**Template Pattern**

**Example 1**

**TemplatePatternDemo.java**

package pack1;

//In Template pattern, an abstract class exposes defined way(s)/template(s)

//to execute its methods. Its subclasses can override the method

//implementation as per need but the invocation is to be in the same way

//as defined by an abstract class.

//This pattern comes under behavior pattern category.

//Advantage

//No code duplication, Reusable code, Easy to implement and readable,

//Flexible, Clearly defined design (avoids ambiguity), Clean architecture,

//Less impact of changes

//

//Disadvantages

//Might Violate SOLID Liskov Substitution Principle, Enforce a

//particular design, Maintenance issue

abstract class ComputerBuilder

{

public final void buildComputer()

{

addMotherboard();

setupMotherboard();

addProcessor();

}

public abstract void addMotherboard();

public abstract void setupMotherboard();

public abstract void addProcessor();

}

class StandardComputer extends ComputerBuilder

{

public void addMotherboard( )

{

System.out.println("Motherboard - Standard Motherboard");

}

public void setupMotherboard()

{

System.out.println("Screwing the motherboard to the case");

System.out.println("Pluging the power supply");

}

public void addProcessor()

{

System.out.println("Processor - Standard Processor");

}

}

class HighEndComputer extends ComputerBuilder

{

public void addMotherboard( )

{

System.out.println("Motherboard - HighEnd Motherboard");

}

public void setupMotherboard()

{

System.out.println("Screwing the motherboard to the case");

System.out.println("Pluging the power supply");

}

public void addProcessor()

{

System.out.println("Processor - HighEnd Processor");

}

}

public class TemplatePatternDemo {

public static void main(String[] args) {

System.out.println("Standard Computer");

ComputerBuilder comType = new StandardComputer();

comType.buildComputer();

System.out.println("\nHigh End Computer");

comType = new HighEndComputer();

comType.buildComputer();

}

}

**Example 2**

**CookingDemo.java**

package pack1;

abstract class CookRecipe

{

public void cookFood()

{

putStoveOn();

cutSpecificVegetable();

cookSpecificVegetable();

putStoveOff();

}

public void putStoveOn( )

{

System.out.println("Turn on the stove");

}

public void putStoveOff(){

System.out.println("Turn off the stove");

}

public abstract void cutSpecificVegetable();

public abstract void cookSpecificVegetable();

}

class CookCarrotRice extends CookRecipe{

public void cutSpecificVegetable( )

{

System.out.println("Cut Carrot");

}

public void cookSpecificVegetable( )

{

System.out.println("Boil and cook carrot rice");

}

}

class CookTomateoRice extends CookRecipe

{

public void cutSpecificVegetable( )

{

System.out.println("Cut Tomateo");

}

public void cookSpecificVegetable( )

{

System.out.println("Boil and cook tomateo rice");

}

}

public class CookingDemo

{

public static void main(String[] args)

{

System.out.println("Cook Carrot Rice");

CookRecipe crobj = new CookCarrotRice( );

crobj.cookFood( );

System.out.println("\nCook Tomateo Rice");

crobj = new CookTomateoRice( );

crobj.cookFood( );

}

}