Factory Method Design Pattern

* Product (Pizza): Is an interface or an abstract class whose subclasses are instantiated by the factory method.
* ConcreteProduct (CheesePizza, PepperoniPizza, and VeggiePizza): Are the concrete subclasses that implement/extend Product. The factory method instantiates these subclasses.
* Creator (BasePizzaFactory): Is an interface or an abstract class that declares the factory method, which returns an object of type Product.
* ConcreteCreator (PizzaFactory): Is a concrete class that implements the factory method to create and return a ConcreteProduct to Client.
* Client: Asks the Creator for a Product.

A Client that requires a ConcreteProduct does not create any object but instead asks the Creator for it. The ConcreteCreator implements the factory method to create the object transparently from the Client. As a result, the Client is not required to be aware of any ConcreteProduct and how they are created. This approach advocates the Object Oriented Programming principle “*Program to an interface, not an implementation*“, which leads to [polymorphism](https://springframework.guru/polymorphism-java/), a key feature of object-oriented programming. In addition, as object creation is centralized in the ConcreteCreator, any changes made to a Product or any ConcreteProduct does not affect the Client.

Spring uses this technique at the root of its [Dependency Injection (DI) framework](https://www.baeldung.com/spring-dependency-injection).

Fundamentally, **Spring treats** **a bean container as a factory that produces beans.**

**public** **interface** **BeanFactory** {

getBean(Class<T> requiredType);

getBean(Class<T> requiredType, Object... args);

getBean(String name); // ... ]

[java - Implement a simple factory pattern with Spring 3 annotations - Stack Overflow](https://stackoverflow.com/questions/6390810/implement-a-simple-factory-pattern-with-spring-3-annotations)

public interface MyService {

String getType();

void checkStatus();

}

@Component

public class MyServiceOne implements MyService {

@Override

public String getType() {

return "one";

}

@Override

public void checkStatus() {

// Your code

}

}

@Component

public class MyServiceTwo implements MyService {

@Override

public String getType() {

return "two";

}

@Override

public void checkStatus() {

// Your code

}

}

@Component

public class MyServiceThree implements MyService {

@Override

public String getType() {

return "three";

}

@Override

public void checkStatus() {

// Your code

}

}

@Service

public class MyServiceFactory {

@Autowired

private List<MyService> services;

private static final Map<String, MyService> myServiceCache = new HashMap<>();

@PostConstruct

public void initMyServiceCache() {

for(MyService service : services) {

myServiceCache.put(service.getType(), service);

}

}

public static MyService getService(String type) {

MyService service = myServiceCache.get(type);

if(service == null) throw new RuntimeException("Unknown service type: " + type);

return service;

}

}