**1. Cách thực hiện Singleton pattent**

Chương trình Xây dựng Singleton

public class MySingleton {  
 *//1. Khai báo biến sẽ là 1 Singleton* private static MySingleton *instance*;  
 int count = 0;  
 *//2. phương thức khởi tạo phải là private hoặc protected* private MySingleton(){  
  
 }  
 public static MySingleton getInstance(){  
 if (*instance* == null)  
 {  
 *instance* = new MySingleton();  
 }  
 return *instance*;  
 }  
 public void InCount(){  
 System.*out*.println(++count);  
 }  
}

Hàm main

public static void main(String[] args) {  
 MySingleton s1 = MySingleton.*getInstance*();  
 MySingleton s2 = MySingleton.*getInstance*();  
 s1.InCount();*//1* s1.InCount();*//2* s2.InCount();*//3* s1.InCount();*//4*}

**2. Cách thực hiện builder pattent**

Chương trình Xây dựng builder

*// 1. Lớp, khi tạo đối tượng phải khởi tạo nhiều thành phần*public class Computer {  
 *// 2. Khai báo các thành phần cần hỗ trợ việc tạo đối tượng* private String CPU, RAM, storage, screen;  
 @Override  
 public String toString() {  
 return "Computer{" +  
 "CPU='" + CPU + '\'' +  
 ", RAM='" + RAM + '\'' +  
 ", storage='" + storage + '\'' +  
 ", screen='" + screen + '\'' +  
 '}';  
 }  
 *//4. Hàm khởi tạo* protected Computer(Builder b) {  
 this.CPU = b.CPU;  
 this.RAM = b.RAM;  
 this.storage = b.storage;  
 this.screen = b.screen;  
 }  
 *//3.thành phần builder* public static class Builder {  
 String CPU, RAM, storage, screen;  
 public Builder addCPU(String CPU)  
 {  
 this.CPU = CPU;  
 return this;  
 }  
 public Builder addRam(String RAM)  
 {  
 this.RAM = RAM;  
 return this;  
 }  
 public Builder addstorage(String storage)  
 {  
 this.storage = storage;  
 return this;  
 }  
 public Builder addScreen(String screen)  
 {  
 this.screen = screen;  
 return this;  
 }  
 *//5. phương thức tạo ra đối tượng computer* public Computer build() {  
 return new Computer(this);  
 }  
 }  
}

Hàm Main:  
public static void main(String[] args) {  
 Computer computer = new Computer.Builder()  
 .addCPU("Intel")  
 .addRam("5GB")  
 .addstorage("1TB")  
 .addScreen("16K HDR LCD").build();  
 System.*out*.println(computer);  
} **3. Các bước thực hiện Factory Methord**B1: Xây dựng Lớp Trừu tượng bao gồm các phương thức và để cho lớp con cài đặt

*//class khởi tạo phải là abstract*

public abstract class Pizza {  
 StringBuilder builder = new StringBuilder();  
 abstract void prepare();  
 abstract void bake();  
 abstract void cut();  
 abstract void box();

@Override  
 public String toString() {  
 return builder.toString();  
 }  
}

B2: Xây dựng các lớp con tương ứng để cài đặt

public class VN\_HaiSanPizza extends Pizza {  
 @Override  
 void prepare() {  
 builder.append("Bột gạo và 4 con cá cơm");  
 }  
 @Override  
 void bake() {  
 builder.append("\n")  
 .append("Nướng nướng trên lò than");  
 }

@Override  
 void cut() {  
 builder.append("\n").append("Cắt bằng Dao");  
 }  
 @Override  
 void box() {  
 builder.append("Đóng hộp ở Uraina");  
 }  
 @Override  
 public String toString() {  
 return super.toString();  
 }  
}

B3: Xây dựng Factory (Abstract Class):

public abstract class PizzaStore {  
 abstract Pizza createPizza(PizzaType p);  
 public Pizza orderPizza(PizzaType p)  
 {  
 Pizza pizza = createPizza(p);  
 pizza.prepare();  
 pizza.bake();  
 pizza.cut();  
 pizza.box();  
 return pizza;  
 }  
}

Bonus: Có thể xây dựng lớp Enum để phù hợp với yêu cầu đề

public enum PizzaType {  
 *HAISAN*, *NAM*, *BO*, *CHUOT\_DONG*}

Hàm Main

public static void main(String[] args) {  
 PizzaStore ps = new PizzaStore() {  
 @Override  
 Pizza createPizza(PizzaType p) {  
 return new VN\_HaiSanPizza();  
 }  
 };  
 Pizza pizza = ps.orderPizza(PizzaType.*HAISAN*);  
 System.*out*.println(pizza.toString());

}

**4. Các bước thực hiện Composite**

B0: Xác định thành phần Composite, Component

**Composite** chứa tập hợp các Component(Leaf) và cài đặt các phương thức có trong base component

**Component :  
+** base Component là 1 interface or abstract class quy định các phương thức chung cho các thành phần khi tham gia vào mẫu

+ Component (Leaf): là lớp cài đặt các phương thức của base Component (implements) và hiện thực các phương thức của Component.

B1:Tạo một base Component:   
public abstract class Abstractfile {  
 *//Các thuộc tín chung cho tất cả thành phần* String ten, ngaytao, duongdan;  
 String p="";  
 public Abstractfile(String ten, String ngaytao) {  
 this.ten = ten;  
 this.ngaytao = ngaytao;  
 this.duongdan = ten;  
 }  
 public abstract void add(Abstractfile f);  
 public abstract void remove(Abstractfile f);  
 public abstract String getStringTreeFolder();  
 public String getPath(){  
 return duongdan;  
 }  
}

B2: Tiến hành cài đặt Composite và Các Component

+ Component  
public class File extends Abstractfile{  
 public File(String ten, String ngaytao) {  
 super(ten, ngaytao);  
 }  
 @Override  
 public void add(Abstractfile f) {  
  
 }  
 @Override  
 public void remove(Abstractfile f) {  
  
 }  
 @Override  
 public String getStringTreeFolder() {  
 return this.ten;  
 }  
 @Override  
 public String getPath() {  
 return super.getPath();  
 }  
}

+ Composite  
public class Folder extends Abstractfile {  
 List<Abstractfile> children = new ArrayList<>();  
  
 public Folder(String ten, String ngaytao) {  
 super(ten, ngaytao);  
 }  
 @Override  
 public void add(Abstractfile f) {  
 children.add(f);  
 f.duongdan = this.duongdan+ "\\" + f.ten;  
 }  
 @Override  
 public void remove(Abstractfile f) {  
 children.remove(f);  
 }  
 @Override  
 public String getStringTreeFolder() {  
 StringBuilder builder = new StringBuilder();  
 builder.append(this.ten);  
 for (var f: children)  
 {  
 f.p = this.p +"\t"; builder.append("\n").append(f.p).append(f.getStringTreeFolder());  
 f.p ="";  
 }  
 return builder.toString();  
 }  
}

Hàm Main  
public static void main(String[] args) {  
 Folder OCung = new Folder("D:","20/1");  
 Folder folder\_HocTap = new Folder("HocThatVui", "20/02");  
 Folder folder\_XemPhim = new Folder("XemPhim", "10/05");  
 OCung.add(folder\_HocTap);  
 OCung.add(folder\_XemPhim);  
 folder\_HocTap.add(new File("Nhung","12/01"));  
 folder\_HocTap.add(new File("C++","10/01"));  
 folder\_HocTap.add(new File("JaValorant","01/01"));  
 folder\_HocTap.add(folder\_XemPhim);  
  
 folder\_XemPhim.add(new File("Nhung","12/01"));  
 folder\_XemPhim.add(new File("C++","10/01"));  
 folder\_XemPhim.add(new File("JaValorant","01/01"));  
 System.*out*.println(OCung.getStringTreeFolder());  
}

Hàm StringBuilder Trường hợp khi không có danh sách

StringBuilder builder = new StringBuilder();  
builder.append(value);  
return builder.toString();