**1. Factory Pattern là gì?**

Factory Pattern là một mẫu thiết kế tạo dựng ([Creation Pattern](https://stackjava.com/design-pattern/design-pattern-la-gi-design-pattern-trong-java.html)), nó được sử dụng rộng rãi trong JDK và các Framework như Spring, Struts.

Factory Pattern được sử dụng khi có một class cha (super class) với nhiều class con (sub-class), dựa trên đầu vào và phải trả về 1 trong những class con đó.

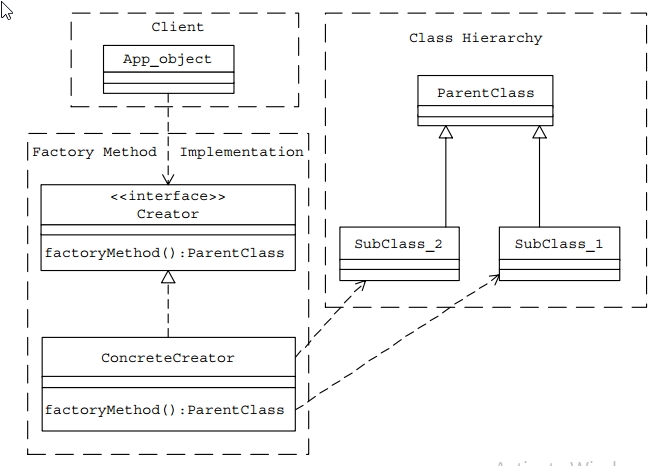
**Lợi ích của Factory Pattern**

* Cung cấp 1 hướng tiếp cận với Interface thay thì các cài đặt.
* Mở rộng code dễ dàng hơn (Khi muốn mở rộng thêm 1 loại class con khác, ta chỉ việc thêm mới nó và sửa đầu vào trong Factory Method là được).

**Ví dụ: Một số class, method áp dụng Factory Pattern trong JDK**

* java.util.Calendar, ResourceBundle and NumberFormat là các class sử dụng Factory Pattern.
* Method valueOf() trong một số wrapper class như Boolean, Integer, Double…

**2. Factory Pattern UML Diagram**

****

**ParrentClass**: super class

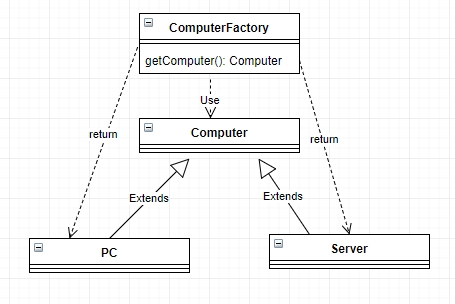
**SubClass\_1, SubClass\_2:** Các class cần trả về

**Creator**: Interface khai báo factoryMethod trả về ParrentClass

**ConcreteCreator**: Thực hiện cài đặt Creator.

(Thông thường vì chỉ có 1 class **ConcreteCreator** implement Creator thì **Creator** sẽ được bỏ qua)

**3. Ví dụ**



Ta có class cha là *Computer* và 2 class con *Server* và *PC.*

*ComputerFactory* thực hiện trả về đối tượng *Server* hoặc *PC* tùy theo dữ liệu đầu vào.

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;
* }
* }

Factory Class

ComputerFactory.java

* public class ComputerFactory {
* public static Computer getComputer(String type, String ram, String hdd, String cpu){
* if("PC".equalsIgnoreCase(type)) return new PC(ram, hdd, cpu);
* else if("Server".equalsIgnoreCase(type)) return new Server(ram, hdd, cpu);
* return null;
* }
* }

Một số điểm quan trọng về Factory Pattern là:

* Chúng ta cần giữ Factory class là Singleton hoặc giữ method thực hiện trả về subclass là static
* Dựa trên tham số đầu vào mà ta tạo và trả về các subclass khác nhau (ví dụ ở trên thì getComputer là factory method)

**Demo**

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