



Name of Course Instructor: Mr. Nischal Shakya

Course Code: **MCS111**

Course Name: **Object Oriented Programming**

Program Name: **MCS**

Semester: **1st**

Assignment No: **Re Assessment**

Assignment Type (Individual/Group):

Individual

Assignment Title: **Re Assessment for the Mid Term**

Date of Submission: **11/03/2023**

Declaration

We, hereby declare that the assignment work submitted is original and completed by us and as per our knowledge and practices.

Name of the Student

ID number

Email Id

Ankur Gajurel

22112018

ankur22112018@licnepal.edu.np

1. We have to calculate the area of a rectangle, a square and a circle. Create an abstract class 'Shape' with three abstract methods namely 'RectangleArea' taking two parameters, 'SquareArea' and 'CircleArea' taking one parameter each. The parameters of 'RectangleArea' are its length and breadth, that of 'SquareArea' is its side and that of 'CircleArea' is its radius. Now create another class 'Area' containing all the three methods 'RectangleArea', 'SquareArea' and 'CircleArea' for printing the area of rectangle, square and circle respectively. Create an object of class 'Area' and call all the three methods. (10 marks)

Code:

Main.java

```
import java.util.Scanner;

no usages new *
public class Main{
    no usages new *
    public static void main(String[] args) {

        Area area = new Area();
        Scanner in = new Scanner(System.in);

        System.out.print("Enter length for rectangle: ");
        double len = in.nextDouble();
        System.out.print("Enter breadth for rectangle: ");
        double bre = in.nextDouble();
        area.RectangleArea(len, bre);

        System.out.print("Enter breadth for square: ");
        double sid = in.nextDouble();
        area.SquareArea(sid);

        System.out.print("Enter breadth for circle: ");
        double rad = in.nextDouble();
        area.CircleArea(rad);

    }
}
```

Area.java

```
1      import static java.lang.Math.*;
2
3      2 usages new *
4      public class Area extends Shape{
5
6          1 usage new *
7          @Override
8          public void RectangleArea(double lenght, double width) {
9              System.out.printf("Rectangle area = %.2f", lenght*width);
10             System.out.println("");
11         }
12
13         1 usage new *
14         @Override
15         public void SquareArea(double side) {
16             System.out.printf("Square area = %.2f", pow(side,2));
17             System.out.println("");
18         }
19
20         1 usage new *
21         @Override
22         public void CircleArea(double radius) {
23             System.out.printf("Circle area = %.2f", PI*pow(radius, 2));
24             System.out.println("");
25         }
26     }
```

Shape.java

```
1 usage 1 inheritor new *
public abstract class Shape {
    1 usage 1 implementation new *
    public abstract void RectangleArea(double length, double width);
    1 usage 1 implementation new *
    public abstract void SquareArea(double side);
    1 usage 1 implementation new *
    public abstract void CircleArea(double radius);
}
```

Explanation:

This code that calculates the areas of different shapes, including a rectangle, a square, and a circle. It prompts the user to input the required dimensions of each shape and then calculates and displays their respective areas.

The code starts with the class Main, which contains the main method. This method creates an instance of the Area class and a Scanner object to read input from the user.

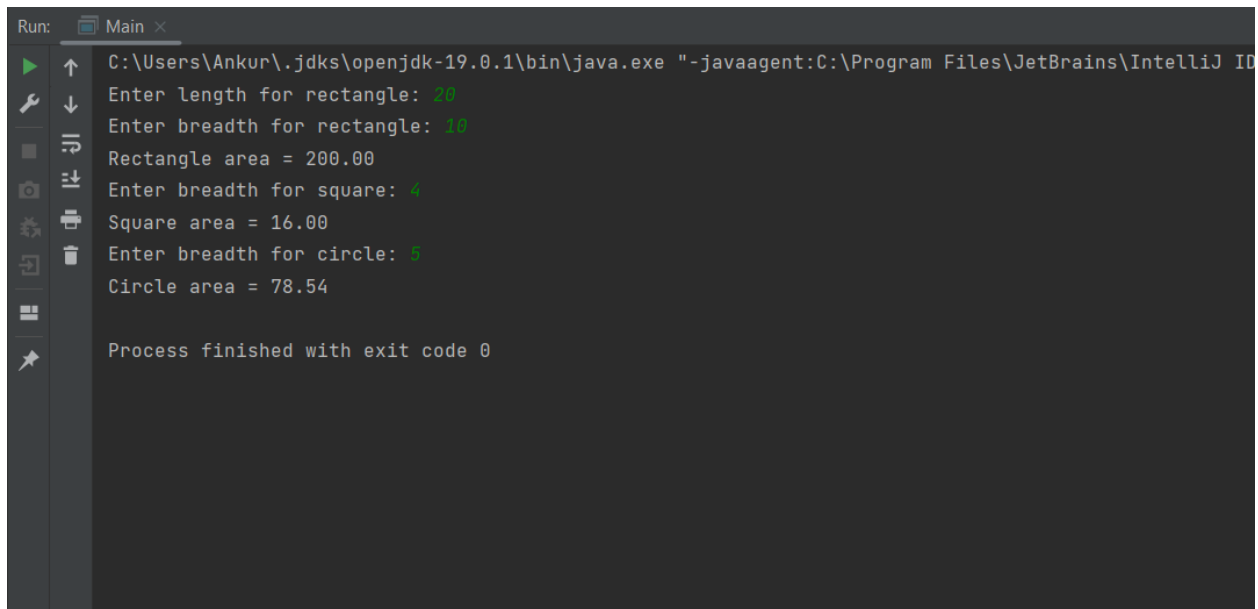
The code then prompts the user to input the length and breadth of a rectangle using the Scanner object and stores the values in the len and bre variables respectively. The RectangleArea() method of the Area class is then called with these values as parameters to calculate and display the area of the rectangle.

Next, the user is prompted to input the length of a square, which is stored in the sid variable. The SquareArea() method of the Area class is then called with this value as a parameter to calculate and display the area of the square.

Finally, the user is prompted to input the radius of a circle, which is stored in the rad variable. The CircleArea() method of the Area class is then called with this value as a parameter to calculate and display the area of the circle.

In summary, this code prompts the user to input dimensions for different shapes and then calculates and displays their respective areas using methods defined in the Area class.

Output:

A screenshot of a Java IDE's run console. The window title is 'Run: Main x'. The command line shows the execution of java.exe with a javaagent. The output consists of several lines: 'Enter length for rectangle: 20', 'Enter breadth for rectangle: 10', 'Rectangle area = 200.00', 'Enter breadth for square: 4', 'Square area = 16.00', 'Enter breadth for circle: 5', 'Circle area = 78.54', and 'Process finished with exit code 0'. The input values are shown in green text.

```
Run: Main x
C:\Users\Ankur\.jdk\openjdk-19.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ ID
Enter length for rectangle: 20
Enter breadth for rectangle: 10
Rectangle area = 200.00
Enter breadth for square: 4
Square area = 16.00
Enter breadth for circle: 5
Circle area = 78.54
Process finished with exit code 0
```

2. Imagine you are creating a furniture shop simulator. The code consists of the classes that represent. (10 Marks)

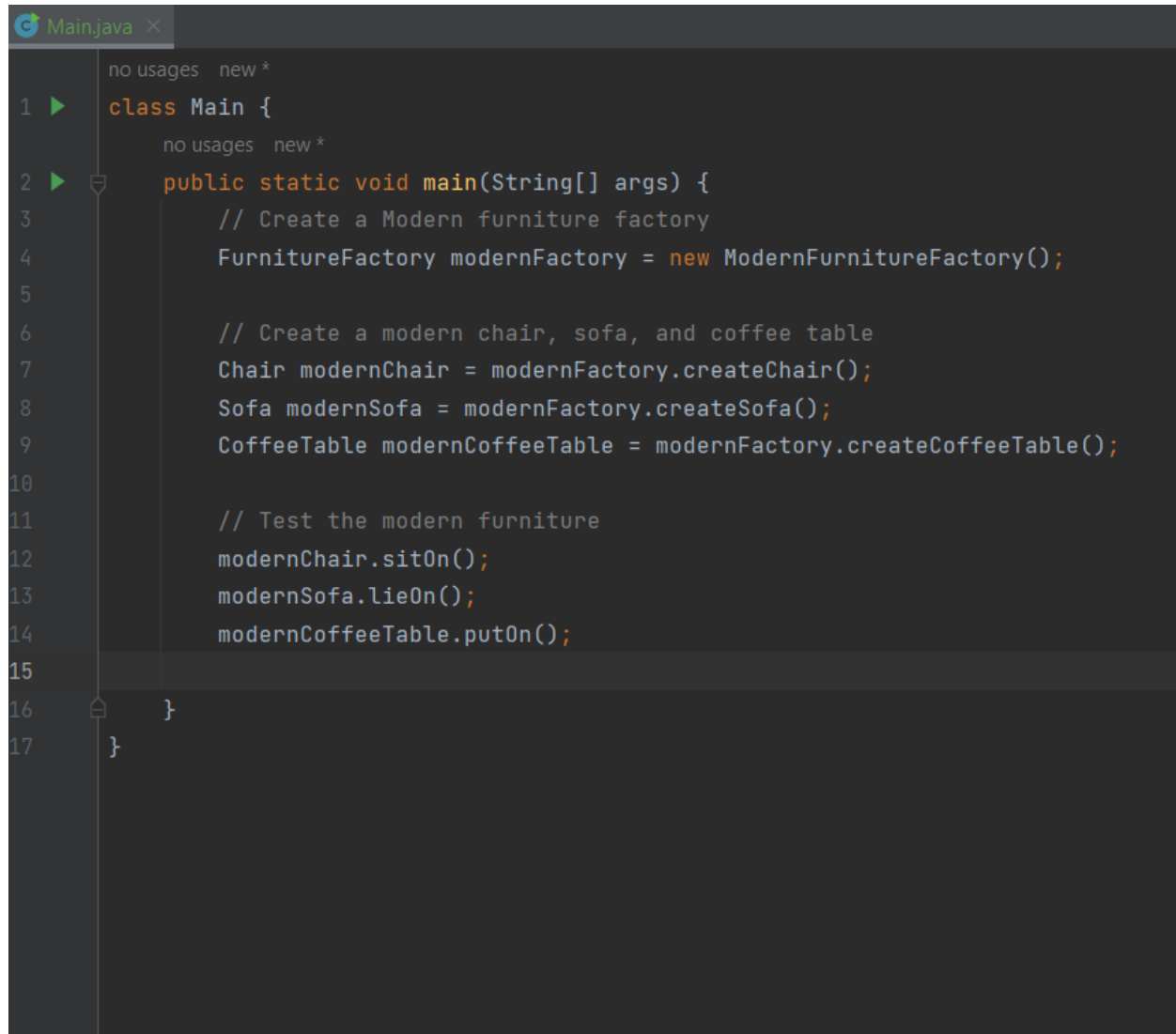
- a. A family of related products, say Chair, Sofa, CoffeeTable
- b. Several variants of this family For example, products Chair, Sofa, CoffeeTable, are available in Modern, Victorian, ArtDeco.

Find the way to create the individual furniture objects so that they match other objects of the same family. Customers will get quite mad when they receive non-matching furniture. Also, you don't want to change existing code when adding new products or families of products to the program. Furniture vendors update their catalogs very often, and you wouldn't want to change the core code each time it happens.

Ans :

Code:

Main.java



```
no usages  new *
1  ► class Main {
    no usages  new *
2  ►  ► public static void main(String[] args) {
3      // Create a Modern furniture factory
4      FurnitureFactory modernFactory = new ModernFurnitureFactory();
5
6      // Create a modern chair, sofa, and coffee table
7      Chair modernChair = modernFactory.createChair();
8      Sofa modernSofa = modernFactory.createSofa();
9      CoffeeTable modernCoffeeTable = modernFactory.createCoffeeTable();
10
11     // Test the modern furniture
12     modernChair.sitOn();
13     modernSofa.lieOn();
14     modernCoffeeTable.putOn();
15
16 }
17 }
```

ArtDecoChair.java

```
ArtDecoChair.java x
1 usage new *
1 class ArtDecoChair implements Chair {
  1 usage new *
2 public void sitOn() {
3     System.out.println("Sit on Art Deco chair.");
4 }
5 }
6
```

ArtDecoCoffeeTable.java

```
ArtDecoCoffeeTable.java x
1 usage new *
1 class ArtDecoCoffeeTable implements CoffeeTable {
  1 usage new *
2 public void putOn() {
3     System.out.println("Putting Coffee on an Art Deco coffee table.");
4 }
5 }
6
```

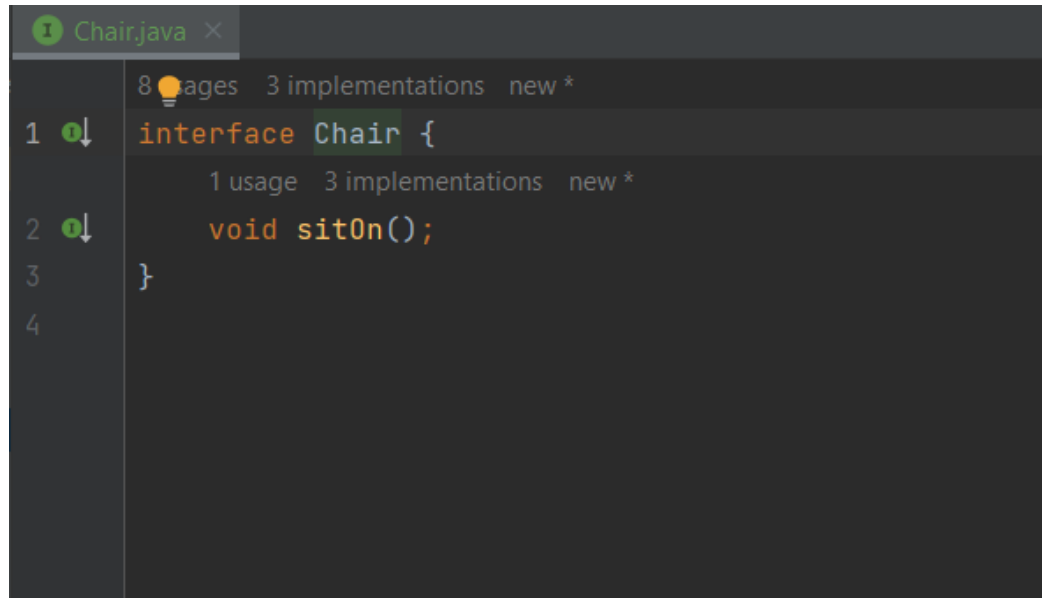
ArtDecoFurnitureFactory.java

```
no usages new *
1  class ArtDecoFurnitureFactory implements FurnitureFactory {
    1 usage new *
2  public Chair createChair() {
3      return new ArtDecoChair();
4  }
5
    1 usage new *
6  public Sofa createSofa() {
7      return new ArtDecoSofa();
8  }
9
    1 usage new *
10 public CoffeeTable createCoffeeTable() {
11     return new ArtDecoCoffeeTable();
12 }
13 }
14
```

ArtDecoSofa.java

```
ArtDecoSofa.java x
1  1 usage new *
2  class ArtDecoSofa implements Sofa {
    1 usage new *
3  public void lieOn() {
4      System.out.println("Lying on an Art Deco sofa.");
5  }
6  }
```


Chair.java

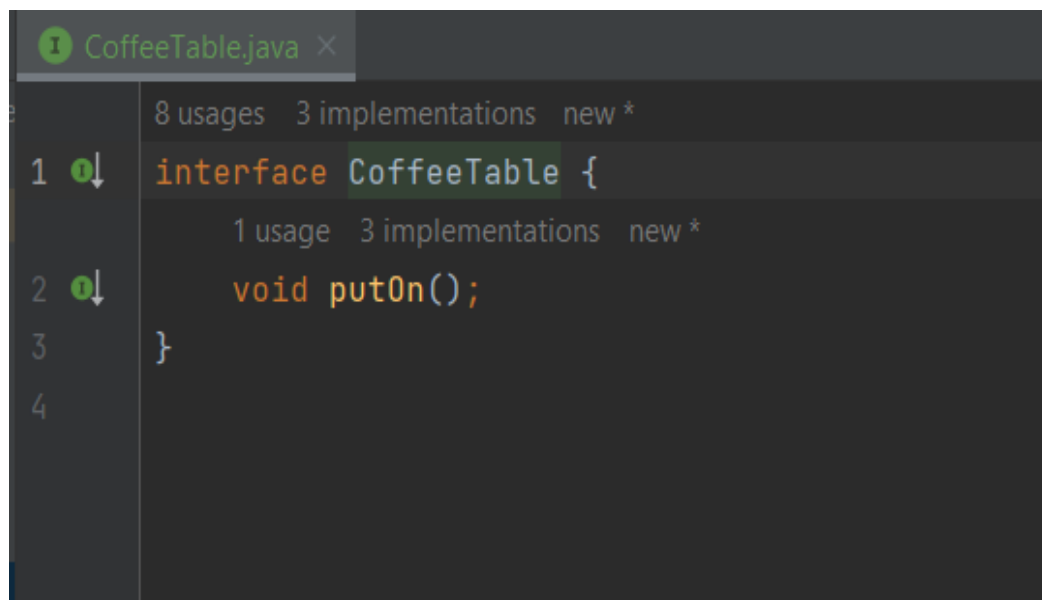


The screenshot shows a code editor window titled "Chair.java". The code defines an interface named "Chair" with a single method "sitOn()". The interface is located on lines 1 to 3 of the file. The code is as follows:

```
1 interface Chair {  
2     void sitOn();  
3 }  
4
```

Line numbers 1, 2, 3, and 4 are visible on the left side of the editor. The word "interface" is highlighted in orange, and "Chair" is highlighted in green. The method "sitOn()" is also highlighted in orange. The closing brace "}" is on line 3.

CoffeeTable.java

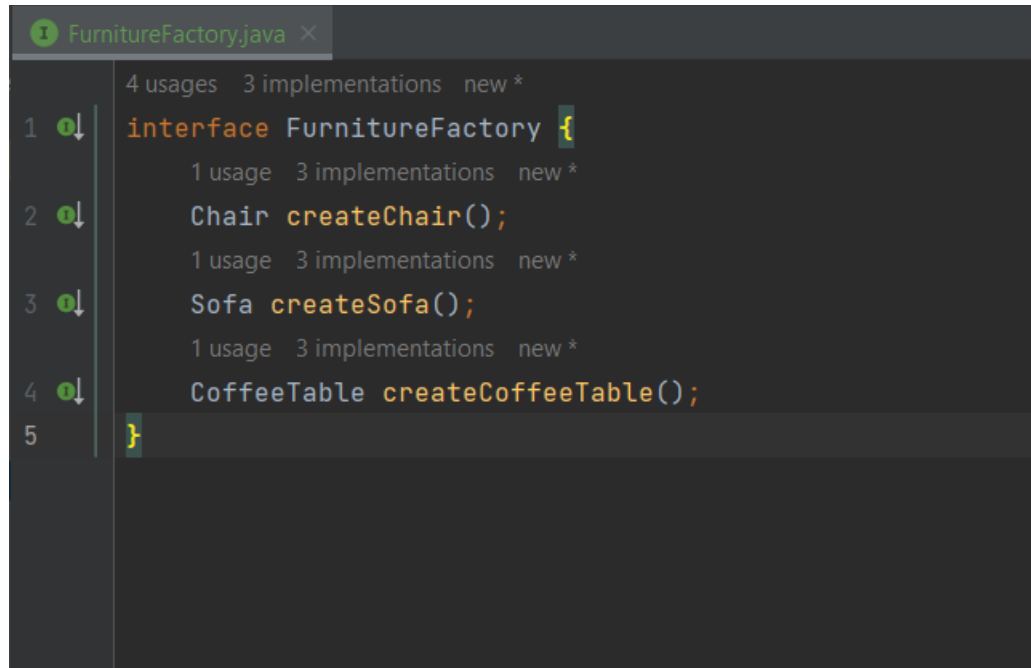


The screenshot shows a code editor window titled "CoffeeTable.java". The code defines an interface named "CoffeeTable" with a single method "putOn()". The interface is located on lines 1 to 3 of the file. The code is as follows:

```
1 interface CoffeeTable {  
2     void putOn();  
3 }  
4
```

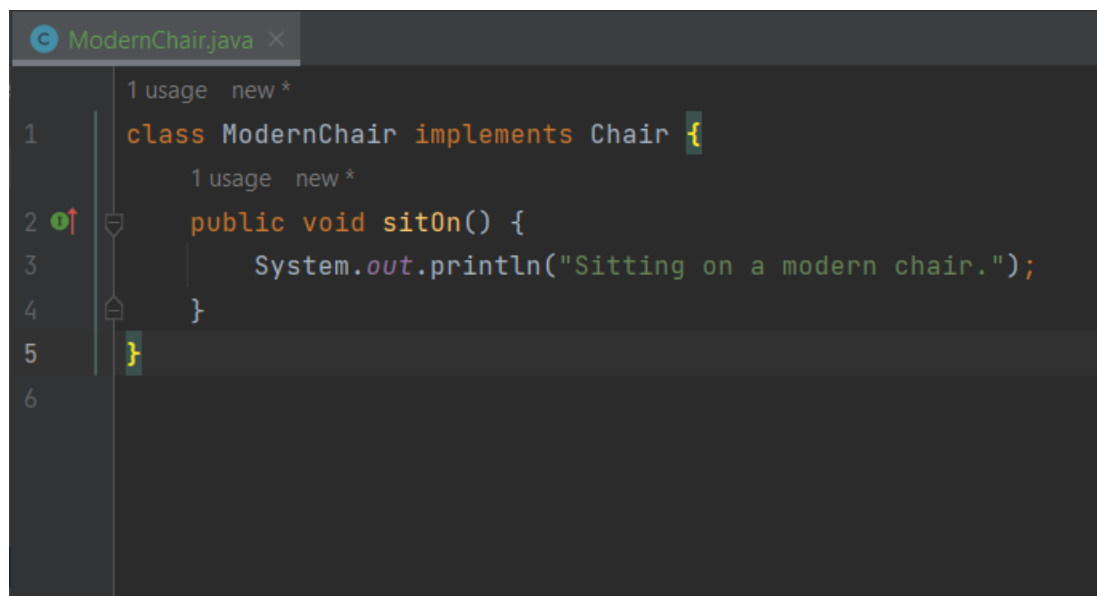
Line numbers 1, 2, 3, and 4 are visible on the left side of the editor. The word "interface" is highlighted in orange, and "CoffeeTable" is highlighted in green. The method "putOn()" is also highlighted in orange. The closing brace "}" is on line 3.

FurnitureFactory.java



```
FurnitureFactory.java x
4 usages 3 implementations new *
1 interface FurnitureFactory {
    1 usage 3 implementations new *
2     Chair createChair();
    1 usage 3 implementations new *
3     Sofa createSofa();
    1 usage 3 implementations new *
4     CoffeeTable createCoffeeTable();
5 }
```

ModernChair.java



```
ModernChair.java x
1 usage new *
1 class ModernChair implements Chair {
    1 usage new *
2     public void sitOn() {
3         System.out.println("Sitting on a modern chair.");
4     }
5 }
6
```

ModernCoffeeTable.java

```
ModernCoffeeTable.java x
1 1 usage new *
1  class ModernCoffeeTable implements CoffeeTable {
2      1 usage new *
3      public void putOn() {
4          System.out.println("Putting Coffee on a modern coffee table.");
5      }
6  }
```

ModernFurnitureFactory.java

```
ModernFurnitureFactory.java x
1 1 usage new *
1  class ModernFurnitureFactory implements FurnitureFactory {
2      1 usage new *
3      public Chair createChair() {
4          return new ModernChair();
5      }
6
7      1 usage new *
8      public Sofa createSofa() {
9          return new ModernSofa();
10     }
11
12     1 usage new *
13     public CoffeeTable createCoffeeTable() {
14         return new ModernCoffeeTable();
15     }
16 }
```

ModernSofa.java

```
ModernSofa.java x
1 usage new *
1 class ModernSofa implements Sofa {
  1 usage new *
2 public void lieOn() {
3     System.out.println("Lying on a modern sofa.");
4 }
5 }
6
```

Sofa.java

```
Sofa.java x
1 8 usages 3 implementations new *
1 interface Sofa {
  1 usage 3 implementations new *
2 void lieOn();
3 }
4
```

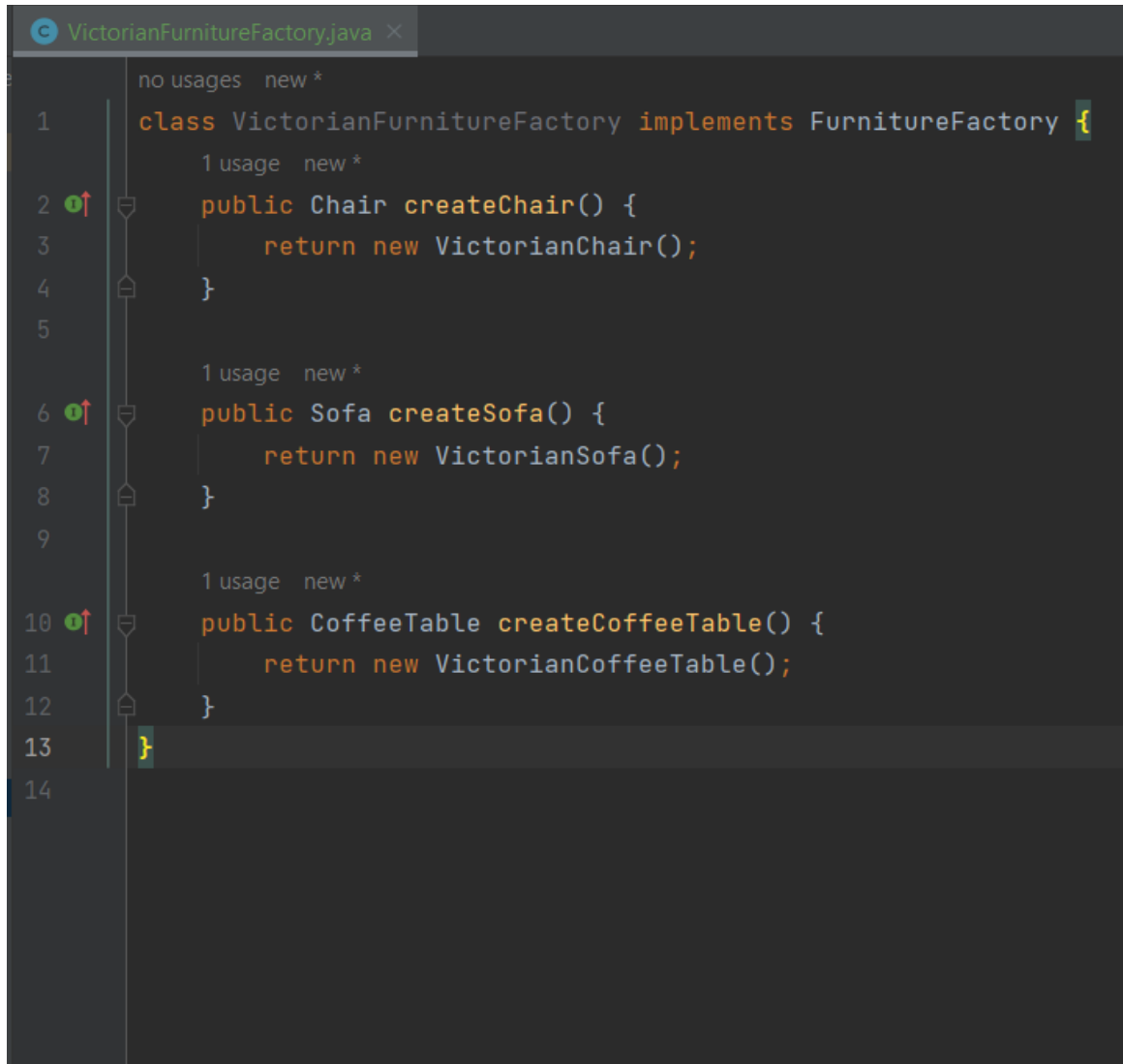
VictorianChair.java

```
VictorianChair.java x
1 usage new *
1 class VictorianChair implements Chair {
  1 usage new *
2 public void sitOn() {
3     System.out.println("Sitting on a Victorian chair.");
4 }
5 }
6
```

VictorianCoffeeTable.java

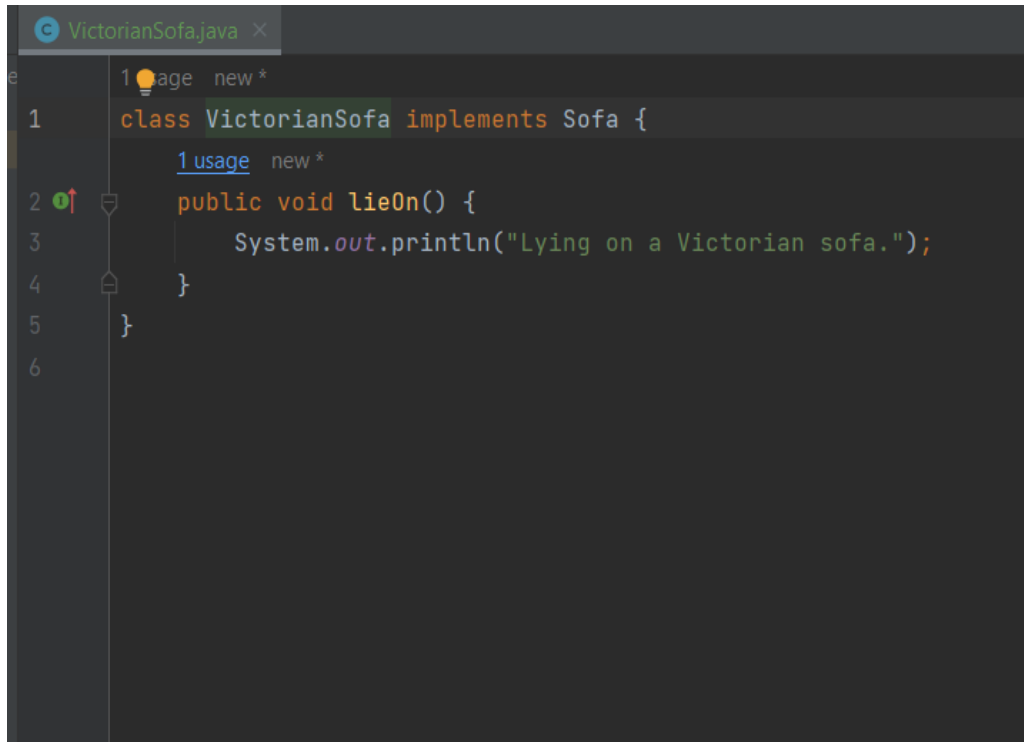
```
VictorianCoffeeTable.java x
1 usage new *
1 class VictorianCoffeeTable implements CoffeeTable {
  1 usage new *
2 public void putOn() {
3     System.out.println("Putting Coffee on a Victorian coffee table.");
4 }
5 }
6
```

VictorianFurnitureFactory.java



```
no usages  new *
1  class VictorianFurnitureFactory implements FurnitureFactory {
2      1 usage  new *
3      public Chair createChair() {
4          return new VictorianChair();
5      }
6
7      1 usage  new *
8      public Sofa createSofa() {
9          return new VictorianSofa();
10     }
11
12     1 usage  new *
13     public CoffeeTable createCoffeeTable() {
14         return new VictorianCoffeeTable();
15     }
16 }
```

VictorianSofa.java



```
1 class VictorianSofa implements Sofa {  
2     public void lieOn() {  
3         System.out.println("Lying on a Victorian sofa.");  
4     }  
5 }  
6
```

Explanation:

Instance of a FurnitureFactory and uses it to create specific furniture objects (a Chair, a Sofa, and a CoffeeTable) from the ModernFurnitureFactory.

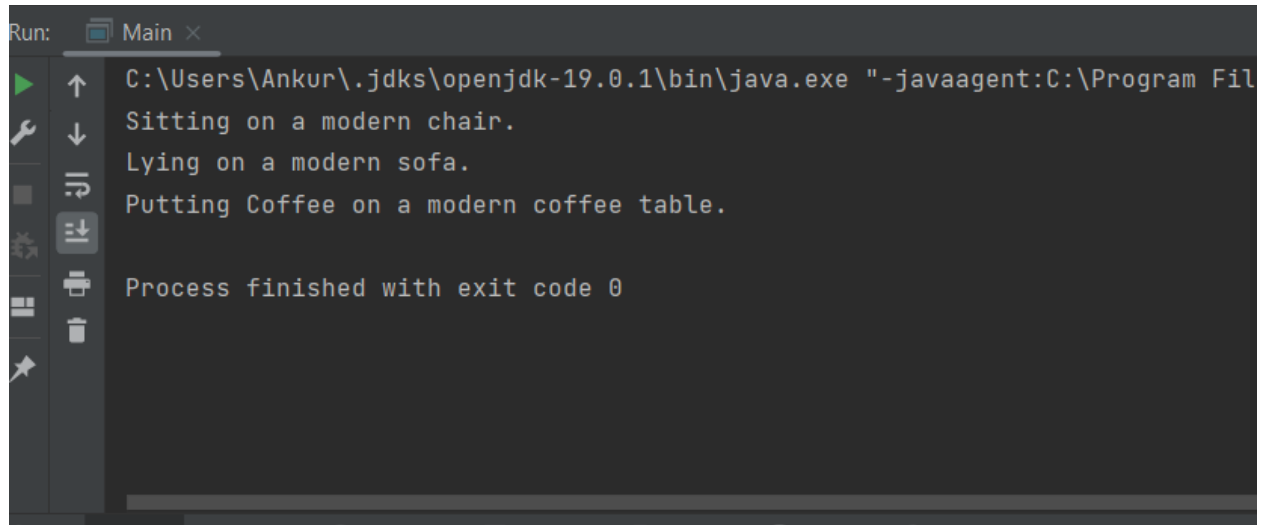
The code begins with the class Main, which is the entry point for the Java application. The public static void main(String[] args) method is the starting point for the program, and it creates an instance of the FurnitureFactory using the ModernFurnitureFactory class.

Next, the code creates specific furniture objects, including a modernChair, modernSofa, and modernCoffeeTable, using the modernFactory object that was previously created. These objects are created by calling the createChair(), createSofa(), and createCoffeeTable() methods respectively, which are defined within the ModernFurnitureFactory class.

Finally, the code calls the sitOn(), lieOn(), and putOn() methods on the Chair, Sofa, and CoffeeTable objects respectively, to demonstrate their functionality.

In summary, this code creates and uses a factory object to create specific instances of furniture objects, and then demonstrates some of their functionality by calling their respective methods.

Output:



The screenshot shows the 'Run' console of an IDE. The title bar indicates the active window is 'Main'. The console output shows the execution of a Java program using the OpenJDK 19.0.1 runtime. The output consists of four lines of text: 'Sitting on a modern chair.', 'Lying on a modern sofa.', 'Putting Coffee on a modern coffee table.', and 'Process finished with exit code 0'. The command line at the top of the console shows the full path to the Java executable and the classpath.

```
Run: Main ×  
C:\Users\Ankur\.jdk\openjdk-19.0.1\bin\java.exe "-javaagent:C:\Program Fil  
Sitting on a modern chair.  
Lying on a modern sofa.  
Putting Coffee on a modern coffee table.  
Process finished with exit code 0
```