**Design pattern**

There are 3 categories of design patterns.

1. Creational design pattern.
2. Structural design pattern.
3. Behavioural design pattern.

***Creational design pattern:***There are 5 creational design patterns.

1. Abstract factory.
2. Builder.
3. Factory Method.
4. Prototype.
5. Singleton.
6. **Abstract Factory.**

The Abstract Factory Pattern is a [creational design pattern](https://www.geeksforgeeks.org/creational-design-pattern/)that provides an interface for creating families of related or dependent objects without specifying their concrete classes, in simpler terms the Abstract Factory Pattern is a way of organizing how you create groups of things that are related to each other.

In simple terms it is kind of parent interface reference pointing to child class object that we generally use every day.

Example: List<String> list = new ArrayList<>();

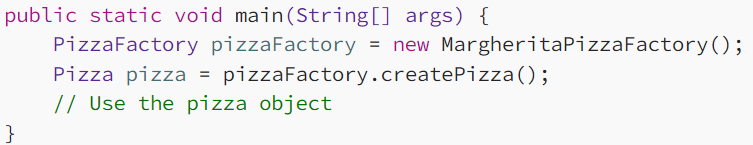
Reference:

<https://refactoring.guru/design-patterns/abstract-factory>

<https://www.geeksforgeeks.org/abstract-factory-pattern/>

1. **Factory Method.**

Factory design pattern provides an interface to create objects using superclass, and allows child classes to alter the type of objects that will be created.



We created a new margarita factory object in common pizza factory interface, and used just common method createPizza() to get an object of margarita pizza.

Reference:

<https://refactoring.guru/design-patterns/factory-method>

<https://medium.com/@eshikashah2001/exploring-the-factory-method-design-pattern-4d270b6ff935>



WTF – both the patterns look similar... Then what’s the difference??

<https://medium.com/@sumit-s/factory-method-pattern-and-abstract-factory-pattern-89fdfb8c364e>

There are many differences in the internet but I’m not convinced… they both are the same.

1. **Builder.**

Builder design pattern is used to construct complex object in a step-by-step manner to reduce complexity in the client-side code.

*Components of the Builder Design Pattern*

1. Product

The Product is the complex object that the Builder pattern is responsible for constructing.

Example car object.

2. Builder

The Builder is an interface or an abstract class that declares the construction steps for building a complex object.

3. Concrete Builder

Concrete Builder classes implement the Builder interface, providing specific implementations for building each part of the product.

The purpose of this class initialises the values to our main object (product).

4. Director

The Director is responsible for managing the construction process of the complex object.

This class drives the code that calls all the required methods and constructors in an order to create our complex object.

5. Client

The Client is the code that initiates the construction of the complex object.

This is the client code that requires our complex object.

References:

<https://www.geeksforgeeks.org/builder-design-pattern/>

<https://refactoring.guru/design-patterns/builder/java/example#example-0>

1. **Prototype.**

The Prototype Design Pattern is a creational pattern that enables the creation of new objects by copying an existing object. Prototype allows us to hide the complexity of making new instances from the client.

In simple terms, when we try to create a clone of an object we create it and assign all its values in our flow of execution, but instead of that, we write that piece of code inside the object class itself as clone () method, so that easy for client side application to get clones.

Main point: if we clone object by traditional way, we can’t copy the values of private variables, this prototype pattern helps us to fix this problem.

References:

<https://refactoring.guru/design-patterns/prototype>

<https://www.geeksforgeeks.org/prototype-design-pattern/>

<https://www.tutorialspoint.com/design_pattern/prototype_pattern.htm>

1. **Singleton.**

**Singleton** is a creational design pattern, which ensures that only one object of its kind exists and provides a single point of access to it for any other code.

Basically, we know Singleton and the only thing we know was singleton.

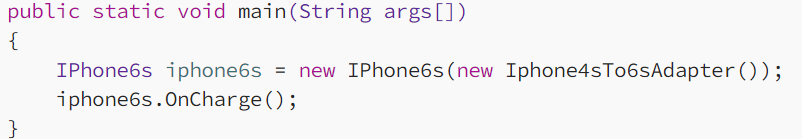
References:  
<https://refactoring.guru/design-patterns/singleton>

<https://www.geeksforgeeks.org/singleton-design-pattern/>

***Structural design pattern:***There are 7 creational design patterns.

1. Adapter.
2. Bridge.
3. **Adapter**

The Adapter Design Pattern is a structural design pattern used to allow two incompatible interfaces or systems to work together. It acts as a bridge between two classes that otherwise couldn’t communicate due to incompatible interfaces.



Consider we have only iphone4s charges, now we want to charge iphone6s in it, then we create an adapter called Iphone4sTo6sAdapter class to charge the mobile. That’s it simple.

References:

Code: <https://medium.com/@akshatsharma0610/adapter-design-pattern-in-java-fa20d6df25b8>

Code: <https://refactoring.guru/design-patterns/adapter/java/example#example-0>

<https://refactoring.guru/design-patterns/adapter/java/example#lang-features>