STEP NO1 : Create an interface Item representing food item and packing.

```
package BuilderPatternDemo;

public interface item {

   String getName();

   Packing getPacking();

   float getPrice();}
```

```
package BuilderPatternDemo;

public interface Packing {

   String pack();

}
```

STEP NO2: Create concrete classes implementing the Packing interface.

```
public class Wrapper implements Packing {

            @Override

            public String pack() {

            return "Paper Wrap";

}
```

```
public class Bottle implements Packing {

   @Override

   public String pack() {

      return "Plastic Bottle"; }
```

STEP NO3: Create abstract classes implementing the item interface providing default functionalities.

```
public abstract class Drink implements item {

   @Override

   public Packing getPacking() {

      return new Bottle(); }

}}
```

```
public abstract class Sandwich implements item {

   @Override

   public Packing getPacking() {

      return new Wrapper();}

}
```

STEP NO4: Create concrete classes extending SandWich and Drink classes

```
public class GrilledSandwich extends Sandwich {

   @Override

   public float getPrice() { return 45.0f;}

@Override

   public String getName() {

      return "Grilled Sandwich";    }}

public class VeggieeSandwich extends Sandwich {

   @Override

   public float getPrice() {return 20.0f;}

   @Override

   public String getName() {

      return "Veggie Sandwich"; }

}
```

```
public class OrangeJuice extends Drink {

   @Override

   public float getPrice() {return 25.0f;}

   @Override

   public String getName() {

      return "Orange Juice";}

public class Water extends Drink {

   @Override

   public float getPrice() {

      return 15.0f;}

   @Override

   public String getName() {

      return "Water";}}
```

STEP NO5: Create a MealOrder class having Item objects defined above.

```
import java.util.ArrayList;
import java.util.List;
public class MealOrder {
   private List<item> items = new ArrayList<>();
   public void addItem(item item) {
      items.add(item);}
   public float getCost() {
      float cost = 0.0f;
      for (item item : items) {
         cost += item.getPrice();
      }
      return cost;
   }
```

```
   public void showItems() {
      for (item item : items) {
         System.out.println("Item: " + item.getName());
         System.out.println("Packaging: " + item.getPacking().pack());
         System.out.println("Price: " + item.getPrice());
      }
   }
}
   @Override
   public String getName() {
      return "Grilled Sandwich";  }
}
```

STEP NO6: Create a MealBuilder class, the actual builder class responsible to create Meal objects.

```
import java.util.Scanner;
public class MealBuilder {
   public MealOrder prepareOrder() {
      MealOrder meal = new MealOrder();
      Scanner sc = new Scanner(System.in);
      System.out.println("Choose a sandwich: 1. Veggie Sandwich 2. Grilled
Sandwich");
      int sandwichChoice = sc.nextInt();
      if (sandwichChoice == 1) {
         meal.addItem(new VeggieeSandwich());
      } else if (sandwichChoice == 2) {
         meal.addItem(new GrilledSandwich());
      }
      System.out.println("Choose a drink: 1. Orange Juice 2. Water");
      int drinkChoice = sc.nextInt();
      if (drinkChoice == 1) {
meal.addItem(new OrangeJuice());
      }
```

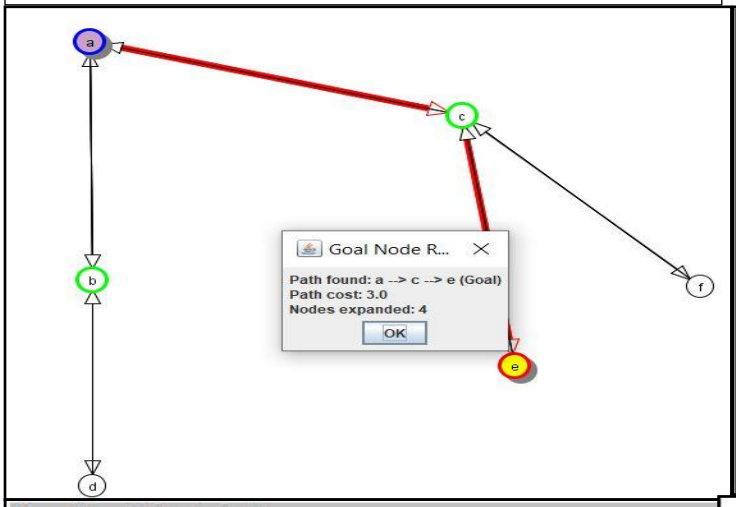
```
       else if (drinkChoice == 2) {
         meal.addItem(new Water());}
      sc.close();
      return meal;
   }
}
   public void showItems() {
      for (item item : items) {
         System.out.println("Item: " + item.getName());
         System.out.println("Packaging: " +
item.getPacking().pack());
         System.out.println("Price: " + item.getPrice());
      }}}
   @Override
   public String getName() {
      return "Grilled Sandwich";
   }}
```

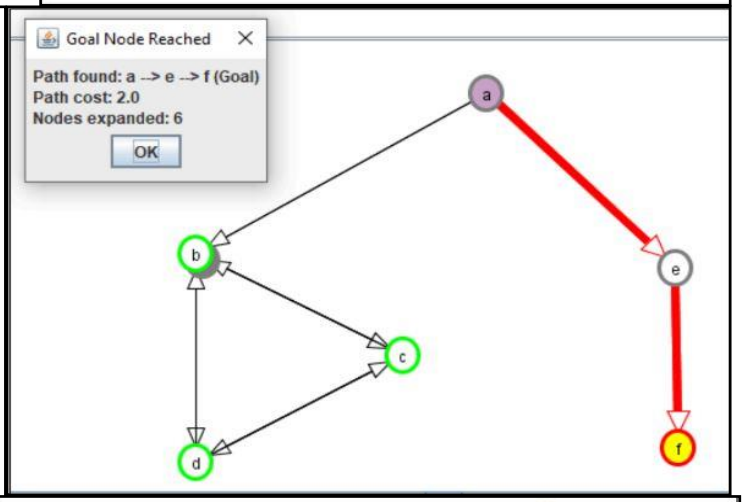
STEP NO7: MealOrderApp uses MealBuider to demonstrate builder pattern.

```
public class MealOrderApp {

    public static void main(String[] args) {

        MealBuilder mealBuilder = new MealBuilder();

        MealOrder meal = mealBuilder.prepareOrder();

        System.out.println("\nYour Meal:");

        meal.showItems();

        System.out.println("Total Cost: " + meal.getCost());

    }}
```

STEP NO8: The output.

```
1
Choose a drink: 1. Orange Juice 2. Water
1

Your Meal:
Item: Veggie Sandwich
Packaging: Paper Wrap
Price: 20.0
Item: Orange Juice
Packaging: Plastic Bottle
Price: 25.0
Total Cost: 45.0
```





Algorithm Selected: A*

PREVIOUS PATH2:
a --> c --> e (Goal)
Path to last Goal Node: a --> c --> e (Goal) Cost: 3.0
Nodes expanded: 4

NEW FRONTIER:

Node: c	Path Cost: 2.0	Path: a --> c --> c --> c
Node: e	Path Cost: 3.0	Path: a --> c --> c --> e
Node: b	Path Cost: 1.0	Path: a --> b
Node: c	Path Cost: 4.0	Path: a --> c --> e --> c
Node: a	Path Cost: 4.0	Path: a --> c --> a
Node: a	Path Cost: 4.0	Path: a --> c --> c --> a