



Code

BuildPatternDemo (main)

```
10 public class BuilderPatternDemo {
11
12     public static void main(String[] args) {
13         try {
14             System.out.print("-----Menu-----\n");
15             MealBuilder mealBuilder = new MealBuilder();
16             Meal meal = mealBuilder.buildMeal();
17             meal.showItems();
18             System.out.print("-----\n");
19             System.out.println("Total Price: " + meal.getCost() + "\n");
20         } catch (InputMismatchException ex) {
21             System.out.println("***Sorrty: InputMismatchException***");
22         }
23     }
24 }
```

MealBuilder Class

```
10 public class MealBuilder {
11
12     Scanner input = new Scanner(System.in);
13
14     public Meal buildMeal() {
15
16         Meal meal=new Meal();
17         System.out.print("1: Veggie Burger \n2: Chicken Burger \nChoose your option: ");
18         int burgerOption = input.nextInt();
19         switch (burgerOption) {
20             case 1:
21                 meal=prepareVegMeal();
22                 break;
23             case 2:
24                 meal=prepareVegMeal();
25                 break;
26             default:
27                 System.out.println("\n***Sorry: Your choice entry is not either 1 or 2***");
28                 break;
29         }
30         return meal;
31     }
32 }
```

```

34 public Meal prepareVegMeal() {
35     Meal meal = new Meal();
36     meal.addItem(new VegBurger());
37     System.out.print("1: Yes \n2: No \nWould you like to have a drink: ");
38     int drinkOption = input.nextInt();
39
40     switch (drinkOption) {
41         case 1:
42             prepareDrink(meal);
43             break;
44         case 2:
45             System.out.print("\n");
46             break;
47         default:
48             System.out.println("\n***Sorry: Your choice entry is not either 1 or 2***");
49             break;
50     }
51
52     return meal;
53
54 }

```

```

56 public Meal prepareNunVegMeal() {
57     Meal meal = new Meal();
58     meal.addItem(new ChickenBurger());
59     System.out.print("1: Yes \n2: No \nWould you like to have a drink: ");
60     int drinkOption = input.nextInt();
61
62     switch (drinkOption) {
63         case 1:
64             prepareDrink(meal);
65         case 2:
66             System.out.print("\n");
67             break;
68         default:
69             System.out.println("\n***Sorry: Your choice entry is not either 1 or 2***");
70             break;
71     }
72     return meal;
73
74 }
75
76 public void prepareDrink(Meal meal) {
77
78     System.out.print("1: Pipse \n2: Coke \nChoose your option: ");
79     int drink = input.nextInt();
80     System.out.print("\n");
81     switch (drink) {
82         case 1:
83             meal.addItem(new Pepsi());
84             break;
85         case 2:
86             meal.addItem(new Coke());
87             break;
88     }
89
90 }
91
92 }

```

Meal Class

```
10 public class Meal {
11
12     private final ArrayList<Item> items= new ArrayList();
13
14     public void addItem(Item item) {
15         items.add(item);
16     }
17
18     public float getCost() {
19
20         float cost = 0;
21         for (int i = 0; i < items.size(); i++) {
22             cost += items.get(i).price();
23         }
24
25         return cost;
26     }
27
28     public void showItems() {
29
30         // items.stream().map((item) -> {
31         //     System.out.print("-----\n");
32         //     return item;
33         // }).map((item) -> {
34         //     System.out.println("Item: " + item.name());
35         //     return item;
36         // }).map((item) -> {
37         //     System.out.println("Packing type: " + item.packing().type());
38         //     return item;
39         // }).forEachOrdered((item) -> {
40         //     System.out.println("Item price: " + item.price());
41         // });
42
43         for (Item item : items) {
44             System.out.print("-----\n");
45             System.out.println("Item: " + item.name());
46             System.out.println("Packing type: " + item.packing().type());
47             System.out.println("Item price: " + item.price());
48
49         }
50     }
51 }
52 }
```

Item Class

```
 8 public interface Item {
 9
10     public String name();
11
12     public Packing packing();
13
14     public float price();
15 }
```

Packing Class

```
 8 public interface Packing {
 9
10     public String type();
11
12 }
```

Bottle Class

```
 8 public class Bottle implements Packing {
 9
10     @Override
11     public String type() {
12         return "Bottle";
13     }
14
15 }
```

Wrapper Class

```
 8 public class Wrapper implements Packing {
 9
10     @Override
11     public String type() {
12         return "Wrapper";
13     }
14
15 }
```

Burger Class

```
 9 public abstract class Burger implements Item {  
10     @Override  
    public abstract String name();  
12  
13     @Override  
    public abstract Packing packing();  
15  
16     @Override  
    public abstract float price();  
18  
19 }
```

VegBurger Class

```
 8 public class VegBurger extends Burger {  
 9  
10     Packing packing = new Wrapper();  
11  
12     @Override  
13     public String name() {  
14         return "Veggie Burger";  
15     }  
16  
17  
18     @Override  
19     public Packing packing() {  
20         return packing;  
21     }  
22  
23  
24     @Override  
25     public float price() {  
26         return 20.70f;  
27     }  
28  
29 }
```

ChickenBurger Class

```
8 public class ChickenBurger extends Burger {
9
10     Packing packing = new Wrapper();
11
12     @Override
13     public String name() {
14         return "Chicken Burger";
15     }
16
17     @Override
18     public Packing packing() {
19         return packing;
20     }
21
22     @Override
23     public float price() {
24         return 18.00f;
25     }
26 }
27
28
29
```

ColdDrink Class

```
8 public abstract class ColdDrink implements Item {
9
10     @Override
11     public abstract String name();
12
13     @Override
14     public abstract Packing packing();
15
16     @Override
17     public abstract float price();
18
19 }
```

Pepsi Class

```
8 public class Pepsi extends ColdDrink {
9
10     private final Packing packing = new Bottle();
11
12     @Override
13     public String name() {
14         return "Pepsi";
15     }
16
17     @Override
18     public Packing packing() {
19         return packing;
20     }
21
22     @Override
23     public float price() {
24         return 2.50f;
25     }
26 }
```

Coke Class

```
8 public class Coke extends ColdDrink {
9
10     private final Packing packing;
11
12     public Coke() {
13         this.packing = new Bottle();
14     }
15
16     @Override
17     public String name() {
18         return "Coke";
19     }
20
21     @Override
22     public Packing packing() {
23         return packing;
24     }
25
26     @Override
27     public float price() {
28         return 3.00f;
29     }
30
31 }
```


Output

```
Output - LabActivity5 (run)
run:
-----Menu-----
1: Veggie Burger
2: Chicken Burger
Choose your option: 2
1: Yes
2: No
Would you like to have a drink: 1
1: Pipse
2: Coke
Choose your option: 2

-----
Item: Veggie Burger
Packing type: Wrapper
Item price: 20.7
-----
Item: Coke
Packing type: Bottle
Item price: 3.0
-----
Total Price: 23.7

BUILD SUCCESSFUL (total time: 7 seconds)
|
```

```
Output - LabActivity5 (run)
run:
-----Menu-----
1: Veggie Burger
2: Chicken Burger
Choose your option: 1
1: Yes
2: No
Would you like to have a drink: 2

-----
Item: Veggie Burger
Packing type: Wrapper
Item price: 20.7
-----
Total Price: 20.7

BUILD SUCCESSFUL (total time: 3 seconds)
|
```