory.getPlan(3);
110         plan.purchase(250);
111
112     }
113
114
115 }
```

5.4 Run the code and the following output will be obtained.



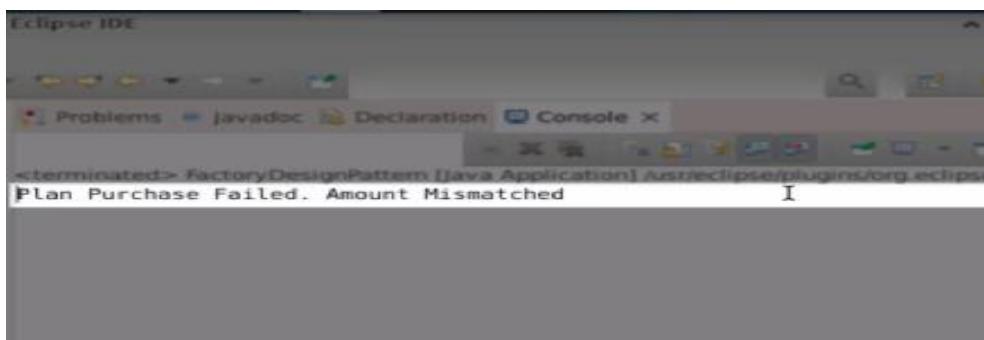
The screenshot shows the Eclipse IDE interface with the title bar "Eclipse IDE". Below the title bar is a toolbar with various icons. The main window has tabs for "Problems", "Javadoc", "Declaration", and "Console". The "Console" tab is selected, displaying the following text:
<terminated> FactoryDesignPattern [java Application] /usr/eclipse/plugins/org.eclipse.Purchasing the Plan: Unlimited Plan3G
Plan Purchased Successfully for an amount value: 250
[MESSAGE] Plan Activated from Thu Jan 20 09:31:04 UTC 2022

5.5 Enter the wrong amount and you will get the message of the amount mismatched.



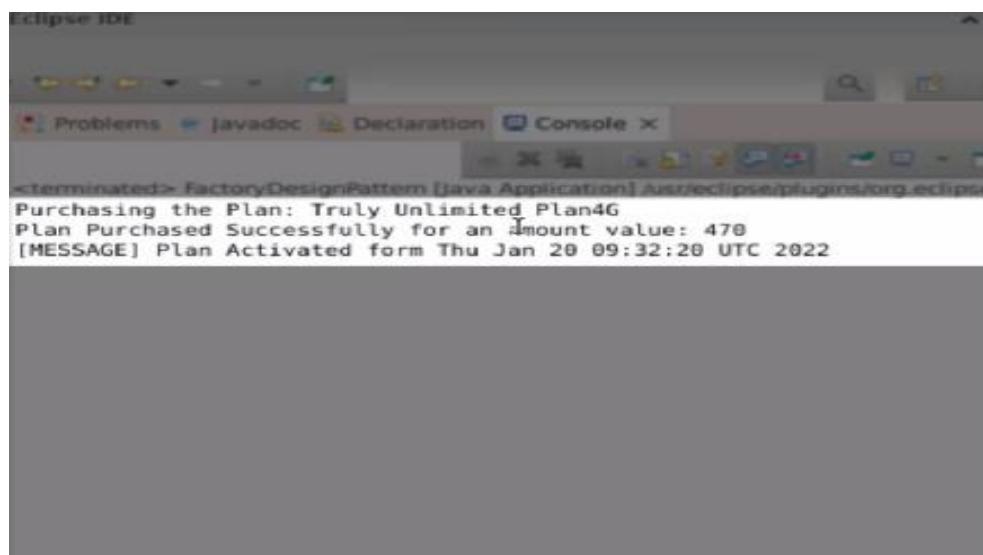
The screenshot shows the Eclipse IDE interface with the title bar "File Edit Source Refactor Navigate Search Project Run Window Help" and the sub-title "java - Session27/src/FactoryDesignPattern.java - Eclipse IDE". The code editor displays the following Java code:

```
79 }  
80 //Design Pattern - Factory Design Pattern  
81  
82 class PlanFactory{  
83  
84     public static Plan getPlan(int typeOfPlan) {  
85         Plan plan = null;  
86         if(typeOfPlan == 3) {  
87             plan = new Plan3G();  
88         }else if(typeOfPlan == 4) {  
89             plan = new Plan4G();  
90         }else if(typeOfPlan == 5) {  
91             plan = new Plan5G();  
92         }else {  
93             System.out.println("Invalid Type of Plan");  
94         }  
95         return plan;  
96     }  
97 }  
98  
99 public class FactoryDesignPattern {  
100     public static void main(String[] args) {  
101         Plan plan = Planfactory.getPlan(3);  
102         plan.purchase();  
103     }  
104 }  
105 }
```



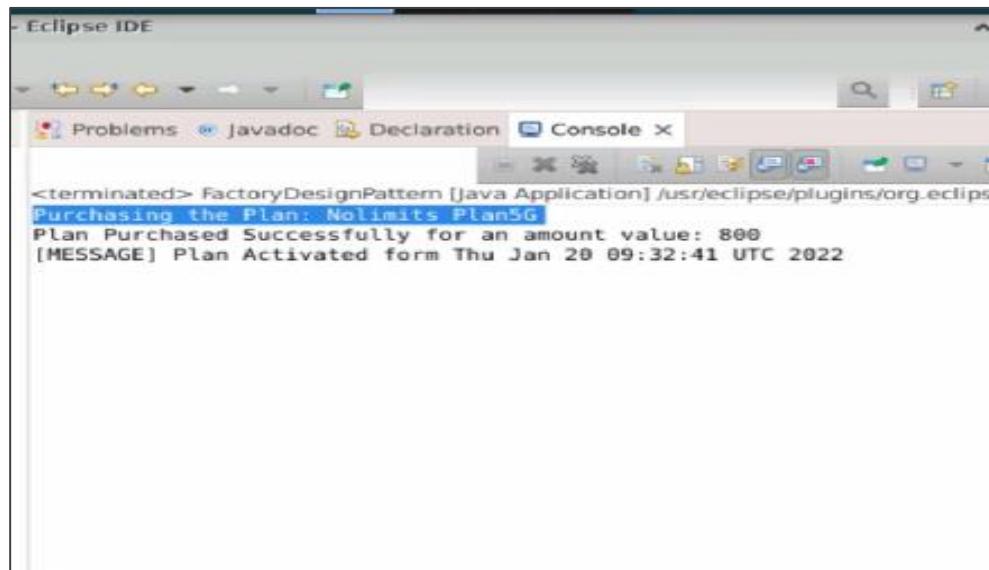
The screenshot shows the Eclipse IDE interface with the title bar "Eclipse IDE". Below the title bar is a toolbar with various icons. The main window has tabs for "Problems", "Javadoc", "Declaration", and "Console". The "Console" tab is selected, displaying the following text:
<terminated> FactoryDesignPattern [java Application] /usr/eclipse/plugins/org.eclipse.Plan Purchase Failed. Amount Mismatched

5.6 Run the code for 4G and 5G plans also by changing the amount and **typeOfplan** value. The following outputs will be obtained.

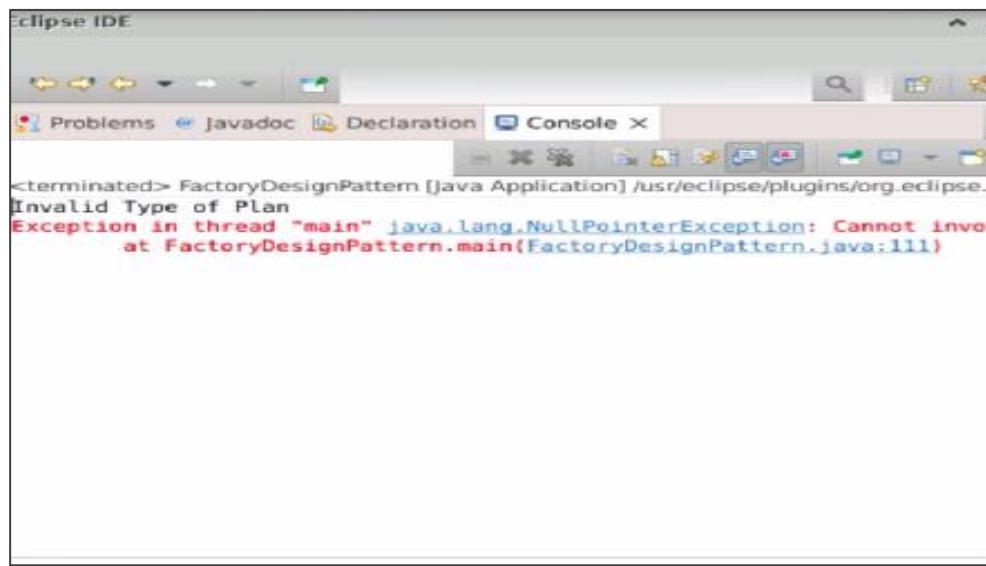


The screenshot shows the Eclipse IDE interface with the title bar "eclipse IDE". Below it is a toolbar with various icons. The main window contains a "Console" tab which is active, showing the following text output:

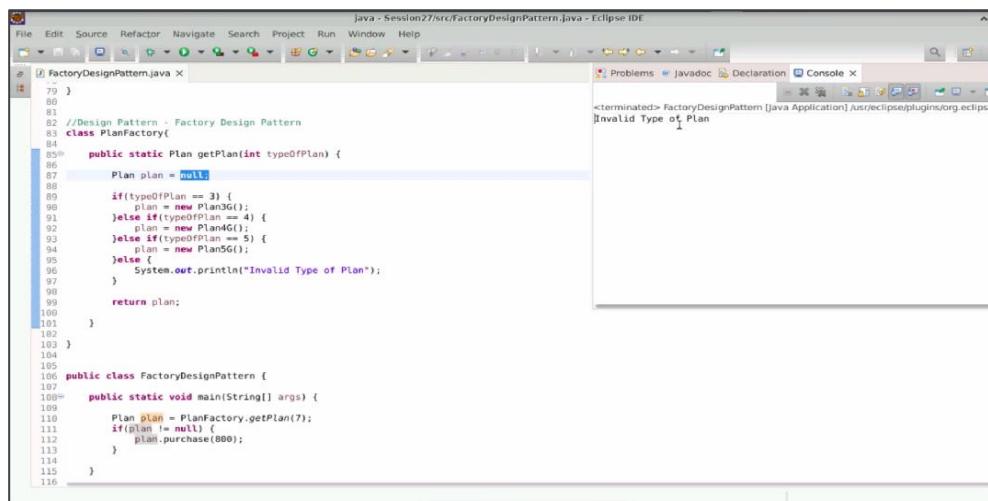
```
<terminated> FactoryDesignPattern [Java Application] AUS/eclipse/plugins/org.eclipse.core.runtime/library/FactoryDesignPattern.jar  
Purchasing the Plan: Truly Unlimited Plan4G  
Plan Purchased Successfully for an Amount value: 478  
[MESSAGE] Plan Activated from Thu Jan 20 09:32:28 UTC 2022
```



5.7 Pass the value as **7** in the getPlan method and run the code. The code will crash.

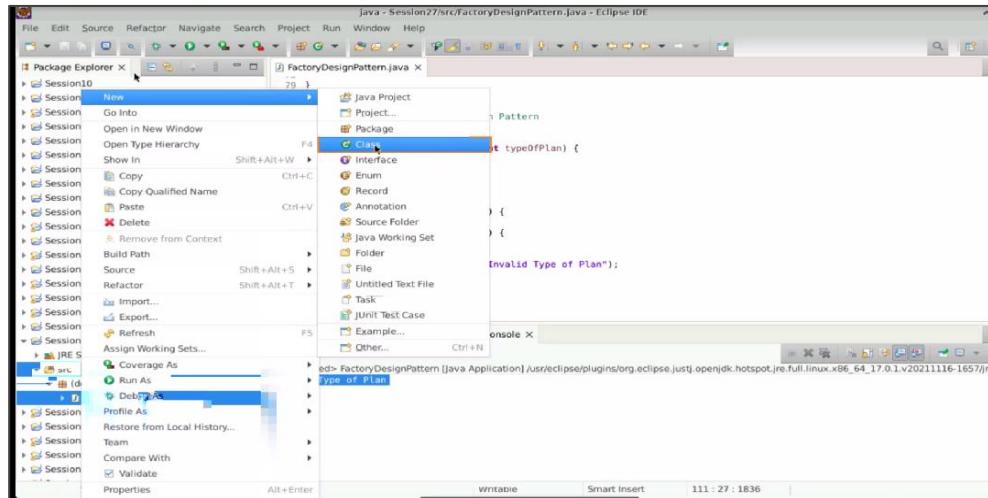


5.8 This is where you need to perform a check. The check involves verifying if your plan is not null. Only if the plan is not null, should you attempt to execute the method called `plan.purchase`. Initially, the plan was null. With this check in place, you won't encounter a crash and instead, you will receive a message indicating an invalid type of plan.

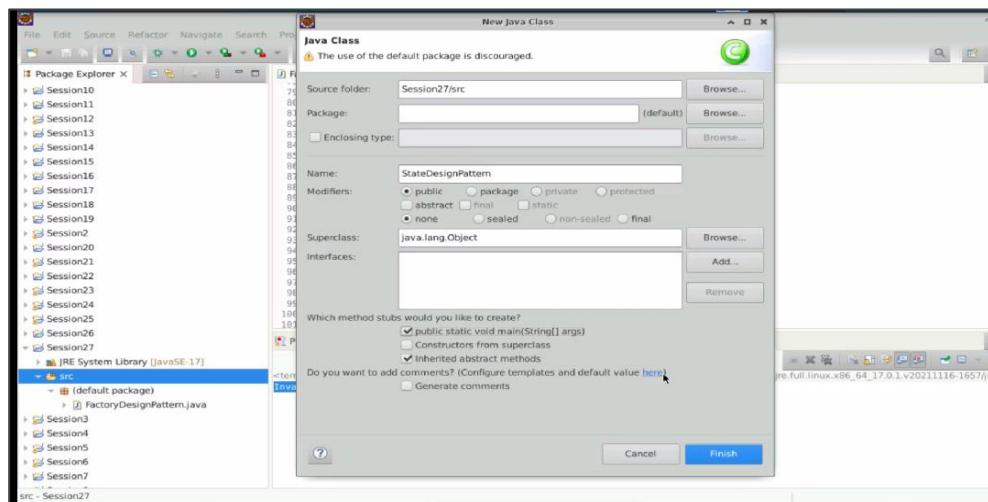


Step 6: Create State Design Patterns

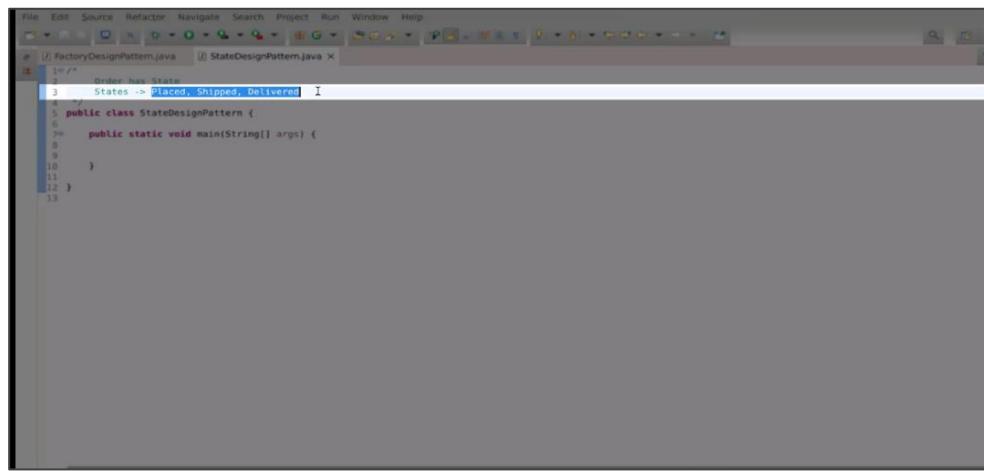
6.1 Right-click on the **src** folder under the project name, select **New**, and then select **Class**.



6.2 Name the class as **StateDesignPattern** and select the option **public static void main(String[] args)**. Select **Finish**. This will create a java class with a **main()** method.



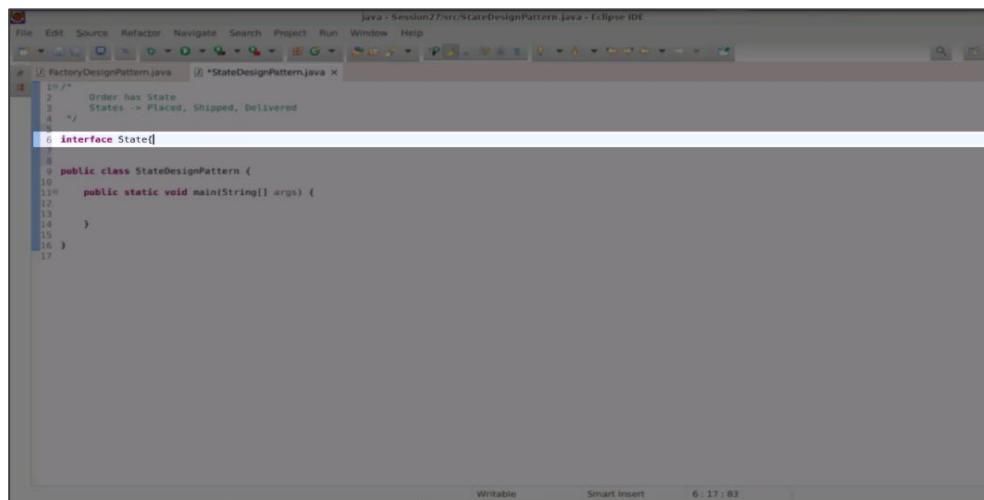
6.3 An order has various states. These states could be **placed**, **shipped**, and **delivered**.



The screenshot shows the Eclipse IDE interface with two tabs open: 'FactoryDesignPattern.java' and 'StateDesignPattern.java'. The code in 'StateDesignPattern.java' is as follows:

```
1 /**
2  * Order has State
3  * States -> Placed, Shipped, Delivered
4 */
5 public class StateDesignPattern {
6     public static void main(String[] args) {
7     }
8 }
9
10
11
12
13
14
15
16
17
```

6.4 In the class **StateDesignPattern**, create an interface **State** by writing **interface State**.



The screenshot shows the Eclipse IDE interface with two tabs open: 'FactoryDesignPattern.java' and '*StateDesignPattern.java'. The code in '*StateDesignPattern.java' is as follows:

```
1 /**
2  * Order has State
3  * States -> Placed, Shipped, Delivered
4 */
5
6 interface State{
7
8
9     public class StateDesignPattern {
10         public static void main(String[] args) {
11         }
12     }
13 }
14
15
16
17
```

6.5 Create three classes, **class Placed**, **class Shipped** and **class Delivered**. All these classes must implement the interface **State**.

```
1*/
2     Order has State
3     States -> Placed, Shipped, Delivered
4     */
5
6     interface State{
7
8 }
9
10    class Placed implements State{
11
12 }
13
14    class Shipped implements State{
15
16 }
17
18    class Delivered implements State{
19
20 }
21
22
23    public class StateDesignPattern {
24
25        public static void main(String[] args) {
26
27
28
29
30
31 }
```

6.6 Create a method **String getStateName()** inside the interface **State**.

```
File Edit Source Refactor Navigate Project Run Window Help
java - Session2 /src/StateDesignPattern.java - Eclipse IDE
FactoryDesignPattern.java *StateDesignPattern.java
1 import java.util.Date;
2
3 /**
4  * Order has State
5  * States -> Placed, Shipped, Delivered
6  */
7
8 interface State{
9     String getStateName();
10 }
11
12 class Placed implements State{
13
14 }
15
16 class Shipped implements State{
17
18 }
19
20 class Delivered implements State{
21
22 }
23
24
25 public class StateDesignPattern {
26
27     public static void main(String[] args) {
28
29
30
31
32 }
```

6.7 Create a new class **Order** which would have a reference to the state. This is called a **has-a-relationship**.

```

File Edit Source Refactor Navigate Search Project Run Window Help
FactoryDesignPattern.java *StateDesignPattern.java
1 import java.util.Date;
2
3 /**
4  * Order has State
5  * States -> Placed, Shipped, Delivered
6 */
7
8 interface State{
9     String getStateName();
10 }
11
12 class Placed implements State{
13 }
14
15 class Shipped implements State{
16 }
17
18 class Delivered implements State{
19 }
20
21 class Order{
22     State state; // Has-A Relationship
23 }
24
25 public class StateDesignPattern {
26     public static void main(String[] args) {
27     }
28 }
29
30
31
32
33
34
35
36
37
38

```

Writable Smart Insert 27 . 39 / 299

6.8 There will be a change in the state. Therefore, create a method **void updateState(State newState)** and pass newState as an input. Make the state as newState.

```

File Edit Source Refactor Navigate Search Project Run Window Help
FactoryDesignPattern.java *StateDesignPattern.java
1 import java.util.Date;
2
3 /**
4  * Order has State
5  * States -> Placed, Shipped, Delivered
6 */
7
8 interface State{
9     String getStateName();
10 }
11
12 class Placed implements State{
13 }
14
15 class Shipped implements State{
16 }
17
18 class Delivered implements State{
19 }
20
21 class Order{
22     State state; // Has-A Relationship
23
24     void updateState(@State newState) {
25         state = newState;
26     }
27 }
28
29 public class StateDesignPattern {
30     public static void main(String[] args) {
31     }
32 }
33
34
35
36
37
38

```

Writable Smart Insert 29 . 36 / 334

6.9 Create the constructor of class **Order** and set the state as placed so that whenever an object of order is created, the state is set to placed by default.

6.10 Complete the unimplemented methods of classes Placed, Shipped and Delivered.

Return the strings **PLACED**, **SHIPPED**, and **DELIVERED** in the **getStateName** method of the three classes.

```

File Edit Source Refactor Navigate Search Project Run Window Help
FactoryDesignPattern.java StateDesignPattern.java
1 import java.util.Date;
2 /**
3  * Order has State
4  * States -> Placed, Shipped, Delivered
5  */
6
7 interface State{
8     String getStateName();
9 }
10
11
12 class Placed implements State{
13     // Adding implemented methods
14 }
15 }
16 // Make type 'Placed' abstract
17 class Placed {
18     // Rename in file (Ctrl+2 R)
19 }
20
21 class
22
23
24 class
25
26
27 // Whenever we create the object of Order, the state is Placed, by default :
28 Order();
29     STATE = new Placed();
30 }
31
32 void updateState(State newState) {
33     state = newState;
34 }
35
36
37
38 }

St_ Press 'Ctrl+Enter' to fix 3 problems of same category in file
Press 'Tab' from proposal table or click for focus
The type Placed must implement the inherited abstract method State.getStateName()

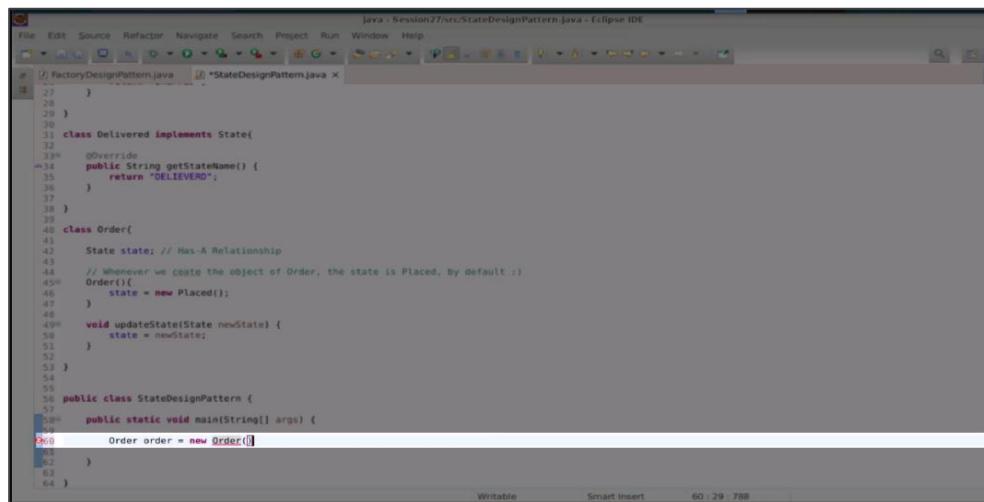
```

```

File Edit Source Refactor Navigate Search Project Run Window Help
FactoryDesignPattern.java StateDesignPattern.java
1 /**
2  * Order has State
3  * States -> Placed, Shipped, Delivered
4  */
5
6 interface State{
7     String getStateName();
8 }
9
10
11 class Placed implements State{
12     @Override
13     public String getStateName() {
14         return "PLACED";
15     }
16 }
17
18 class Shipped implements State{
19     @Override
20     public String getStateName() {
21         // TODO Auto-generated method stub
22         return "SHIPPED";
23     }
24 }
25
26 class Delivered implements State{
27     @Override
28     public String getStateName() {
29         // TODO Auto-generated method stub
30         return "DELIVERED";
31     }
32 }
33
34 class Order{
35 }
36
37
38 }

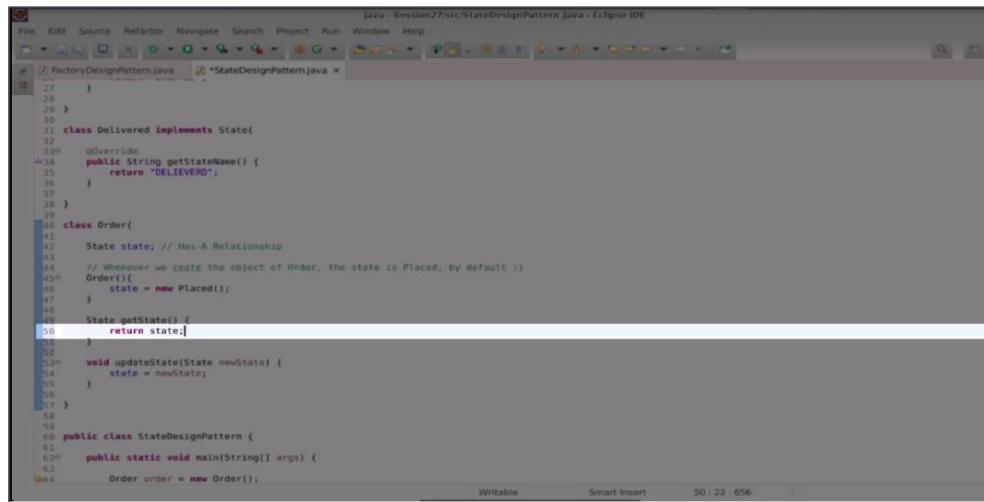

```

6.11 Inside the class **StateDesignPattern** create an object of class **Order**.



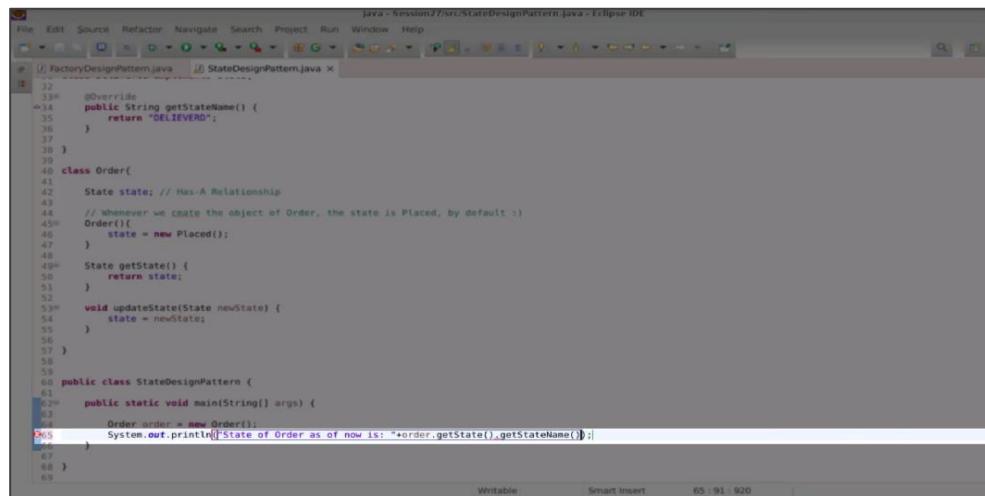
```
File Edit Source Refactor Navigate Search Project Run Window Help
src/FactoryDesignPattern.java *StateDesignPattern.java
27 }
28 }
29 }
30
31 class Delivered implements State{
32
33     @Override
34     public String getStateName() {
35         return "DELIVERED";
36     }
37 }
38
39 class Order{
40
41     State state; // Has-A Relationship
42
43     // whenever we create the object of Order, the state is Placed, by default :
44     Order(){
45         state = new Placed();
46     }
47
48     void updateState(State newState) {
49         state = newState;
50     }
51 }
52
53
54
55 public class StateDesignPattern {
56
57     public static void main(String[] args) {
58
59         Order order = new Order();
60
61     }
62 }
63
64 }
```

6.12 In the class **Order**, create a method **getState()** which returns the state of an order.



```
File Edit Source Refactor Navigate Search Project Run Window Help
src/FactoryDesignPattern.java *StateDesignPattern.java
27 }
28 }
29 }
30
31 class Delivered implements State{
32
33     @Override
34     public String getStateName() {
35         return "DELIVERED";
36     }
37 }
38
39 class Order{
40
41     State state; // Has-A Relationship
42
43     // whenever we create the object of Order, the state is Placed, by default :
44     Order(){
45         state = new Placed();
46     }
47
48     State getState(){
49         return state;
50     }
51
52     void updateState(State newState) {
53         state = newState;
54     }
55 }
56
57
58
59 public class StateDesignPattern {
60
61     public static void main(String[] args) {
62
63         Order order = new Order();
64
65     }
66 }
67
68 }
```

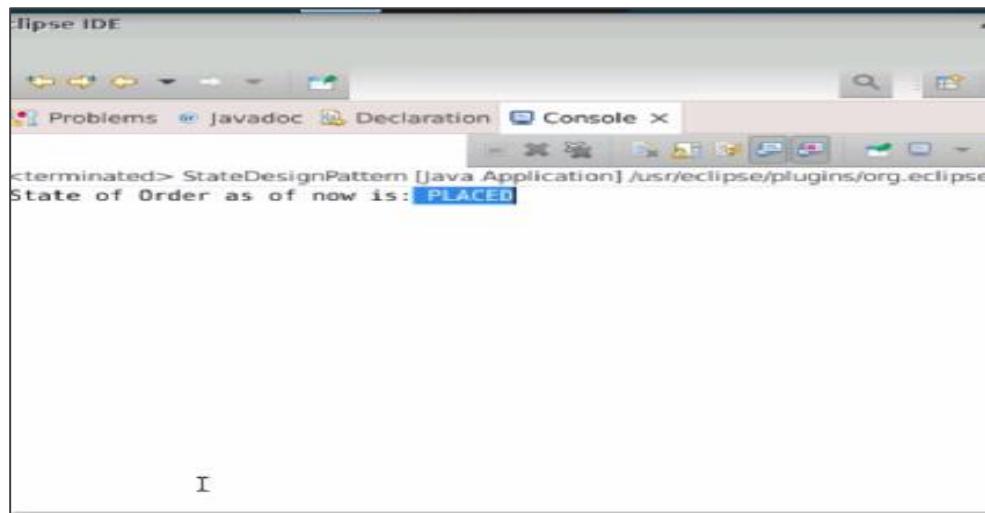
6.13 Print the state of the order as `order.getState().getizedName()`.



The screenshot shows the Eclipse IDE interface with two tabs open: "FactoryDesignPattern.java" and "StateDesignPattern.java". The "StateDesignPattern.java" tab is active, displaying the following Java code:

```
1  package com.simplilearn.StateDesignPattern;
2
3  public class FactoryDesignPattern {
4
5      public static void main(String[] args) {
6          Order order = new Order();
7          System.out.println("State of Order as of now is: "+order.getState().getizedName());
8      }
9  }
10
11  class Order {
12
13      @Override
14      public String getSimpleName() {
15          return "DELIVERD";
16      }
17
18  }
19
20  class Order{
21
22      State state; // Has-A Relationship
23
24      // whenever we create the object of Order, the state is Placed, by default :
25      Order(){
26          state = new Placed();
27      }
28
29      State getState() {
30          return state;
31      }
32
33      void updateState(State newState) {
34          state = newState;
35      }
36
37  }
38
39  public class StateDesignPattern {
40
41      public static void main(String[] args) {
42          Order order = new Order();
43          System.out.println("State of Order as of now is: "+order.getState().getizedName());
44      }
45  }
46
47 }
```

6.14 Run the code and implement the basic code of **StateDesignPattern** and obtain the following output.



The screenshot shows the Eclipse IDE interface with the "Console" tab selected. The console window displays the following text:

```
terminated> StateDesignPattern [Java Application] /usr/eclipse/plugins/org.eclipse.jdt.core
State of Order as of now is: PLACED
```

6.15 Write the statement **Date dateTimeStamp** in the three classes to record the date timestamp associated with these classes.



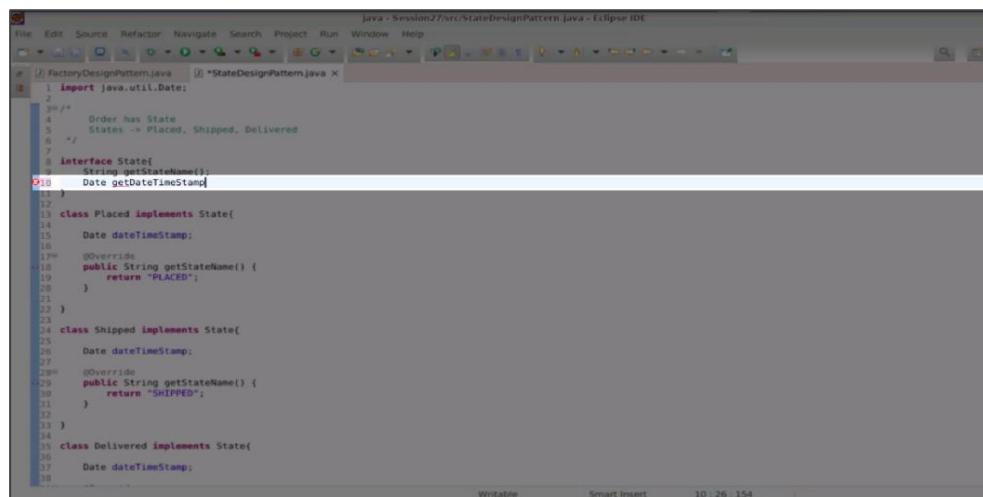
```

12
13 class Placed implements State{
14
15     Date dateTimeStamp;
16
17     @Override
18     public String getStateName() {
19         return "PLACED";
20     }
21
22 }
23
24 class Shipped implements State{
25
26     Date dateTimeStamp;
27
28     @Override
29     public String getStateName() {
30         return "SHIPPED";
31     }
32
33 }
34
35 class Delivered implements State{
36
37     Date dateTimeStamp;
38

```

Writable Smart Insert 37 : 24 : 444

6.16 Create two more methods in the state interface. The methods are: **void updateTimeStamp()** and **Date getDateTimeStamp()**.



```

1 // Order has State
2 // States -> Placed, Shipped, Delivered
3
4 interface State{
5     String getStateName();
6     Date getDateTimeStamp();
7 }
8
9 class Placed implements State{
10
11     Date dateTimeStamp;
12
13     @Override
14     public String getStateName() {
15         return "PLACED";
16     }
17 }
18
19 class Shipped implements State{
20
21     Date dateTimeStamp;
22
23     @Override
24     public String getStateName() {
25         return "SHIPPED";
26     }
27 }
28
29 class Delivered implements State{
30
31     Date dateTimeStamp;
32
33 }
34
35 
```

Writable Smart Insert 10 : 26 : 154

```

1  package com.simplilearn;
2
3  import java.util.Date;
4
5  /**
6   * Order has State
7   * States -> Placed, Shipped, Delivered
8   */
9
10 interface State{
11     void updateTimeStamp(Date date);
12     Date getDateTimeStamp();
13 }
14
15 class Placed implements State{
16     Date dateTimeStamp;
17
18     @Override
19     public String getStateName() {
20         return "PLACED";
21     }
22 }
23
24 class Shipped implements State{
25     Date dateTimeStamp;
26
27     @Override
28     public String getStateName() {
29         return "SHIPPED";
30     }
31 }
32
33 class Delivered implements State{
34     Date dateTimeStamp;
35
36     @Override
37     public String getStateName() {
38         return "DELIVERED";
39     }
40 }
41
42 class Order{
43     State state; // Has-A Relationship
44
45     // Whenever we create the object of Order, the state is Placed, by default.
46 }

```

- 6.17 Implement these two methods inside the three classes. In the updateTimeStamp method, copy the date to dateTimeStamp. And in the getDateStamp method, simply return the dateTimeStamp attribute.

```

1  package com.simplilearn;
2
3  import java.util.Date;
4
5  /**
6   * Order has State
7   * States -> Placed, Shipped, Delivered
8   */
9
10 interface State{
11     void updateTimeStamp(Date date);
12     Date getDateTimeStamp();
13 }
14
15 class Placed implements State{
16     Date dateTimeStamp;
17
18     @Override
19     public Date getDateTimeStamp() {
20         // TODO Auto-generated method stub
21         return null;
22     }
23 }
24
25 class Shipped implements State{
26     Date dateTimeStamp;
27
28     @Override
29     public String getStateName() {
30         return "SHIPPED";
31     }
32 }
33
34 class Delivered implements State{
35     Date dateTimeStamp;
36
37     @Override
38     public void updateTimeStamp(Date date) {
39         dateTimeStamp = date;
40     }
41 }
42
43 class Order{
44     State state; // Has-A Relationship
45
46     // Whenever we create the object of Order, the state is Placed, by default.
47 }

```

```

File Edit Source Refactor Navigate Search Project Run Window Help
java - Session27/src/StateDesignPattern.java - Eclipse IDE
1  package FactoryDesignPattern;
2
3  public class StateDesignPattern {
4
5      public static void main(String[] args) {
6          Order order = new Order();
7          System.out.println("State of Order as of now is: "+order.getState().getStateName());
8      }
9  }
10
11  interface State {
12      void updateTimeStamp(Date date);
13      Date getDateStamp();
14  }
15
16  class Delivered implements State{
17      Date dateTimeStamp;
18
19      @Override
20      public String getStateName() {
21          return "DELIVERED";
22      }
23
24      @Override
25      public void updateTimeStamp(Date date) {
26          dateTimeStamp = date;
27      }
28
29      @Override
30      public Date getDateStamp() {
31          return dateTimeStamp;
32      }
33  }
34
35  class Order{
36      State state; // Has-A Relationship
37
38      // Whenever we create the object of Order, the state is Placed, by default :)
39      Order(){
40          state = new Placed();
41          state.updateTimeStamp(new Date());
42      }
43
44      State getState() {
45          return state;
46      }
47
48      void updateState(State newState) {
49          state = newState;
50      }
51
52  }
53
54  public class FactoryDesignPattern {
55
56      public static void main(String[] args) {
57          Order order = new Order();
58          System.out.println("State of Order as of now is: "+order.getState().getStateName());
59      }
60  }

```

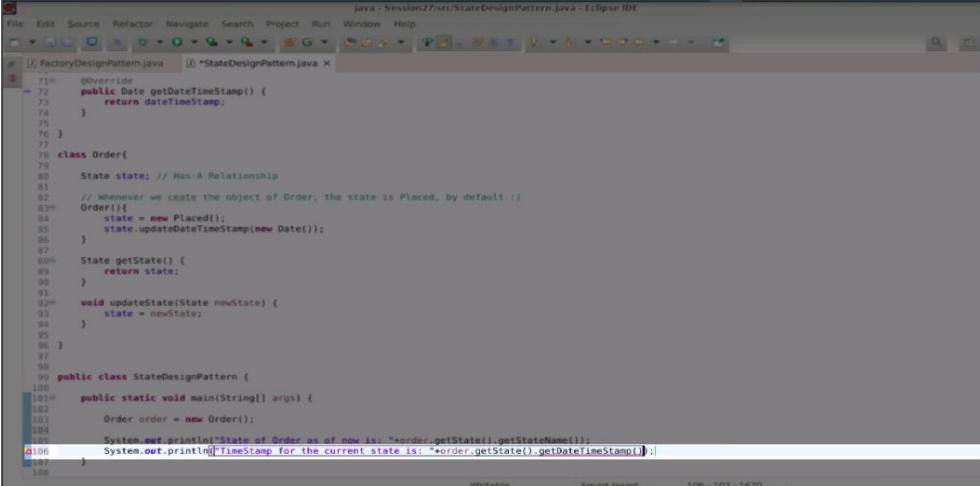
6.18 Create an object of date to display the current date when you place an order by writing **state.updateTimeStamp(new Date) in the constructor **Order**.**

```

File Edit Source Refactor Navigate Search Project Run Window Help
java - Session27/src/StateDesignPattern.java - Eclipse IDE
1  package FactoryDesignPattern;
2
3  public class StateDesignPattern {
4
5      public static void main(String[] args) {
6          Order order = new Order();
7          System.out.println("State of Order as of now is: "+order.getState().getStateName());
8      }
9  }
10
11  interface State {
12      void updateTimeStamp(Date date);
13      Date getDateStamp();
14  }
15
16  class Delivered implements State{
17      Date dateTimeStamp;
18
19      @Override
20      public String getStateName() {
21          return "DELIVERED";
22      }
23
24      @Override
25      public void updateTimeStamp(Date date) {
26          dateTimeStamp = date;
27      }
28
29      @Override
30      public Date getDateStamp() {
31          return dateTimeStamp;
32      }
33  }
34
35  class Order{
36      State state; // Has-A Relationship
37
38      // Whenever we create the object of Order, the state is Placed, by default :)
39      Order(){
40          state = new Placed();
41          state.updateTimeStamp(new Date());
42      }
43
44      State getState() {
45          return state;
46      }
47
48      void updateState(State newState) {
49          state = newState;
50      }
51
52  }
53
54  public class FactoryDesignPattern {
55
56      public static void main(String[] args) {
57          Order order = new Order();
58          System.out.println("State of Order as of now is: "+order.getState().getStateName());
59      }
60  }

```

6.19 Check the state of your order and date timestamp by calling **order.getState().getDateTimeStamp()**.



```
File Edit Source Refactor Navigate Search Project Run Window Help
I FactoryDesignPattern.java  I *StateDesignPattern.java - Eclipse IDE
719
720     @Override
721     public Date getDateTimeStamp() {
722         return dateTimeStamp;
723     }
724 }
725
726 class Order{
727
728     State state; // Has-A Relationship
729
730     // Whenever we create the object of Order, the state is Placed, by default :)
731     Order(){
732         state = new Placed();
733         state.updateTimeStamp(new Date());
734     }
735
736     State getState() {
737         return state;
738     }
739
740     void updateState(State newState) {
741         state = newState;
742     }
743 }
744
745 public class StateDesignPattern {
746
747     public static void main(String[] args) {
748
749         Order order = new Order();
750
751         System.out.println("State of Order as of now is: "+order.getState().getStateName());
752         System.out.println("TimeStamp for the current state is: "+order.getState().getDateTimeStamp());
753     }
754 }
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1787
1787
1788
1789
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2869
2870
2871
2872
2873
2874

```

- 6.21 Introduce a delay of 3 seconds by writing Thread.sleep(3000). Surround this by try catch block by clicking on the error bulb.

```

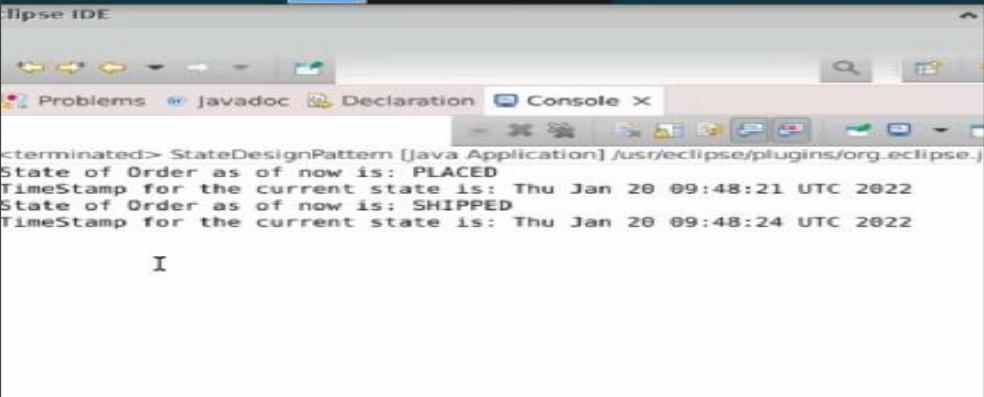
113     }
114 
115     State state;
116     state = new Placed();
117     state.updateTimeStamp(new Date());
118 
119     order.updateState(state);
120 
121     System.out.println("State of Order as of now is: "+order.getState().getStateName());
122     System.out.println("TimeStamp for the current state is: "+order.getState().getDateTimeStamp());
123 
124 }
125 
```

- 6.22 Create a new state called shipped state and update the date timestamp and give once again a new object of date. Then update the state to this new state by writing order.updateState(state). Copy paste the two print statements above.

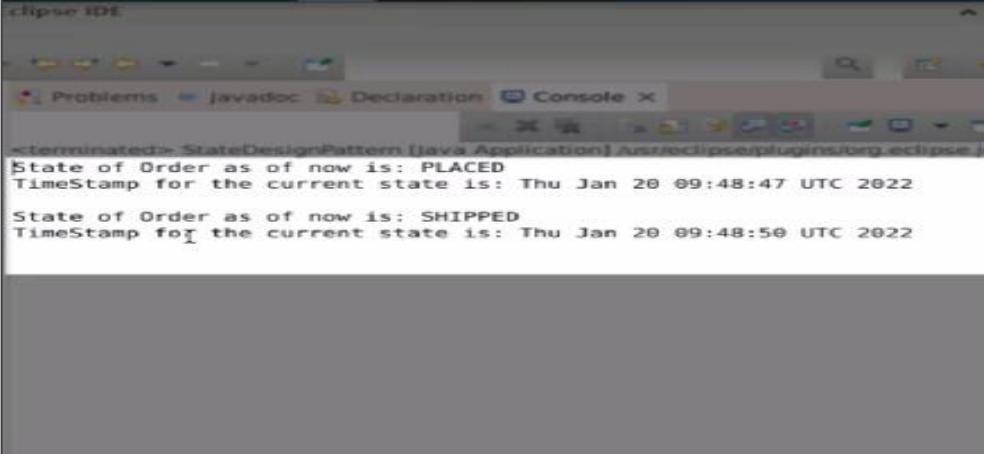
```

114 
115     State state;
116     state = new Shipped();
117     state.updateTimeStamp(new Date());
118 
119     order.updateState(state);
120 
121     System.out.println("State of Order as of now is: "+order.getState().getStateName());
122     System.out.println("TimeStamp for the current state is: "+order.getState().getDateTimeStamp());
123 
124 }
125 
```

6.23 Run the code.



```
<terminated> StateDesignPattern [Java Application] /usr/eclipse/plugins/org.eclipse.jdt.core.prefs
State of Order as of now is: PLACED
TimeStamp for the current state is: Thu Jan 20 09:48:21 UTC 2022
State of Order as of now is: SHIPPED
TimeStamp for the current state is: Thu Jan 20 09:48:24 UTC 2022
```



```
<terminated> StateDesignPattern [Java Application] /usr/eclipse/plugins/org.eclipse.jdt.core.prefs
State of Order as of now is: PLACED
TimeStamp for the current state is: Thu Jan 20 09:48:47 UTC 2022
State of Order as of now is: SHIPPED
TimeStamp for the current state is: Thu Jan 20 09:48:50 UTC 2022
```

By following the above steps, you have successfully implemented the Factory and State design patterns. These patterns enhance flexibility, reusability, and scalability in your software, creating a robust and efficient solution.