Factory Method

A variation on a theme
Recap - The problem with simple factory

Simple Factory used only one factory class for creating any kind of character (and likewise with the weapon factory).

We had to add cases to the switch-case of the createCharacter method whenever a new concrete character was introduced. That’s a violation of “open-closed”.

We should be open for extension (adding more characters) but closed for modification (we shouldn’t have to modify the factory each time).
The factory method pattern (as we interpret it) is an improvement over simple factory, because we add new factories rather than maintaining one single factory.

Client code now chooses the correct factory instead. We can keep adding character classes and corresponding factories.
The CharacterFactory interface becomes simpler

```java
public interface CharacterFactory{
    public Character createCharacter(WeaponType weapon,
                                    String name);
    public Character createUnarmedCharacter(String name);
}
// No need to specify character type!
```
The client code becomes simpler

CharacterFactory cFactory = new TrollFactory();
Character troll = cFactory.createCharacter(WeaponType.CLUB, "Lillfjant");

cFactory = new KnightFactory();
Character sirJames = cFactory.createCharacter(WeaponType.SWORD, "Sir James");
Character blackKnight = cFactory
    .createUnarmedCharacter("Fistfighting Black Knight");

cFactory = new OrchFactory();
Character evilOrch = cFactory.createCharacter(WeaponType.SHOTGUN,
                                           "Ugly scary Orch");
Character unarmedOrch = cFactory
    .createUnarmedCharacter("Ugly scary unarmed Orch");
Virtual method version

Client code:
CarFactory cf = new VolvoFactory();
System.out.println(cf.createCar());
cf = new BMWFactory();
System.out.println(cf.createCar());
Virtual method version

public abstract class CarFactory{
    // Virtual method - A concrete method which calls an abstract method
    public Car createCar(){
        return makeIt();
    }
    protected abstract Car makeIt(); // Overriding classes decide type
}

class VolvoFactory extends CarFactory{
    @Override
    protected Car makeIt(){
        return new Volvo();
    }
}

class BMWFactory extends CarFactory{
    @Override
    protected Car makeIt(){
        return new BMW();
    }
}
Virtual method version

CarFactory cf = new VolvoFactory();
System.out.println(cf.createCar());
cf = new BMWFactory();
System.out.println(cf.createCar());

// createCar() is inherited “as is”
// it calls the overridden version
// of makeIt()

class VolvoFactory extends CarFactory{
    @Override
    protected Car makeIt(){
        return new Volvo();
    }
}
class BMWFactory extends CarFactory{
    @Override
    protected Car makeIt(){
        return new BMW();
    }
}