

# Java Factory Design Pattern - Mẫu thiết kế Factory Design trong Java

Factory Design Pattern được sử dụng khi chúng ta có một super-class với nhiều sub-class và dựa trên đầu vào, chúng ta cần trả về một trong các sub-class. Design Pattern này nhận trách nhiệm khởi tạo một lớp từ chương trình client đến lớp factory.

## 1. Tại sao cần Factory Design Pattern

### - Factory Pattern được sử dụng khi:

- Chúng ta có một super class với nhiều sub-class và dựa trên đầu vào, chúng ta cần trả về một sub-class. Mô hình này giúp chúng ta đưa trách nhiệm của việc khởi tạo một lớp từ phía người dùng (client) sang lớp Factory.
- Chúng ta không biết sau này sẽ cần đến những sub-class nào nữa. Khi cần mở rộng, hãy tạo ra sub class và implement thêm vào factory method cho việc khởi tạo sub class này.

### - Lợi ích của Factory Pattern:

- Factory Pattern giúp giảm sự phụ thuộc giữa các module (loose coupling): cung cấp 1 hướng tiếp cận với Interface thay thì các implement. Giúp chương trình độc lập với những lớp cụ thể mà chúng ta cần tạo 1 đối tượng, code ở phía client không bị ảnh hưởng khi thay đổi logic ở factory hay sub class.
- Mở rộng code dễ dàng hơn: khi cần mở rộng, chỉ việc tạo ra sub class và implement thêm vào factory method.
- Khởi tạo các Objects mà che giấu đi xử lý logic của việc khởi tạo đấy. Người dùng không biết logic thực sự được khởi tạo bên dưới phương thức factory.
- Dễ dàng quản lý life cycle của các Object được tạo bởi Factory Pattern.
- Thống nhất về naming convention: giúp cho các developer có thể hiểu về cấu trúc source code.

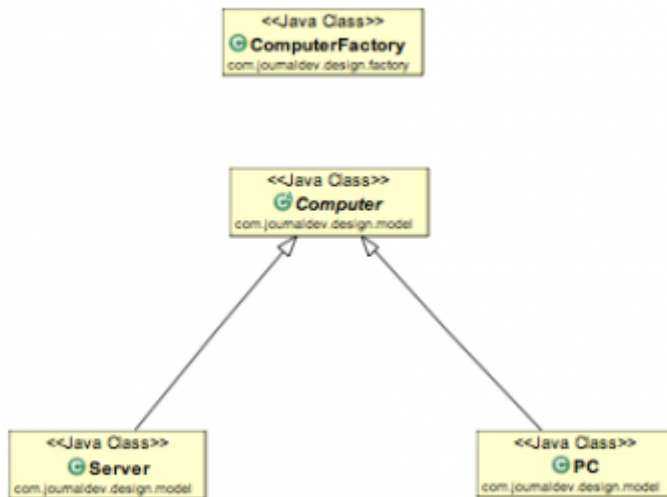
## 2. Thực thi Factory Design Pattern

Một Factory Pattern bao gồm các thành phần cơ bản sau:

- **Super Class:** một supper class trong Factory Pattern có thể là một **interface**, **abstract class** hay một **class** thông thường.

- **Sub Classes:** các sub class sẽ implement các phương thức của **supper class** theo nghiệp vụ riêng của nó.
- **Factory Class:** một class chịu trách nhiệm khởi tạo các đối tượng **sub class** dựa theo tham số đầu vào. Lưu ý: lớp này là **Singleton** hoặc cung cấp một **public static method** cho việc truy xuất và khởi tạo đối tượng. Factory class sử dụng if-else hoặc switch-case để xác định class con đầu ra.

**Ví dụ:** Chương trình sau thể hiện cách chúng ta xây dựng mô hình quan hệ giữa các computer: PC, server



#### - Factory Design Pattern Super Class

```

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

```

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;
}

}

```

```

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

```

import design.model.Computer;
import design.model.PC;
import design.model.Server;

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;
    }
}

```

## - Test Class

```

import design.factory.ComputerFactory;
import design.model.Computer;

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);

}

}
```

### - Output:

```
Factory PC Config: RAM= 2 GB, HDD=500 GB, CPU=2.4 GHz
Factory Server Config: RAM= 16 GB, HDD=1 TB, CPU=2.9 GHz
```

---

Revision #4

Created 2 November 2019 15:08:28 by Laptrinh.vn

Updated 12 April 2020 14:54:29 by Laptrinh.vn