Guide To Java Employment

**Contents**

[SOLID 2](#_Toc118647243)

[S – Single Responsibility Principle 2](#_Toc118647244)

[O – Open-Closed Principle 2](#_Toc118647245)

[L – Liskov Substitution Principle 2](#_Toc118647246)

[I – Interface Segregation Principle 2](#_Toc118647247)

[D – Dependency Inversion Principle 2](#_Toc118647248)

[Design Patterns 3](#_Toc118647249)

[Creational Patterns 3](#_Toc118647250)

[Factory Pattern 3](#_Toc118647251)

[Abstract Factory Pattern 4](#_Toc118647252)

[Builder Pattern 5](#_Toc118647253)

# SOLID

## S – Single Responsibility Principle

Single Responsibility Principle states that a class should only have one responsibility. It should also only have one reason to change.

## O – Open-Closed Principle

Classes should be open for extension but closed for modification.

*If a class is modified with new logic, it shouldn’t affect other logic. If I add extra logic, it should be one extra line to use said logic.*

## L – Liskov Substitution Principle

If class A is a subtype of class B, class A should be replaceable by class B without disrupting the behavior of the program.

*Don’t inherit/extend a class if you don’t plan on using that logic or if that logic is not appliable to your class, because if you have a behavior that doesn’t work on your class, when you interchange your class with a child of your parent, it will crash.*

## I – Interface Segregation Principle

Classes shouldn’t be forced to depend on interface methods that they do not use.

## D – Dependency Inversion Principle

High-level modules shouldn’t depend on lower-level modules, they should depend on abstractions.

*If you plan to use a function from a class that will be a variable inside your class, then apply an abstraction layer to your class variable, so that if you add extra logic in the future, you both won’t need to modify existing classes and the logic can be added just by constructing the object with a different type of class.*

# Design Patterns

## Creational Patterns

### Factory Pattern



If there are multiple objects that share the same parent/interface, use a factory in order to avoid having many ifs inside your class. The entire if/switch logic will go inside the factory and then everyone instantiates the factory and only applies a single method in order to get the required object.

### Abstract Factory Pattern

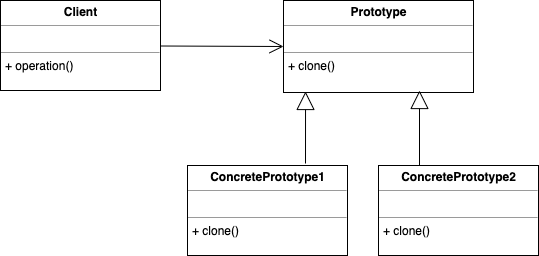


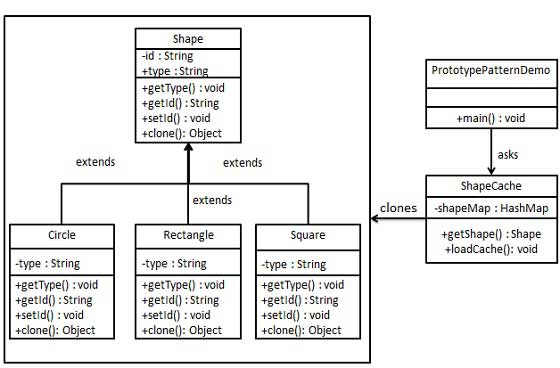
Abstract factory is like a factory, only that it builds other factories. In order to use the abstract factory, first you instantiate the factory depending on parameters, then you create objects with instantiated factory.

### Builder Pattern

Maybe sometimes we don’t want to create a class with all it’s components, or we want to create an object piece by piece. This is where the builder pattern comes in.

### Prototype Pattern





For creating an instance of the class (prototype) and then create new objects by just copying the prototype.

### Singleton

The purpose of a singleton is to ensure that a class has only one instance, while providing a global access point to this instance.