**1. Abstract Factory Pattern.**

Nếu bạn đã quen với  [Factory Pattern](https://stackjava.com/design-pattern/factory-pattern-code-vi-du-factory-pattern-bang-java.html) thì bạn sẽ thấy nó chỉ có 1 Factory class, factory class này sẽ trả về các class con (sub-class) dựa trên dầu vào (factory class sử dụng if-else hoặc switch để xác định class con đầu ra).

Trong Abstract Factory pattern, chúng ta không cần if-else hoặc switch. Mỗi sub-class sẽ có một factory class. Abstract Factory class sẽ trả về sub-class dựa trên đầu vào là factory class.

**Một số ví dụ về Abstract Factory trong JDK:**

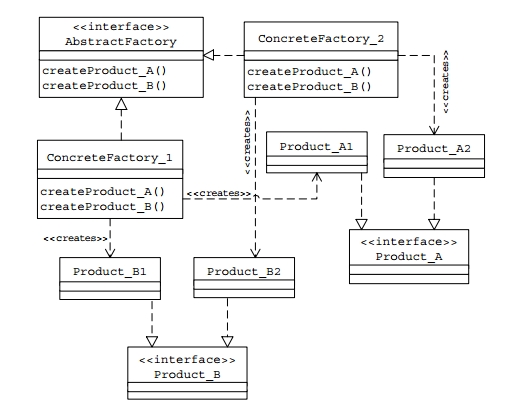
* javax.xml.parsers.DocumentBuilderFactory#newInstance()
* javax.xml.transform.TransformerFactory#newInstance()
* javax.xml.xpath.XPathFactory#newInstance()

**Lợi ích của Abstract Factory Pattern.**

* Abstract Factory pattern cung cấp hưỡng tiếp cận code bằng interface thay vì các cài đặt (Giống với Factory Pattern)
* Abstract Factory pattern là “factory của các factory” và có thể dễ dạng mở rộng để chứa thêm các factory và các sub-class
* Abstract Factory giúp tránh được việc sử dụng điều kiện logic như bên trong Factory Pattern.

**2. Abstract Factory Pattern UML Diagram**

Cũng giống như   [Factory Pattern](https://stackjava.com/design-pattern/factory-pattern-code-vi-du-factory-pattern-bang-java.html), chúng ta sẽ sử dụng các super class, sub-class

****

Khác với Factory Pattern, trong Abstract Factory có nhiều Factory Class (ConcreateFactory\_1, ConcreteFactory\_2) cùng implement AbstractFactory, mỗi Factory Class con này sẽ thực hiện trả về các object khác nhau.

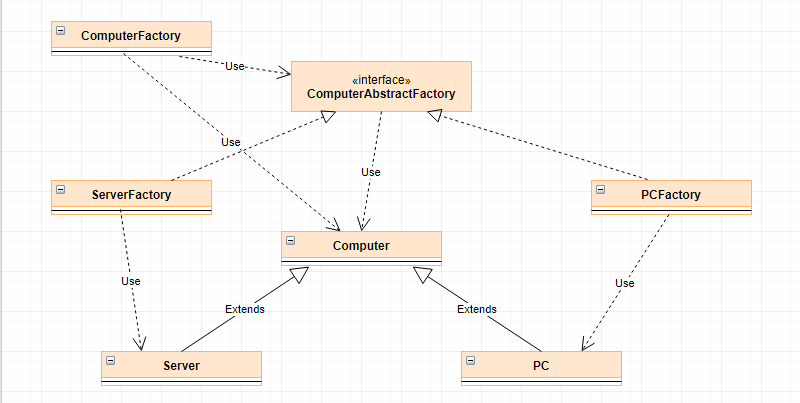
**3.Ví dụ**

Ta có class cha là computer và 2 class con Server và PC.

ComputerAbstractFactory là một interface cung cấp phương thức trả về đối tượng Computer.

ServerFactory và PCFactory sẽ cài đặt các phương thức của ComputerAbstractFactory.

ComputerFactory sẽ sử dụng ComputerAbstractFactory để tạo đối tượng Computer. (Trường hợp ComputerAbstractFactory là ServerFactory thì sẽ trả về đối tượng Server, trường hợp ComputerAbstractFactory là PCFactory thì sẽ trả về đối tượng PC)



Computer.java

* public abstract class Computer {
* public abstract String getRAM();
* public abstract String getHDD();
* public abstract String getCPU();
* @Override
* public String toString(){
* return "RAM= "+this.getRAM()+", HDD="+this.getHDD()+", CPU="+this.getCPU();
* }
* }

Factory Design Pattern Sub Classes

PC.java

* public class PC extends Computer {
* private String ram;
* private String hdd;
* private String cpu;
* public PC(String ram, String hdd, String cpu){
* this.ram=ram;
* this.hdd=hdd;
* this.cpu=cpu;
* }
* @Override
* public String getRAM() {
* return this.ram;
* }
* @Override
* public String getHDD() {
* return this.hdd;
* }
* @Override
* public String getCPU() {
* return this.cpu;
* }
* }

Server.java

* public class Server extends Computer {
* private String ram;
* private String hdd;
* private String cpu;
* public Server(String ram, String hdd, String cpu){
* this.ram=ram;
* this.hdd=hdd;
* this.cpu=cpu;
* }
* @Override
* public String getRAM() {
* return this.ram;
* }
* @Override
* public String getHDD() {
* return this.hdd;
* }
* @Override
* public String getCPU() {
* return this.cpu;
* }
* }

ComputerAbstractFactory.java

* import com.journaldev.design.model.Computer;
* public interface ComputerAbstractFactory {
* public Computer createComputer();
* }

PCFactory.java

* public class PCFactory implements ComputerAbstractFactory {
* private String ram;
* private String hdd;
* private String cpu;
* public PCFactory(String ram, String hdd, String cpu){
* this.ram=ram;
* this.hdd=hdd;
* this.cpu=cpu;
* }
* @Override
* public Computer createComputer() {
* return new PC(ram,hdd,cpu);
* }
* }

ServerFactory.java

* public class ServerFactory implements ComputerAbstractFactory {
* private String ram;
* private String hdd;
* private String cpu;
* public ServerFactory(String ram, String hdd, String cpu){
* this.ram=ram;
* this.hdd=hdd;
* this.cpu=cpu;
* }
* @Override
* public Computer createComputer() {
* return new Server(ram,hdd,cpu);
* }
* }

ComputerFactory.java

* public class ComputerFactory {
* public static Computer getComputer(ComputerAbstractFactory factory){
* return factory.createComputer();
* }
* }

Demo:

* public class TestDesignPatterns {
* public static void main(String[] args) {
* testAbstractFactory();
* }
* private static void testAbstractFactory() {
* Computer pc = ComputerFactory.getComputer(new PCFactory("2 GB","500 GB","2.4 GHz"));
* Computer server = ComputerFactory.getComputer(new ServerFactory("16 GB","1 TB","2.9 GHz"));
* System.out.println("AbstractFactory PC Config::"+pc);
* System.out.println("AbstractFactory Server Config::"+server);
* }
* }