CPIT252

Lab Activity #07

Elaf Yousef Aloufi, 1911265 - CAR

Code

BuilderDemo Class (main)

```
13  public class BuilderDemo {
14
15      public static void main(String[] args) throws IOException {
16          try {
17              System.out.print("------------------------Menu-------------------------\n");
18              OrderBuilder orderBuilder = new OrderBuilder();
19              OrderedItems items = orderBuilder.preparePizzaSize();
20              items.showItems();
21              System.out.print("-----------------------------------------------------\n");
22              System.out.println("Total Price: " + items.getCost() + "\n");
23          } catch (InputMismatchException ex) {
24              System.out.println("***Sorrty: InputMismatchException***");
25          }
26      }
27  }
```

OrderBuilder Class

```
14  public class OrderBuilder {
15
16      OrderedItems items = new OrderedItems();
17      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
18      private static int sizeOption;
19
20      public OrderedItems preparePizza(Pizza pizza) throws IOException {
21
22          do {
23              System.out.print("1: Cheeze \n2: Olives \n3: Mushroom \n4: Onion \n5: Done \nChoose toppings for the pizza: ");
24              sizeOption = Integer.parseInt(br.readLine());
25              System.out.println();
26              switch (sizeOption) {
27                  case 1:
28                      pizza = new Cheeze(pizza);
29                      break;
30
31                  case 2:
32                      pizza = new Olives(pizza);
33                      break;
34
35                  case 3:
36                      pizza = new Mushroom(pizza);
37                      break;
38
39                  case 4:
40                      pizza = new Onion(pizza);
41
42                      break;
43                  case 5:
44
45                      continue;
46
47                  default:
48                      System.out.println("Sorry: No option available");
49
50              }//end of switch
```

```
52          } while (sizeOption != 5);
53
54          System.out.print("1: Yes \n2: No \nWould you like to have a drink: ");
55          int drinkOption = Integer.parseInt(br.readLine());
56          System.out.println();
57          switch (drinkOption) {
58              case 1:
59                  preparedDrik();
60                  break;
61              case 2:
62                  System.out.print("\n");
63                  break;
64              default:
65                  System.out.println("\n***Sorry: Your choice entry is not either 1 or 2***");
66                  break;
67          }
68
69          items.addItem(pizza);
70          return items;
71
72      }
73
74      public OrderedItems preparePizzaSize() throws IOException {
75          System.out.print("1: Small Pizza \n2: Medium Pizza \n3: Large Pizza \nChoose a pizza size: ");
76          int size = Integer.parseInt(br.readLine());
77          System.out.println();
78          switch (size) {
79              case 1: {
80                  preparePizza(new SmallPizza());
81
82              }
83              break;
84
85              case 2: {
86                  preparePizza(new MediumPizza());
87
88              }
```

```
89                  break;
90                  case 3: {
91                      preparePizza(new LargePizza());
92                  }
93                  break;
94
95                  default: {
96                      System.out.println("Sorry: No option available");
97                  }
98
99              }//end of switch
100
101             return items;
102
103         }
104
105         public OrderedItems preparedDrik() throws IOException {
106
107             System.out.print("1: Pepsi \n2: Coke \nChoose your drink option: ");
108             int drinkOption = Integer.parseInt(br.readLine());
109             System.out.println();
110             switch (drinkOption) {
111                 case 1:
112                     System.out.print("1: Small \n2: Medium \n2: Large \nChoose a pepsi size: ");
113                     int pepsiSize = Integer.parseInt(br.readLine());
114                     System.out.println();
115                     switch (pepsiSize) {
116                         case 1:
117                             items.addItem(new SmallPepsi());
118                             break;
119                         case 2:
120                             items.addItem(new MediumPepsi());
121                             break;
122                         case 3:
123                             items.addItem(new LargePepsi());
124                             break;
125                     }
126
127                     break;
128                 case 2:
129                     System.out.print("1: Small \n2: Medium \n2: Large \nChoose a coke size: ");
130                     int cokeSize = Integer.parseInt(br.readLine());
131                     System.out.println();
132                     switch (cokeSize) {
133                         case 1:
134                             items.addItem(new SmallCoke());
135                             break;
136                         case 2:
137                             items.addItem(new MediumCoke());
138                             break;
139                         case 3:
140                             items.addItem(new LargeCoke());
141                             break;
142                     }
143                     break;
144
145             }
146
147             return items;
148
149         }
150
151     }
```

OrderedItems Class

```
12  public class OrderedItems {
13
14      ArrayList<Item> items = new ArrayList();
15
16      public void addItem(Item item) {
17          items.add(item);
18      }
19
20      public float getCost() {
21
22          float cost = 0;
23
24          //cost = items.stream().map((item) -> (item.price())).reduce(cost, (accumulator, _item) -> accumulator + _item);
25          for (int i = 0; i < items.size(); i++) {
26              cost += items.get(i).price();
27          }
28
29          return cost;
30      }
31
32      public void showItems() {
            for (Item item : items) {
34              System.out.print("---------------------------------------------------------\n");
35              System.out.println("Item: " + item.name());
36              System.out.println("Item Size: " + item.size());
37              System.out.println("Item price: " + item.price());
38
39          }
40      }
```

Item Interface

```
    public interface Item {
11
        public String name();
13
        public String size();
15
        public float price();
17  }
```

Pizza Class

```
    public abstract class Pizza implements Item {
11
12      @Override
        public abstract float price();
14
15  }
```

SmallPizza Class

```
10  public class SmallPizza extends Pizza {
11
12      @Override
13      public float price() {
14          return 18.0f;
15
16      }
17
18      @Override
19      public String name() {
20          return "Pizza";
21
22      }
23
24      @Override
25      public String size() {
26          return "Small";
27
28      }
```

MediumPizza Class

```
10  public class MediumPizza extends Pizza {
11
12      @Override
13      public float price() {
14          return 21.5f;
15
16      }
17
18      @Override
19      public String name() {
20          return "Pizza";
21
22      }
23
24      @Override
25      public String size() {
26          return "Medium";
27
28      }
```

LargePizza Class

```
10  public class LargePizza extends Pizza {
11
12      @Override
13      public float price() {
14          return 33.25f;
15
16      }
17
18      @Override
19      public String name() {
20          return "Pizza";
21
22      }
23
24      @Override
25      public String size() {
26          return "Large";
27
28      }
```

PizzaDecorator Class

```
10  public abstract class PizzaDecorator extends Pizza {
11
12      private Pizza newPizza;
13
14      public PizzaDecorator(Pizza newPizza) {
15          this.newPizza = newPizza;
16      }
17
18      @Override
19      public float price() {
20          return newPizza.price();
21      }
22
23      @Override
24      public String name() {
25          return newPizza.name();
26      }
27
28      @Override
29      public String size() {
30          return newPizza.size();
31      }
```

Cheeze Class

```
10  public class Cheeze extends PizzaDecorator {
11
12      public Cheeze(Pizza newPizza) {
13          super(newPizza);
14      }
15
16      @Override
17      public float price() {
18          return super.price() + 4.0f;
19
20      }
21
22      @Override
23      public String name() {
24          return super.name() + " (with Cheeze)";
25
26      }
27
28      @Override
29      public String size() {
30          return super.size();
31
32      }
```

Olives Class

```
10  public class Olives extends PizzaDecorator {
11
12      public Olives(Pizza newPizza) {
13          super(newPizza);
14      }
15
16      @Override
        public float price() {
18          return super.price() + 2.5f;
19
20      }
21
22      @Override
        public String name() {
24          return super.name() + " (with Olives)";
25
26      }
27
28      @Override
        public String size() {
30          return super.size();
31
32      }
```

Mushroom Class

```
10  public class Mushroom extends PizzaDecorator {
11
12      public Mushroom(Pizza newPizza) {
13          super(newPizza);
14      }
15
16      @Override
        public float price() {
18          return super.price() + 5;
19
20      }
21
22      @Override
        public String name() {
24          return super.name() + " (with Mushroom)";
25
26      }
27
28      @Override
        public String size() {
30          return super.size();
31
32      }
```

Onion Class

```
10  public class Onion extends PizzaDecorator {
11
12      public Onion(Pizza newPizza) {
13          super(newPizza);
14      }
15
16      @Override
        public float price() {
18          return super.price() + 1.5f;
19
20      }
21
22      @Override
        public String name() {
24          return super.name() + " (with Onion)";
25
26      }
27
28      @Override
        public String size() {
30          return super.size();
31
32      }
```

ColdDrink Class

```
    public abstract class ColdDrink implements Item {
11
12      @Override
        public abstract float price();
14  }
```

Pepsi Class

```
public abstract class Pepsi extends ColdDrink {

    @Override
    public abstract String name();

    @Override
    public abstract String size();

    @Override
    public abstract float price();
```

SmallPepsi Class

```
public class SmallPepsi extends Pepsi {

    @Override
    public String size() {
        return "Small";

    }

    @Override
    public float price() {
        return 3.0f;

    }

    @Override
    public String name() {
        return "Pepsi";
    }
```

MediumPepsi Class

```
10  public class MediumPepsi extends Pepsi {
11
12      @Override
13      public String size() {
14          return "Medium";
15
16      }
17
18      @Override
19      public float price() {
20          return 5.0f;
21
22      }
23
24      @Override
25      public String name() {
26          return "Pepsi";
27      }
```

LargePepsi Class

```
10  public class LargePepsi extends Pepsi {
11
12      @Override
13      public String size() {
14          return "Large";
15
16      }
17
18      @Override
19      public float price() {
20          return 10.25f;
21
22      }
23
24      @Override
25      public String name() {
26          return "Pepsi";
27      }
```

Coke Class

```
public abstract class Coke extends ColdDrink {

    @Override
    public abstract String name();

    @Override
    public abstract String size();

    @Override
    public abstract float price();
```

SmallCoke Class

```
public class SmallCoke extends Coke {

    @Override
    public String name() {
        return "Coke";

    }

    @Override
    public String size() {
        return "Small";

    }

    @Override
    public float price() {
        return 3.25f;

    }
```

MediumCoke Class

```
10  public class MediumCoke extends Coke {
11
12      @Override
13      public String name() {
14          return "Coke";
15
16      }
17
18      @Override
19      public String size() {
20          return "Medium";
21
22      }
23
24      @Override
25      public float price() {
26          return 6.50f;
27
28      }
```

LargeCoke Class

```
10  public class LargeCoke extends Coke {
11
12      @Override
13      public String name() {
14          return "Coke";
15
16      }
17
18      @Override
19      public String size() {
20          return "Large";
21
22      }
23
24      @Override
25      public float price() {
26          return 11.25f;
27
28      }
```

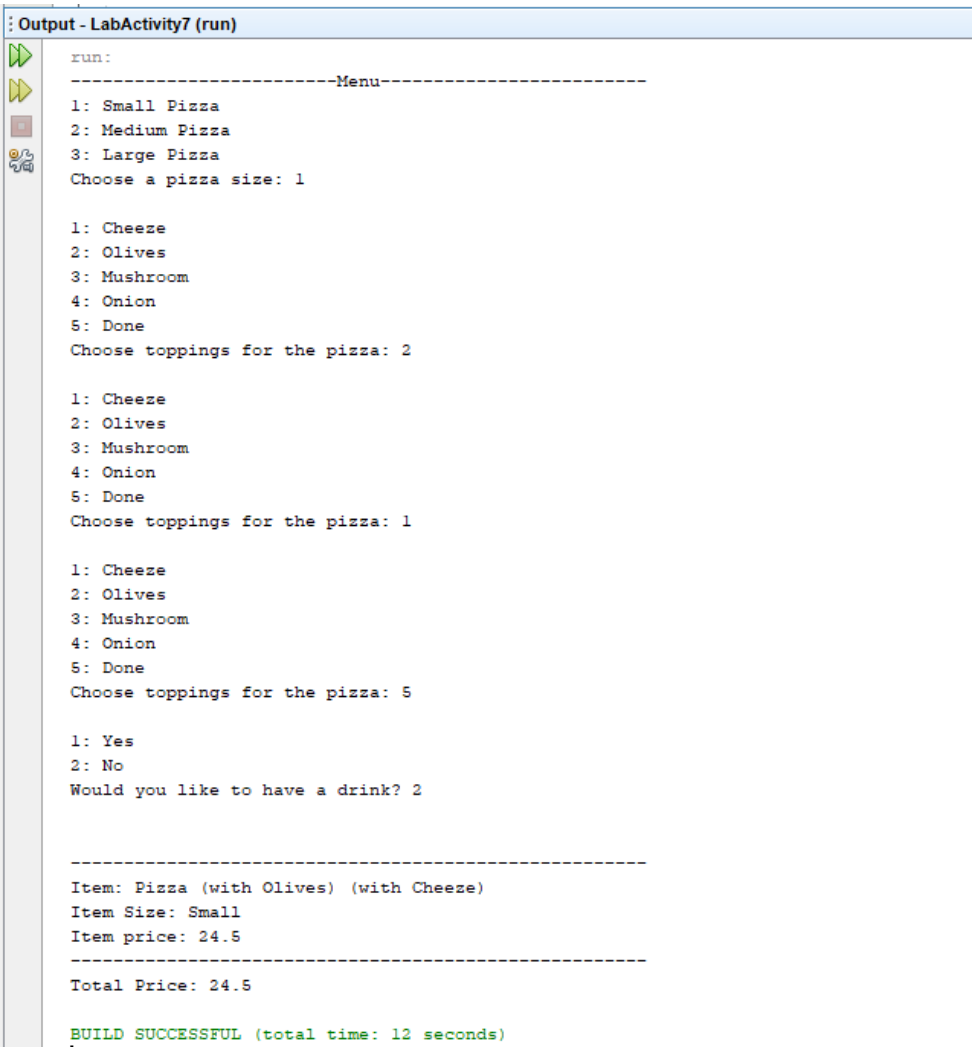
Output


Output - LabActivity7 (run)
run:
-------------------------Menu-------------------------
1: Small Pizza
2: Medium Pizza
3: Large Pizza
Choose a pizza size: 2
1: Cheeze
2: Olives
3: Mushroom
4: Onion
5: Done
Choose toppings for the pizza: 3
1: Cheeze
2: Olives
3: Mushroom
4: Onion
5: Done
Choose toppings for the pizza: 2
1: Cheeze
2: Olives
3: Mushroom
4: Onion
5: Done
Choose toppings for the pizza: 5
1: Yes
2: No
Would you like to have a drink: 1
1: Pepsi
2: Coke
Choose your drink option: 2
1: Small
2: Medium
2: Large
Choose a coke size: 1

Item: Coke
Item Size: Small
Item price: 3.25

Item: Pizza (with Mushroom) (with Olives)
Item Size: Medium
Item price: 29.0

Total Price: 32.25
BUILD SUCCESSFUL (total time: 25 seconds)


Output - LabActivity7 (run)
run:
-------------------------Menu-------------------------
1: Small Pizza
2: Medium Pizza
3: Large Pizza
Choose a pizza size: 1
1: Cheeze
2: Olives
3: Mushroom
4: Onion
5: Done
Choose toppings for the pizza: 2
1: Cheeze
2: Olives
3: Mushroom
4: Onion
5: Done
Choose toppings for the pizza: 1
1: Cheeze
2: Olives
3: Mushroom
4: Onion
5: Done
Choose toppings for the pizza: 5
1: Yes
2: No
Would you like to have a drink? 2

Item: Pizza (with Olives) (with Cheeze)
Item Size: Small
Item price: 24.5

Total Price: 24.5
BUILD SUCCESSFUL (total time: 12 seconds)