Task 01:

Apple.java

package Demo\_Codes.Module\_03\_OOAD.AbstractFactoryDP.AbstractFactoryDpPack;

// create samsung and oneplus classes..

public class Apple {

    private Apple() {

        public static Mobile getMobile(String model) {

            if(Object.equals(model, "iphone16")) {

                retrun new Mobile("Here is your iphone 16");

            }

            else if (Object.equals(model, "iphone16MaxPro")) {

                retrun new Mobile("Here is your iphone 16 Max Pro");

            }

            return new NoMobile();

        }

    }

}

Mobile.java

package Demo\_Codes.Module\_03\_OOAD.AbstractFactoryDP.AbstractFactoryDpPack;

public class Mobile {

    String desc;

    public Mobile(String model){

        this.desc = model;

    }

    public void getDesc() {

        System.out.println("this.desc");

    }

}

MobileStore.java

package Demo\_Codes.Module\_03\_OOAD.AbstractFactoryDP.AbstractFactoryDpPack;

// super factory

public class MobileStore {

    private MobileStore() {

        System.out.println("hello welcome to the world of Mobile");

    }

    public static Mobile getMobile(String model) {

            if(Object.equals(brand, "Apple")) {

                System.out.println("Here are your Apple Models");

                retrun Apple.getMobile(model);

            } // else if (Object.equals(brand, "Samsung")) {

            //     System.out.println("Here are your Samsung Models");

                // retrun Samsung.getMobile(model); }

            return new NoMobile();

        }

}

Correct code:

package Demo\_Codes.Module\_03\_OOAD.AbstractFactoryDP.AbstractFactoryDpPack;

public class Mobile {

String desc;

public Mobile(String model) {

this.desc = model;

}

public void getDesc() {

System.out.println(this.desc);

}

}

package Demo\_Codes.Module\_03\_OOAD.AbstractFactoryDP.AbstractFactoryDpPack;

public class NoMobile extends Mobile {

public NoMobile() {

super("Sorry, invalid model.");

}

}

package Demo\_Codes.Module\_03\_OOAD.AbstractFactoryDP.AbstractFactoryDpPack;

public class Apple {

public static Mobile getMobile(String model) {

if ("iphone16".equalsIgnoreCase(model)) {

return new Mobile("Here is your iPhone 16");

} else if ("iphone16MaxPro".equalsIgnoreCase(model)) {

return new Mobile("Here is your iPhone 16 Max Pro");

}

return new NoMobile();

}

}

package Demo\_Codes.Module\_03\_OOAD.AbstractFactoryDP.AbstractFactoryDpPack;

public class MobileStore {

public static Mobile getMobile(String brand, String model) {

if ("Apple".equalsIgnoreCase(brand)) {

System.out.println("Here are your Apple models:");

return Apple.getMobile(model);

}

// You can add Samsung, OnePlus logic later

return new NoMobile();

}

}

package Demo\_Codes.Module\_03\_OOAD.AbstractFactoryDP.AbstractFactoryDpPack;

public class ClientAbstractFactory {

public static void main(String[] args) {

Mobile mObj = MobileStore.getMobile("Apple", "iphone16");

mObj.getDesc();

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

}

}

Output:

Picked up JAVA\_TOOL\_OPTIONS: -Dlog4j2.formatMsgNoLookups=true

Here are your Apple models:

Here is your iPhone 16

...

Process finished with exit code 0

Builder method Design pattern

Task 02:

// product Component

class Laptop {

    int memory;

    int storage;

    // graphic card,processor...

    Laptop() {....}

    getters()..

}

public interface LaptopBuilder {

    LaptopBuilder buildMemory(int memory);

    LaptopBuilder buildStorage(int storage);

    Laptop build();

}

public class LaptopConcreteBuilder {

    private Laptop laptop;

    public LaptopConcreteBuilder() {

        this.laptop = new Laptop();

    }

    @Override

    public LaptopBuilder buildMemory(int memory) {

        laptop.setMemory(memory);

        return this;

    }

    @Override

    public LaptopBuilder buildStorage(int storage) {

        laptop.setStorage(storage);

        return this;

    }

    @Override

     public LaptopBuilder build() {

        return laptop;

    }

}

//Director (optional)

public class LaptopDirector {

    private LaptopBuilder laptopBuilder;

    LaptopDirector(LaptopBuilder laptopBuilder) {

        this.laptopBuilder = laptopBuilder;

    }

    public Laptop constructLaptop() {

        return laptopBuilder

                    .builderMemory

                    .buildStorage

                    .build();

    }

}

public class ClientBuildMethodDP {

    psvm (String....) {

        LaptopBuilder lbobj = new laptopConcreteBuilder();

        LaptopDirector dir = new LaptopDirector(lbobj);

        Laptop lobj = dir.constructLaptop();

        sout(lobj);

    }

}

Correct code:

package Aug8.BuilderesignPattern;  
  
// Product component  
class Laptop {  
 private int memory;  
 private int storage;  
  
 public void setMemory(int memory) {  
 this.memory = memory;  
 }  
  
 public void setStorage(int storage) {  
 this.storage = storage;  
 }  
  
 public int getMemory() {  
 return memory;  
 }  
  
 public int getStorage() {  
 return storage;  
 }  
  
 @Override  
 public String toString() {  
 return "Laptop [Memory = " + memory + " GB, Storage = " + storage + " GB]";  
 }  
}  
  
// Builder interface  
interface LaptopBuilder {  
 LaptopBuilder buildMemory(int memory);  
 LaptopBuilder buildStorage(int storage);  
 Laptop build();  
}  
  
// Concrete Builder  
class LaptopConcreteBuilder implements LaptopBuilder {  
 private final Laptop laptop;  
  
 public LaptopConcreteBuilder() {  
 this.laptop = new Laptop();  
 }  
  
 @Override  
 public LaptopBuilder buildMemory(int memory) {  
 laptop.setMemory(memory);  
 return this;  
 }  
  
 @Override  
 public LaptopBuilder buildStorage(int storage) {  
 laptop.setStorage(storage);  
 return this;  
 }  
  
 @Override  
 public Laptop build() {  
 return laptop;  
 }  
}  
  
// Director (optional)  
class LaptopDirector {  
 private final LaptopBuilder laptopBuilder;  
  
 public LaptopDirector(LaptopBuilder laptopBuilder) {  
 this.laptopBuilder = laptopBuilder;  
 }  
  
 public Laptop constructLaptop() {  
 return laptopBuilder  
 .buildMemory(16)  
 .buildStorage(512)  
 .build();  
 }  
}  
  
// Client  
public class ClientBuildMethodDP {  
 public static void main(String[] args) {  
 LaptopBuilder builder = new LaptopConcreteBuilder();  
 LaptopDirector director = new LaptopDirector(builder);  
 Laptop laptop = director.constructLaptop();  
  
 System.*out*.println(laptop);  
 }  
}

output:

Picked up JAVA\_TOOL\_OPTIONS: -Dlog4j2.formatMsgNoLookups=true

Laptop [Memory = 16 GB, Storage = 512 GB]

Process finished with exit code 0

Task 03:

package com.example;

//Prototype interface

public interface Colors {

    Colors clone();

    String getName();

    void setName(String name);

}

package com.example;

public class BlackConcreteprototype implements Colors{

    private String name;

    public BlackConcreteprototype() {

        System.out.println(" BlackConcreteprototype constructor is called");

    }

    public BlackConcreteprototype(String name) {

        this.name = name;

    }

    public Colors clone() {

        return new BlackConcreteprototype(this.name);

    }

    public String getName() {

        return name;

    }

    public void setName(String name){

        this.name = name;

    }

}

package com.example;

public class WhiteConcretePrototype implements Colors{

    private String name;

    public WhiteConcretePrototype() {

        System.out.println(" WhiteConcretePrototype constructor is called");

    }

    public WhiteConcretePrototype(String name) {

        this.name = name;

    }

    public Colors clone() {

        return new WhiteConcretePrototype(this.name);

    }

    public String getName() {

        return name;

    }

    public void setName(String name){

        this.name = name;

    }

}

package com.example;

// client code of Prototype method Design pattern

public class Main {

    public static void main(String[] args) {

        Colors BlackPrototypeObj = new BlackConcreteprototype("Black Color");

        Colors WhitePrototypeObj = new WhiteConcretePrototype("white color");

        Colors clonedBlackObj = BlackPrototypeObj.clone();

        Colors clonedWhiteObj = WhitePrototypeObj.clone();

        clonedBlackObj.setName("dark color");

        clonedWhiteObj.setName("light color");

        System.out.println("black color is " + clonedBlackObj.getName());

        System.out.println("while color is "+ clonedWhiteObj.getName());

    }

}

Prototype Method Design pattern

    prototype interface -- Colors

    Concrete prototype --- class

        WhiteConcretePrototype

            constructor()

            constructor(String...)

            clone()

        BlackConcreteprototype

    client code

correct code:

package com.example;

public interface Colors {

Colors clone();

String getName();

void setName(String name);

}

package com.example;

public class BlackConcretePrototype implements Colors {

private String name;

public BlackConcretePrototype() {

System.out.println("BlackConcretePrototype constructor is called");

}

public BlackConcretePrototype(String name) {

this.name = name;

}

@Override

public Colors clone() {

return new BlackConcretePrototype(this.name);

}

@Override

public String getName() {

return name;

}

@Override

public void setName(String name) {

this.name = name;

}

}

package com.example;

public class WhiteConcretePrototype implements Colors {

private String name;

public WhiteConcretePrototype() {

System.out.println("WhiteConcretePrototype constructor is called");

}

public WhiteConcretePrototype(String name) {

this.name = name;

}

@Override

public Colors clone() {

return new WhiteConcretePrototype(this.name);

}

@Override

public String getName() {

return name;

}

@Override

public void setName(String name) {

this.name = name;

}

}

package com.example;

public class Main {

public static void main(String[] args) {

Colors blackPrototype = new BlackConcretePrototype("Black Color");

Colors whitePrototype = new WhiteConcretePrototype("White Color");

Colors clonedBlack = blackPrototype.clone();

Colors clonedWhite = whitePrototype.clone();

clonedBlack.setName("Dark Color");

clonedWhite.setName("Light Color");

System.out.println("Cloned black color is: " + clonedBlack.getName());

System.out.println("Cloned white color is: " + clonedWhite.getName());

}

}

Output:

BlackConcretePrototype constructor is called

WhiteConcretePrototype constructor is called

Cloned black color is: Dark Color

Cloned white color is: Light Color