Builder Design Pattern

Builder Pattern says that **"construct a complex object from simple objects using step-by-step approach"**

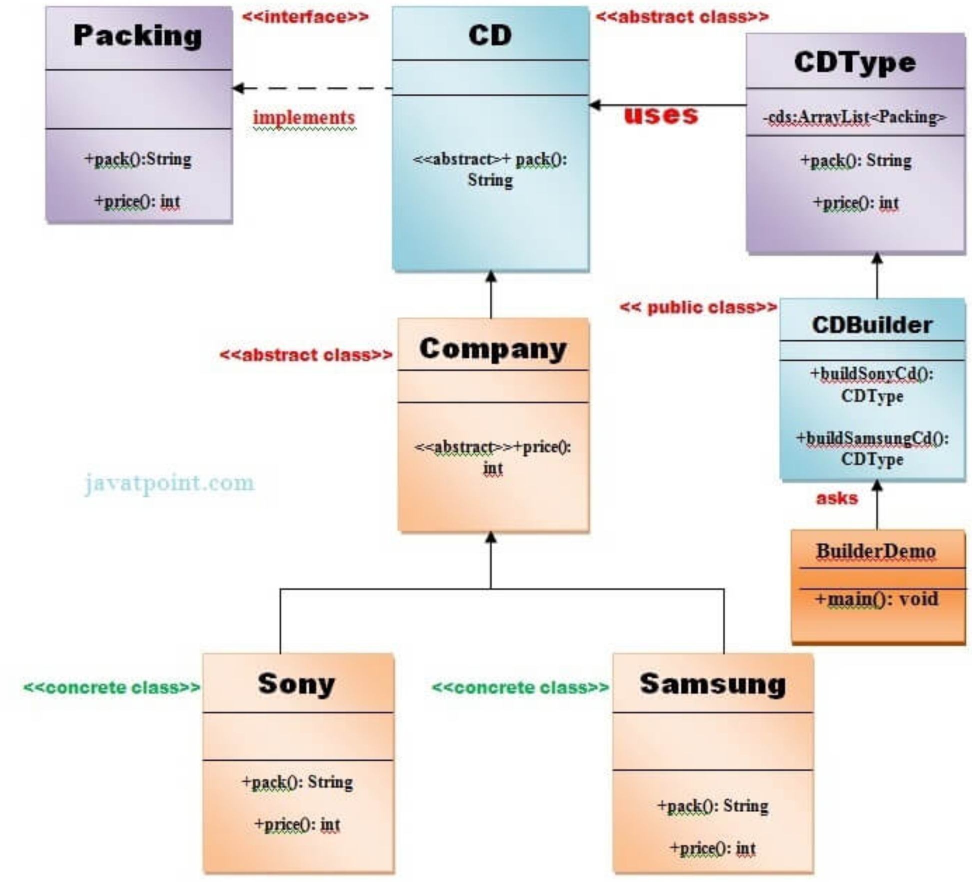
It is mostly used when object can't be created in single step like in the de-serialization of a complex object.

Advantage of Builder Design Pattern

The main advantages of Builder Pattern are as follows:

- It provides clear separation between the construction and representation of an object.
- It provides better control over construction process.
- It supports to change the internal representation of objects.

UML for Builder Pattern Example



Example of Builder Design Pattern

To create simple example of builder design pattern, you need to follow 6 following steps.

1. Create Packing interface
2. Create 2 abstract classes CD and Company
3. Create 2 implementation classes of Company: Sony and Samsung
4. Create the CDType class
5. Create the CDBuilder class
6. Create the BuilderDemo class

1) Create Packing interface

File: Packing.java

```
public interface Packing {
            public String pack();
            public int price();
}
```

2) Create 2 abstract classes CD and Company

Create an abstract class CD which will implement Packing interface.

File: CD.java

```
public abstract class CD implements Packing{
public abstract String pack();
}
```

File: Company.java

```
public abstract class Company extends CD{
   public abstract int price();
}
```

3) Create 2 implementation classes of Company: Sony and Samsung

File: Sony.java

```
public class Sony extends Company{
    @Override
        public int price(){
                        return 20;
      }
    @Override
    public String pack(){
             return "Sony CD";
        }
}//End of the Sony class.
```

File: Samsung.java

```
public class Samsung extends Company {
    @Override
        public int price(){
                        return 15;
    }
    @Override
    public String pack(){
             return "Samsung CD";
        }
}//End of the Samsung class.
```

4) Create the CDType class

File: CDType.java

```
import java.util.ArrayList;
import java.util.List;
public class CDType {
        private List<Packing> items=new ArrayList<Packing>();
        public void addItem(Packing packs) {
            items.add(packs);
        }
        public void getCost(){
         for (Packing packs : items) {
               packs.price();
         }
        }
        public void showItems(){
         for (Packing packing : items){
        System.out.print("CD name : "+packing.pack());
        System.out.println(", Price : "+packing.price());
      }
       }
}//End of the CDType class.
```

5) Create the CDBuilder class

File: CDBuilder.java

```
public class CDBuilder {
           public CDType buildSonyCD(){
             CDType cds=new CDType();
             cds.addItem(new Sony());
             return cds;
         }
         public CDType buildSamsungCD(){
        CDType cds=new CDType();
        cds.addItem(new Samsung());
        return cds;
         }
}// End of the CDBuilder class.
```

6) Create the BuilderDemo class

File: BuilderDemo.java

```
public class BuilderDemo{
 public static void main(String args[]){
  CDBuilder cdBuilder=new CDBuilder();
  CDType cdType1=cdBuilder.buildSonyCD();
  cdType1.showItems();
```

```
    CDType cdType2=cdBuilder.buildSamsungCD();
    cdType2.showItems();
  }
}
```

download this builder pattern example

Output of the above example

```
CD name : Sony CD, Price : 20
CD name : Samsung CD, Price : 15
```

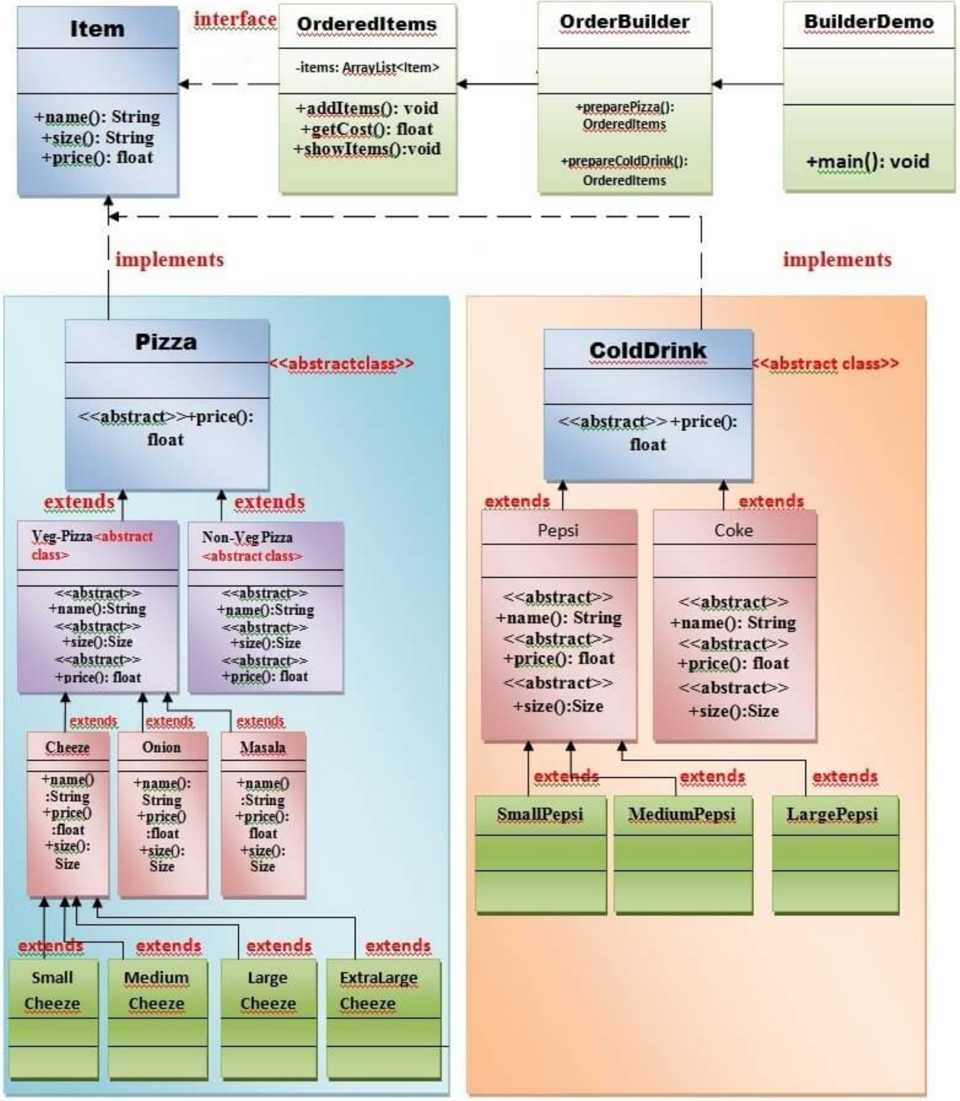
Another Real world example of Builder Pattern

UML for Builder Pattern:

We are considering a business case of **pizza-hut** where we can get different varieties of pizza and cold-drink.

Pizza can be either a Veg pizza or Non-Veg pizza of several types (like cheese pizza, onion pizza, masala-pizza etc) and will be of 4 sizes i.e. small, medium, large, extra-large.

Cold-drink can be of several types (like Pepsi, Coke, Dew, Sprite, Fanta, Maaza, Limca, Thums-up etc.) and will be of 3 sizes small, medium, large.


Item
+name(): String
+size(): String
+price(): float
interface
OrderedItems
-items: ArrayList<Item>
+addItems(): void
+getCost(): float
+showItems():void
OrderBuilder
+preparePizza(): OrderedItems
+prepareColdDrink(): OrderedItems
BuilderDemo
+main(): void
implements
implements
Pizza
<<abstractclass>>
<<abstract>>+price(): float
ColdDrink
<<abstract class>>
<<abstract>> +price(): float
extends
extends
extends
extends
Veg-Pizza<abstract class>
<<abstract>> +name():String <<abstract>> +size():Size <<abstract>> +price(): float
Non-Veg Pizza <abstract class>
<<abstract>> +name():String <<abstract>> +size():Size <<abstract>> +price(): float
Pepsi
<<abstract>> +name(): String <<abstract>> +price(): float <<abstract>> +size():Size
Coke
<<abstract>> +name(): String <<abstract>> +price(): float <<abstract>> +size():Size
extends
extends
extends
Cheeze
+name() :String +price() :float +size(): Size
Onion
+name(): String +price() :float +size(): Size
Masala
+name() :String +price(): float +size(): Size
extends
extends
extends
SmallPepsi
MediumPepsi
LargePepsi
extends
extends
extends
extends
Small Cheeze
Medium Cheeze
Large Cheeze
ExtraLarge Cheeze

Real world example of builder pattern

Let's see the step by step real world example of Builder Design Pattern.

Step 1:**Create an interface Item that represents the Pizza and Cold-drink.**

File: Item.java

```
public  interface  Item
{
  public String name();
   public String size();
   public float price();
}// End of the interface Item.
```

Step 2:**Create an abstract class Pizza that will implement to the interface Item.**

File: Pizza.java

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

Step 3:**Create an abstract class ColdDrink that will implement to the interface Item.**

File: ColdDrink.java

```
public abstract class ColdDrink implements Item{
 @Override
 public abstract float price();
```

Step 4:**Create an abstract class VegPizza that will extend to the abstract class Pizza.**

File: VegPizza.java

```
public abstract class VegPizza extends Pizza{
    @Override
    public abstract float price();
    @Override
    public abstract  String name();
    @Override
    public abstract  String size();
}// End of the abstract class VegPizza.
```

Step 5:**Create an abstract class NonVegPizza that will extend to the abstract class Pizza.**

File: NonVegPizza.java

```
public abstract class NonVegPizza extends Pizza{
    @Override
    public abstract float price();
    @Override
    public abstract String name();
    @Override
    public abstract String size();
}// End of the abstract class NonVegPizza.
```

Step 6:**Now, create concrete sub-classes SmallCheezePizza, MediumCheezePizza, LargeCheezePizza, ExtraLargeCheezePizza that will extend to the abstract class VegPizza.**

File: SmallCheezePizza.java

```
public class SmallCheezePizza extends VegPizza{
    @Override
    public float price() {
        return 170.0f;
    }
    @Override
    public String name() {
        return "Cheeze Pizza";
    }
    @Override
    public String size() {
       return "Small size";
    }
}// End of the SmallCheezePizza class.
```

File: MediumCheezePizza.java

```
public class MediumCheezePizza extends VegPizza{
    @Override
    public float price() {
       return  220.f;
    }
    @Override
    public String name() {
        return "Cheeze Pizza";
    }
    @Override
    public String size() {
     return "Medium Size";
 }
}// End of the MediumCheezePizza class.
</textaera></div>

<div id="filename">File: LargeCheezePizza.java</div>
<div class="codeblock"><textarea  name="code" class="java">
public class LargeCheezePizza extends VegPizza{
    @Override
    public float price() {
        return 260.0f;
    }
    @Override
    public String name() {
        return "Cheeze Pizza";
    }
    @Override
    public String size() {
        return "Large Size";
    }
}// End of the LargeCheezePizza class.
```

File: ExtraLargeCheezePizza.java

```
public class ExtraLargeCheezePizza extends VegPizza{
    @Override
    public float price() {
        return 300.f;
    }
    @Override
    public String name() {
        return  "Cheeze Pizza";
    }
    @Override
    public String size() {
        return "Extra-Large Size";
    }
}// End of the ExtraLargeCheezePizza class.
```

Step 7:**Now, similarly create concrete sub-classes SmallOnionPizza, MediumOnionPizza, LargeOnionPizza, ExtraLargeOnionPizza that will extend to the abstract class VegPizza.**

File: SmallOnionPizza.java

```
public class SmallOnionPizza extends VegPizza {
    @Override
    public float price() {
        return 120.0f;
    }
    @Override
    public String name() {
        return  "Onion Pizza";
    }
    @Override
    public String size() {
       return  "Small Size";
    }
}// End of the SmallOnionPizza class.
```

File: MediumOnionPizza.java

```
public class MediumOnionPizza extends VegPizza {
    @Override
    public float price() {
        return 150.0f;
    }
   @Override
    public String name() {
        return  "Onion Pizza";
    }
   @Override
    public String size() {
       return  "Medium Size";
    }
}// End of the MediumOnionPizza class.
```

File: LargeOnionPizza.java

```
public class LargeOnionPizza extends  VegPizza{
    @Override
    public float price() {
        return 180.0f;
    }
    @Override
    public String name() {
         return "Onion Pizza";
    }
    @Override
    public String size() {
       return  "Large size";
    }
}// End of the LargeOnionPizza class.
```

File: ExtraLargeOnionPizza.java

```
public class ExtraLargeOnionPizza extends VegPizza {
    @Override
    public float price() {
        return 200.0f;
    }
    @Override
    public String name() {
         return  "Onion Pizza";
    }
    @Override
    public String size() {
       return  "Extra-Large Size";
    }
}// End of the ExtraLargeOnionPizza class
```

Step 8:**Now, similarly create concrete sub-classes SmallMasalaPizza, MediumMasalaPizza, LargeMasalaPizza, ExtraLargeMasalaPizza that will extend to the abstract class VegPizza.**

File: SmallMasalaPizza.java

```
public class SmallMasalaPizza extends VegPizza{
    @Override
    public float price() {
        return 100.0f;
    }
    @Override
    public String name() {
        return  "Masala Pizza";
    }
    @Override
    public String size() {
       return  "Samll Size";
    }
}// End of the SmallMasalaPizza class
```

File: MediumMasalaPizza.java

```
public class MediumMasalaPizza extends VegPizza {

    @Override
    public float price() {
        return 120.0f;
    }

    @Override
    public String name() {

        return  "Masala Pizza";

    }

    @Override
    public String size() {
       return  "Medium Size";
    }
```

File: LargeMasalaPizza.java

```
public class LargeMasalaPizza extends  VegPizza{
    @Override
    public float price() {
        return 150.0f;
    }

    @Override
    public String name() {

        return  "Masala Pizza";

    }

    @Override
    public String size() {
       return  "Large Size";
    }
} //End of the LargeMasalaPizza class
```

File: ExtraLargeMasalaPizza.java

```
public class ExtraLargeMasalaPizza extends VegPizza {
    @Override
    public float price() {
        return 180.0f;
    }

    @Override
    public String name() {
        return  "Masala Pizza";

    }

    @Override
    public String size() {
       return  "Extra-Large Size";
    }
}// End of the ExtraLargeMasalaPizza class
```

Step 9:**Now, create concrete sub-classes SmallNonVegPizza, MediumNonVegPizza, LargeNonVegPizza, ExtraLargeNonVegPizza that will extend to the abstract class NonVegPizza.**

File: SmallNonVegPizza.java

```
public class SmallNonVegPizza extends NonVegPizza {

    @Override
    public float price() {
        return 180.0f;
    }

    @Override
    public String name() {
       return "Non-Veg Pizza";
    }

    @Override
    public String size() {
        return "Samll Size";
    }

}// End of the SmallNonVegPizza class
```

File: MediumNonVegPizza.java

```
public class MediumNonVegPizza extends NonVegPizza{

    @Override
    public float price() {
        return 200.0f;
    }

    @Override
    public String name() {
       return "Non-Veg Pizza";
    }

    @Override
    public String size() {
        return "Medium Size";
    }
```

File: LargeNonVegPizza.java

```
public class LargeNonVegPizza extends NonVegPizza{


    @Override
    public float price() {
        return 220.0f;
    }

    @Override
    public String name() {
       return "Non-Veg Pizza";
    }

    @Override
    public String size() {
        return "Large Size";
    }

}// End of the LargeNonVegPizza class
```

File: ExtraLargeNonVegPizza.java

```
public class ExtraLargeNonVegPizza extends NonVegPizza {
    @Override
    public float price() {
        return 250.0f;
    }

    @Override
    public String name() {
       return "Non-Veg Pizza";
    }

    @Override
    public String size() {
        return "Extra-Large Size";
    }

}

 // End of the ExtraLargeNonVegPizza class
```

Step 10:**Now, create two abstract classes Pepsi and Coke that will extend abstract class ColdDrink.**

File: Pepsi.java

```
public abstract class Pepsi extends ColdDrink {

    @Override
    public abstract  String name();

    @Override
    public abstract  String size();

    @Override
    public abstract  float price();

}// End of the Pepsi class
```

File: Coke.java

```
public abstract class Coke  extends ColdDrink {

    @Override
    public abstract  String name();

    @Override
    public abstract  String size();

    @Override
    public abstract  float price();

}// End of the Coke class

</textaea></div>

<p>Step 11:<b>Now, create concrete sub-
classes SmallPepsi, MediumPepsi, LargePepsi that will extend to the abstract class Pepsi.
</b></p>
<div id="filename">File: SmallPepsi.java</div>
<div class="codeblock"><textarea  name="code" class="java">
public class SmallPepsi  extends Pepsi{

    @Override
    public String name() {
       return "300 ml Pepsi";
    }

    @Override
    public float price() {
        return 25.0f;
    }
    @Override
    public String size() {
        return "Small Size";
    }
}// End of the SmallPepsi class
```

File: MediumPepsi.java

```
public class MediumPepsi extends Pepsi {

    @Override
    public String name() {
       return "500 ml Pepsi";
    }

    @Override
    public String size() {
        return "Medium Size";
    }

    @Override
    public float price() {
        return 35.0f;
    }
}// End of the MediumPepsi class
```

File: LargePepsi.java

```
public class LargePepsi extends Pepsi{
    @Override
    public String name() {
       return "750 ml Pepsi";
    }

    @Override
    public String size() {
        return "Large Size";
    }

    @Override
    public float price() {
        return 50.0f;
    }
}// End of the LargePepsi class
```

Step 12:**Now, create concrete sub-classes SmallCoke, MediumCoke, LargeCoke that will extend to the abstract class Coke.**

File: SmallCoke.java

```
public class SmallCoke extends Coke{
    @Override
    public String name() {
         return "300 ml Coke";
    }

    @Override
    public String size() {

        return "Small Size";
    }

    @Override
    public float price() {

        return  25.0f;
    }
 }// End of the SmallCoke class
```

File: MediumCoke.java

```
public class MediumCoke extends Coke{

    @Override
    public String name() {
         return "500 ml Coke";
    }

    @Override
    public String size() {

        return "Medium Size";
    }

    @Override
    public float price() {

        return  35.0f;
   }
}// End of the MediumCoke class
```

File: LargeCoke.java

```
public class LargeCoke extends Coke {
     @Override
    public String name() {
         return "750 ml Coke";
    }

    @Override
    public String size() {

        return "Large Size";
    }

    @Override
    public float price() {

        return  50.0f;
    }
}// End of the LargeCoke class

</textrea></div>

<p>Step 13:
<b>Create an OrderedItems class that are having Item objects defined above.</b>
</p>
<div id="filename">File: OrderedItems.java</div>
<div class="codeblock"><textarea  name="code" class="java">
import java.util.ArrayList;
import java.util.List;
public class OrderedItems {

    List<Item> items=new ArrayList<Item>();

    public void addItems(Item item){

        items.add(item);
    }
    public float getCost(){

        float cost=0.0f;
            for (Item item : items) {
                cost+=item.price();
             }
        return cost;
    }
    public void showItems(){

        for (Item item : items) {
            System.out.println("Item is:" +item.name());
            System.out.println("Size is:" +item.size());
            System.out.println("Price is: " +item.price());

        }
    }

}// End of the OrderedItems class
```

Step 14:**Create an OrderBuilder class that will be responsible to create the objects of OrderedItems class.**

File: OrdereBuilder.java

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class OrderBuilder {
    public OrderedItems preparePizza() throws IOException{

        OrderedItems itemsOrder=new OrderedItems();
         BufferedReader br =new BufferedReader(new InputStreamReader(System.in));

        System.out.println(" Enter the choice of Pizza ");
        System.out.println("============================");
        System.out.println("        1. Veg-Pizza       ");
        System.out.println("        2. Non-Veg Pizza   ");
        System.out.println("        3. Exit            ");
        System.out.println("============================");

        int pizzaandcolddrinkchoice=Integer.parseInt(br.readLine());
        switch(pizzaandcolddrinkchoice)
        {
           case 1:{

                    System.out.println("You ordered Veg Pizza");
                    System.out.println("\n\n");
                    System.out.println(" Enter the types of Veg-Pizza ");
                    System.out.println("------------------------------");
                    System.out.println("        1.Cheeze Pizza        ");
                    System.out.println("        2.Onion Pizza        ");
                    System.out.println("        3.Masala Pizza        ");
                    System.out.println("        4.Exit            ");
                    System.out.println("------------------------------");
                                int vegpizzachoice=Integer.parseInt(br.readLine());
                    switch(vegpizzachoice)
                    {
                         case 1:
                               {
                                 System.out.println("You ordered Cheeze Pizza");

                                 System.out.println("Enter the cheeze pizza size");
                                 System.out.println("------------------------------------");
                                 System.out.println("    1. Small Cheeze Pizza ");
                                 System.out.println("    2. Medium Cheeze Pizza ");
                                 System.out.println("    3. Large Cheeze Pizza ");
                                 System.out.println("    4. Extra-Large Cheeze Pizza ");
         System.out.println("-------------------------------------");
         int cheezepizzasize=Integer.parseInt(br.readLine());
         switch(cheezepizzasize)
            {
                 case 1:
                     itemsOrder.addItems(new SmallCheezePizza());
                     break;
                case 2:
                     itemsOrder.addItems(new MediumCheezePizza());
                     break;
                case 3:
                     itemsOrder.addItems(new LargeCheezePizza());
                     break;
                case 4:
                     itemsOrder.addItems(new ExtraLargeCheezePizza());
                     break;
case 2:
     {
         System.out.println("You ordered Onion Pizza");
         System.out.println("Enter the Onion pizza size");
         System.out.println("-----------------------------------");
         System.out.println("    1. Small Onion Pizza ");
         System.out.println("    2. Medium Onion Pizza ");
         System.out.println("    3. Large Onion Pizza ");
         System.out.println("    4. Extra-Large Onion Pizza ");
         System.out.println("-----------------------------------");
         int onionpizzasize=Integer.parseInt(br.readLine());
         switch(onionpizzasize)
                 {
                    case 1:
                     itemsOrder.addItems(new SmallOnionPizza());
                     break;

                    case 2:
                     itemsOrder.addItems(new MediumOnionPizza());
                     break;

                    case 3:
                     itemsOrder.addItems(new LargeOnionPizza());
                     break;

                    case 4:
                     itemsOrder.addItems(new ExtraLargeOnionPizza());
                     break;

                 }
         }
```

```
                    break;
              case 3:
                    {
                       System.out.println("You ordered Masala Pizza");
                       System.out.println("Enter the Masala pizza size");
                       System.out.println("------------------------------------");
                       System.out.println("    1. Small Masala Pizza ");
                       System.out.println("    2. Medium Masala Pizza ");
                       System.out.println("    3. Large Masala Pizza ");
                       System.out.println("    4. Extra-Large Masala Pizza ");
                       System.out.println("------------------------------------");
                                     int masalapizzasize=Integer.parseInt(br.readLine());
                           switch(masalapizzasize)
                                 {
                                     case 1:
                                      itemsOrder.addItems(new SmallMasalaPizza());
                                      break;

                                     case 2:
                                      itemsOrder.addItems(new MediumMasalaPizza());
                                      break;

                                     case 3:
                                      itemsOrder.addItems(new LargeMasalaPizza());
                                      break;

                                     case 4:
                                      itemsOrder.addItems(new ExtraLargeMasalaPizza());
                                      break;

                                 }

                    }
                    break;

         }

       }
       break;// Veg- pizza choice completed.

case 2:
       {
           System.out.println("You ordered Non-Veg Pizza");
           System.out.println("\n\n");

                       System.out.println("Enter the Non-Veg pizza size");
                       System.out.println("------------------------------------");
```

```
                     System.out.println("    1. Small Non-Veg  Pizza ");
                     System.out.println("    2. Medium Non-Veg  Pizza ");
                     System.out.println("    3. Large Non-Veg  Pizza ");
                     System.out.println("    4. Extra-Large Non-Veg  Pizza ");
                     System.out.println("------------------------------------");


             int nonvegpizzasize=Integer.parseInt(br.readLine());

              switch(nonvegpizzasize)
                 {

                    case 1:
                      itemsOrder.addItems(new SmallNonVegPizza());
                      break;

                    case 2:
                      itemsOrder.addItems(new MediumNonVegPizza());
                      break;

                    case 3:
                      itemsOrder.addItems(new LargeNonVegPizza());
                      break;

                    case 4:
                      itemsOrder.addItems(new ExtraLargeNonVegPizza());
                      break;
                 }

             }
              break;
      default:
      {
         break;

      }

}//end of main Switch

           //continued?..
    System.out.println(" Enter the choice of ColdDrink ");
    System.out.println("============================");
    System.out.println("        1. Pepsi             ");
    System.out.println("        2. Coke              ");
    System.out.println("        3. Exit             ");
    System.out.println("============================");
           int coldDrink=Integer.parseInt(br.readLine());
```

```
switch (coldDrink)
  {
    case 1:
            {
              System.out.println("You ordered Pepsi ");
              System.out.println("Enter the  Pepsi Size ");
              System.out.println("------------------------");
              System.out.println("    1. Small Pepsi ");
              System.out.println("    2. Medium Pepsi ");
              System.out.println("    3. Large Pepsi ");
              System.out.println("------------------------");
                    int pepsisize=Integer.parseInt(br.readLine());
              switch(pepsisize)
                    {
                      case 1:
                       itemsOrder.addItems(new SmallPepsi());
                       break;

                      case 2:
                       itemsOrder.addItems(new MediumPepsi());
                       break;

                      case 3:
                       itemsOrder.addItems(new LargePepsi());
                       break;

                     }
             }
            break;
         case 2:
            {
              System.out.println("You ordered Coke");
              System.out.println("Enter the Coke Size");
              System.out.println("------------------------");
              System.out.println("    1. Small Coke ");
              System.out.println("    2. Medium Coke  ");
              System.out.println("    3. Large Coke  ");
              System.out.println("    4. Extra-Large Coke ");
              System.out.println("------------------------");

              int cokesize=Integer.parseInt(br.readLine());
              switch(cokesize)
                    {
                      case 1:
                       itemsOrder.addItems(new SmallCoke());
                       break;
```

```
                                                    case 2:
                                                     itemsOrder.addItems(new MediumCoke());
                                                     break;

                                                    case 3:
                                                     itemsOrder.addItems(new LargeCoke());
                                                     break;



                                                }

                                  }
                                  break;
                      default:
                                        {
                                                     break;


                                        }

                                }//End of the Cold-Drink switch
                   return itemsOrder;

           } //End of the preparePizza() method
```

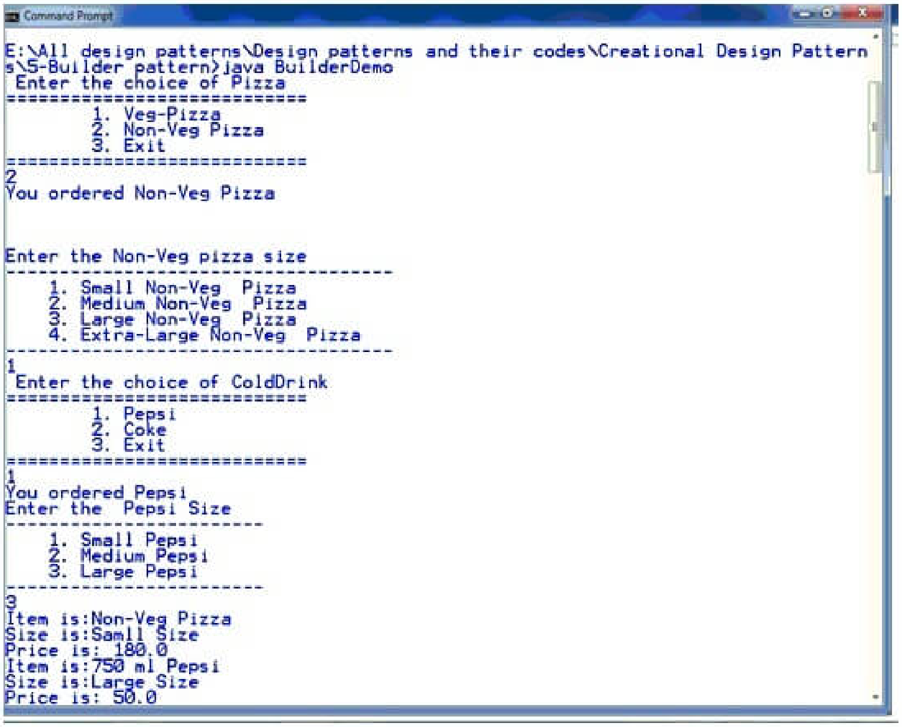
Step 15:**Create a BuilderDemo class that will use the OrderBuilder class.**

File: Prototype.java

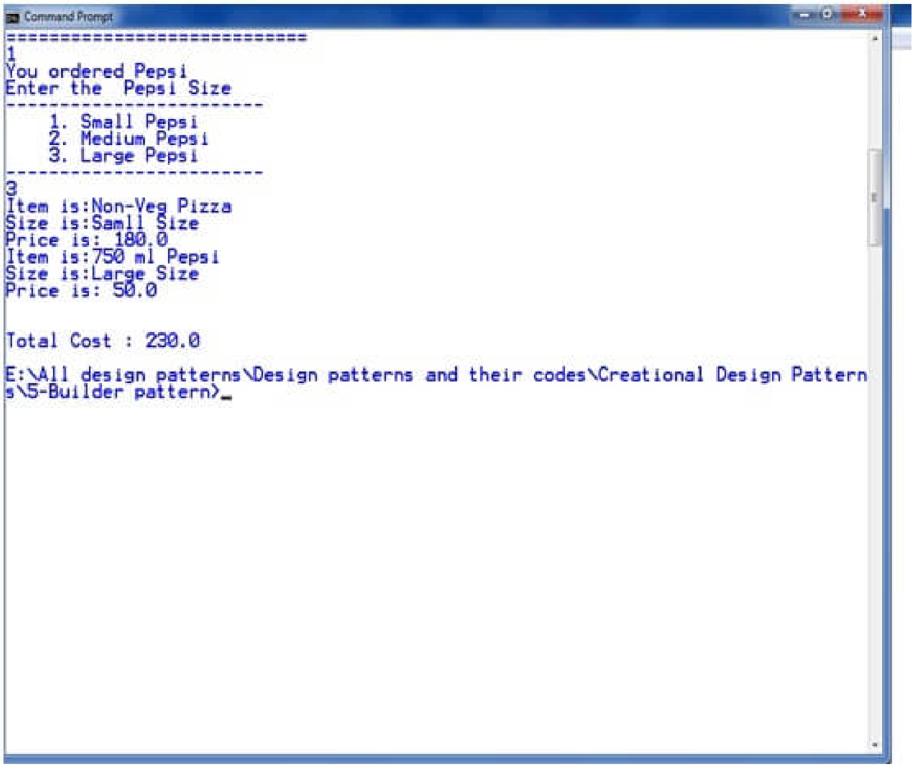
```
import java.io.IOException;
public class BuilderDemo {

    public static void main(String[] args) throws IOException {
        // TODO code application logic here

        OrderBuilder builder=new OrderBuilder();

        OrderedItems orderedItems=builder.preparePizza();

        orderedItems.showItems();

        System.out.println("\n");
        System.out.println("Total Cost : "+ orderedItems.getCost());

    }
}// End of the BuilderDemo class
```

download this Builder Pattern Example

Output



```
E:\All design patterns\Design patterns and their codes\Creational Design Pattern
s\5-Builder pattern>java BuilderDemo
 Enter the choice of Pizza
============================
        1. Veg-Pizza
        2. Non-Veg Pizza
        3. Exit
============================
2
You ordered Non-Veg Pizza


Enter the Non-Veg pizza size
------------------------------------
    1. Small Non-Veg  Pizza
    2. Medium Non-Veg  Pizza
    3. Large Non-Veg  Pizza
    4. Extra-Large Non-Veg  Pizza
------------------------------------
1
 Enter the choice of ColdDrink
============================
        1. Pepsi
        2. Coke
        3. Exit
============================
1
You ordered Pepsi
Enter the  Pepsi Size
------------------------
    1. Small Pepsi
    2. Medium Pepsi
    3. Large Pepsi
------------------------
3
Item is:Non-Veg Pizza
Size is:Samll Size
Price is: 180.0
Item is:750 ml Pepsi
Size is:Large Size
Price is: 50.0
```



```
============================
1
You ordered Pepsi
Enter the  Pepsi Size
------------------------
    1. Small Pepsi
    2. Medium Pepsi
    3. Large Pepsi
------------------------
3
Item is:Non-Veg Pizza
Size is:Samll Size
Price is: 180.0
Item is:750 ml Pepsi
Size is:Large Size
Price is: 50.0

Total Cost : 230.0

E:\All design patterns\Design patterns and their codes\Creational Design Pattern
s\5-Builder pattern>_
```

← prev next →

Help Others, Please Share

Join Javatpoint Test Series

Placement Papers
TCS
HCL
Infosys
IBM
Accenture

AMCAT
eLitmas
Java
Python
C Programming
Networking

Bank PO/Clerk
UPSSSC
Government Exams
SSC
Civil Services
SBI

GATE
NEET
CAT
Railway
CTET
IIT JEE

Learn Latest Tutorials

Ansible

Mockito

Talend

Azure

SharePoint

Powershell

Kali Linux Tutorial

Kali Linux

OpenCV Tutorial

OpenCV

Kafka Tutorial

Kafka

Pandas Tutorial

Pandas

Joomla Tutorial

Joomla

Reinforcement Learning Tutorial

Reinforcement

Preparation

Aptitude

Logical Reasoning

Verbal Ability

Interview Questions

⇧

Aptitude

Reasoning

Verbal A.

Interview

Company Interview Questions

Company

Trending Technologies

Artificial Intelligence Tutorial

AI

AWS Tutorial

AWS

Selenium tutorial

Selenium

Cloud tutorial

Cloud

Hadoop tutorial

Hadoop

ReactJS Tutorial

ReactJS

Data Science Tutorial

D. Science

Angular 7 Tutorial

Angular 7

Blockchain Tutorial

Blockchain

Git Tutorial

Git

Machine Learning Tutorial

ML

DevOps Tutorial

DevOps

B.Tech / MCA

DBMS tutorial

DBMS

Data Structures tutorial

DS

DAA tutorial

DAA

Operating System tutorial

OS

Computer Network tutorial

C. Network

Compiler Design tutorial

Compiler D.

Computer Organization and Architecture

COA

Discrete Mathematics Tutorial

D. Math.

Ethical Hacking Tutorial

E. Hacking

Computer Graphics Tutorial

C. Graphics

Software Engineering Tutorial

Software E.

html tutorial

Web Tech.

⇧

Cyber Security tutorial

Cyber Sec.

Automata Tutorial

Automata

C Language tutorial

C

C++ tutorial

C++

Java tutorial

Java

.Net Framework tutorial

.Net

Python tutorial

Python

List of Programs

Programs

Control Systems tutorial

Control S.

Data Mining Tutorial

Data Mining

⇧