**Design Patterns in Java**

A design pattern is a **well-proved solution** for solving the specific problem/task.

Design patterns are programming language independent strategies for solving the common object-oriented design problems. That means, a design pattern represents an idea, not a particular implementation.

By using the design patterns you can make your code more flexible, reusable and maintainable. It is the most important part because java internally follows design patterns.

#### Understanding the problem without using Design Patterns

Now, a question will be arising in your mind what kind of specific problem? Let me explain by an example:

**Problem Given:**  
Suppose you want to create a class for which only a single instance (or object) should be created and that single object can be used by all other classes.

**Solution:**  
**Singleton design pattern** is the best solution of above specific problem. So, every design pattern has **some specification or set of rules** for solving the problems. What are those specifications, you will see later in the types of design patterns.

### Advantage of design pattern

1. They are reusable in multiple projects.
2. They provide the solutions that help to define the system architecture.
3. They capture the software engineering experiences.
4. They provide transparency to the design of an application.
5. They are well-proved and testified solutions since they have been built upon the knowledge and experience of expert software developers.
6. Design patterns don't guarantee an absolute solution to a problem. They provide clarity to the system architecture and the possibility of building a better system.

### When should we use the design patterns?

We must use the design patterns **during the analysis and requirement phase of SDLC**(Software Development Life Cycle).

Design patterns ease the analysis and requirement phase of SDLC by providing information based on prior hands-on experiences.

#### Types of design patterns

Basically, design patterns are categorized into two parts:

1. Core java (or JavaSE) Design Patterns.
   * Creational Design Pattern
   * Structural Design Pattern
   * Behavioral Design Pattern
2. JavaEE (J2EE) Design Patterns.
   * Presentation Layer Design Pattern
   * Business Layer Design Pattern
   * Data Layer Design Pattern

# Core Java Design Patterns

In core java, there are mainly three types of design patterns, which are further divided into their sub-parts:

## 1. Creational Design Pattern

1. Factory Pattern
2. Abstract Factory Pattern
3. Singleton Pattern
4. Prototype Pattern
5. Builder Pattern.

## 2. Structural Design Pattern

1. Adapter Pattern
2. Bridge Pattern
3. Composite Pattern
4. Decorator Pattern
5. Facade Pattern
6. Flyweight Pattern
7. Proxy Pattern

## 3. Behavioral Design Pattern

1. Chain Of Responsibility Pattern
2. Command Pattern
3. Interpreter Pattern
4. Iterator Pattern
5. Mediator Pattern
6. Memento Pattern
7. Observer Pattern
8. State Pattern
9. Strategy Pattern
10. Template Pattern
11. Visitor Pattern

# Creational design patterns

Creational design patterns are concerned with **the way of creating objects.** These design patterns are used when a decision must be made at the time of instantiation of a class (i.e. creating an object of a class).

But everyone knows an object is created by using new keyword in java. For example:

1. StudentRecord s1=new StudentRecord();

Hard-Coded code is not the good programming approach. Here, we are creating the instance by using the new keyword. Sometimes, the nature of the object must be changed according to the nature of the program. In such cases, we must get the help of creational design patterns to provide more general and flexible approach.

## Types of creational design patterns

There are following 5 types of creational design patterns.

1. Factory Method Pattern
2. Abstract Factory Pattern
3. Singleton Pattern
4. Prototype Pattern
5. Builder Pattern

# Factory Method Pattern

A Factory Pattern or Factory Method Pattern says that just **define an interface or abstract class for creating an object but let the subclasses decide which class to instantiate.** In other words, subclasses are responsible to create the instance of the class.

The Factory Method Pattern is also known as **Virtual Constructor.**

#### Advantage of Factory Design Pattern

#### Factory Method Pattern allows the sub-classes to choose the type of objects to create.

#### It promotes the loose-coupling by eliminating the need to bind application-specific classes into the code. That means the code interacts solely with the resultant interface or abstract class, so that it will work with any classes that implement that interface or that extends that abstract class.

#### Usage of Factory Design Pattern

#### When a class doesn't know what sub-classes will be required to create

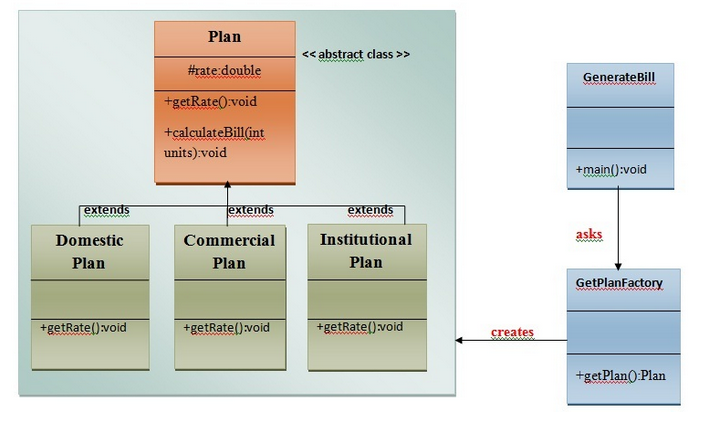
#### When a class wants that its sub-classes specify the objects to be created.

#### When the parent classes choose the creation of objects to its sub-classes.

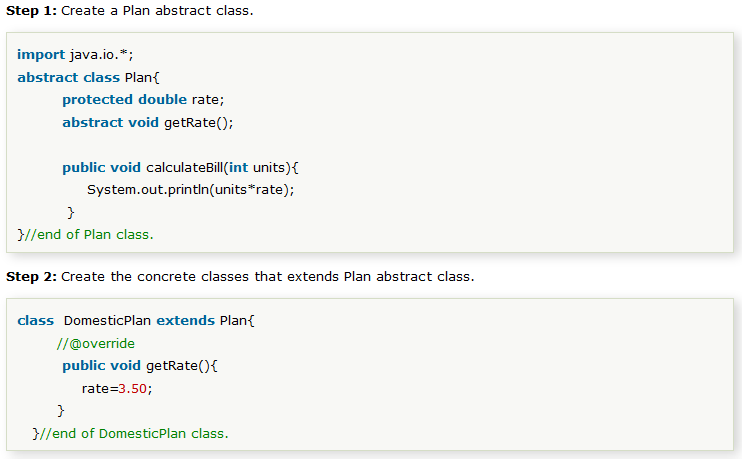
#### UML for Factory Method Pattern

#### We are going to create a Plan abstract class and concrete classes that extends the Plan abstract class. A factory class GetPlanFactory is defined as a next step.

#### GenerateBill class will use GetPlanFactory to get a Plan object. It will pass information (DOMESTICPLAN / COMMERCIALPLAN / INSTITUTIONALPLAN) to GetPalnFactory to get the type of object it needs.

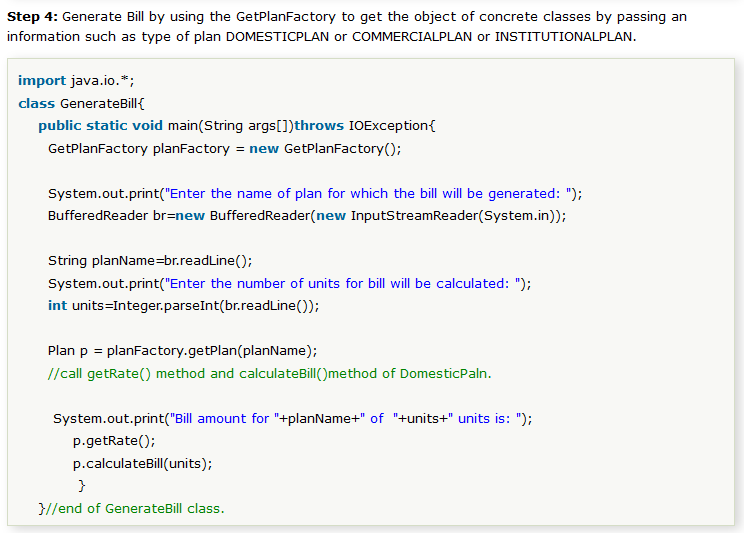


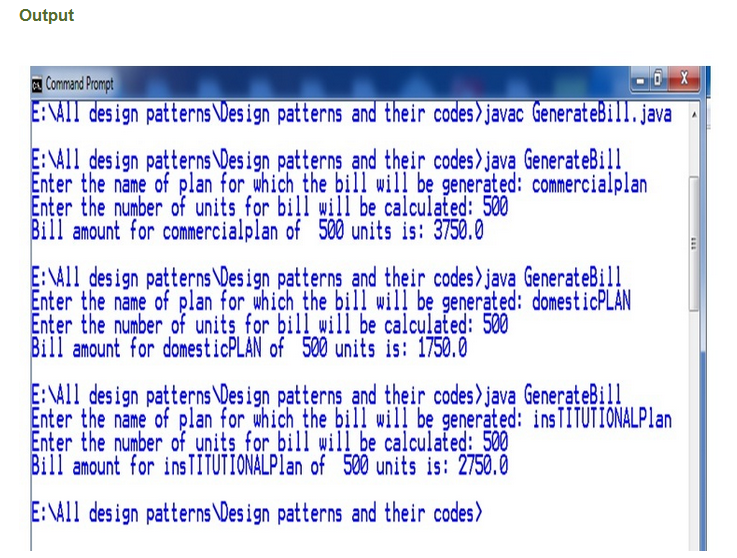
### Calculate Electricity Bill : A Real World Example of Factory Method











# Abstract Factory Pattern

Abstract Factory Pattern says that just **define an interface or abstract class for creating families of related (or dependent) objects but without specifying their concrete sub-classes.**That means Abstract Factory lets a class returns a factory of classes. So, this is the reason that Abstract Factory Pattern is one level higher than the Factory Pattern.

An Abstract Factory Pattern is also known as **Kit.**

#### Advantage of Abstract Factory Pattern

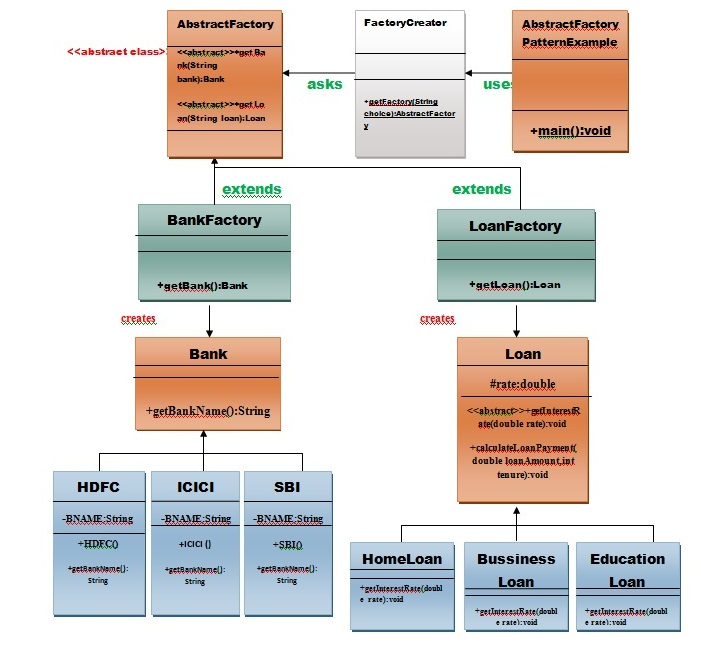
* Abstract Factory Pattern isolates the client code from concrete (implementation) classes.
* It eases the exchanging of object families.
* It promotes consistency among objects.

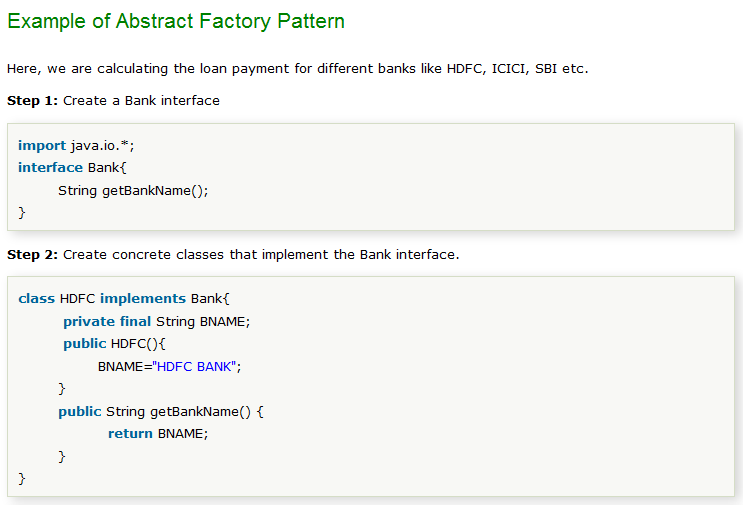
#### Usage of Abstract Factory Pattern

* When the system needs to be independent of how its object are created, composed, and represented.
* When the family of related objects has to be used together, then this constraint needs to be enforced.
* When you want to provide a library of objects that does not show implementations and only reveals interfaces.
* When the system needs to be configured with one of a multiple family of objects.

### UML for Abstract Factory Pattern

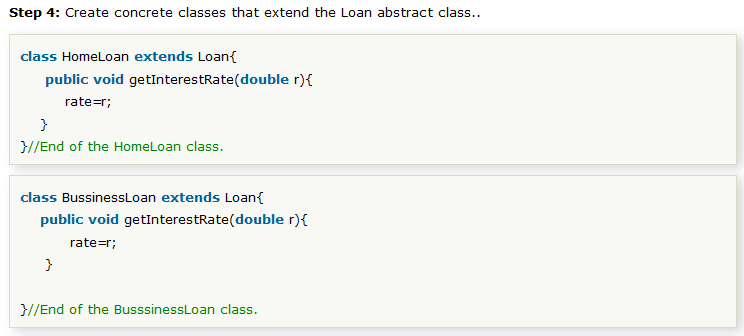
* We are going to create a **Bank interface** and a **Loan abstract class** as well as their sub-classes.
* Then we will create **AbstractFactory** class as next step.
* Then after we will create concrete classes, **BankFactory,** and **LoanFactory** that will extends **AbstractFactory class**
* After that, **AbstractFactoryPatternExample** class uses the **FactoryCreator** to get an object of **AbstractFactory** class.
* See the diagram carefully which is given below



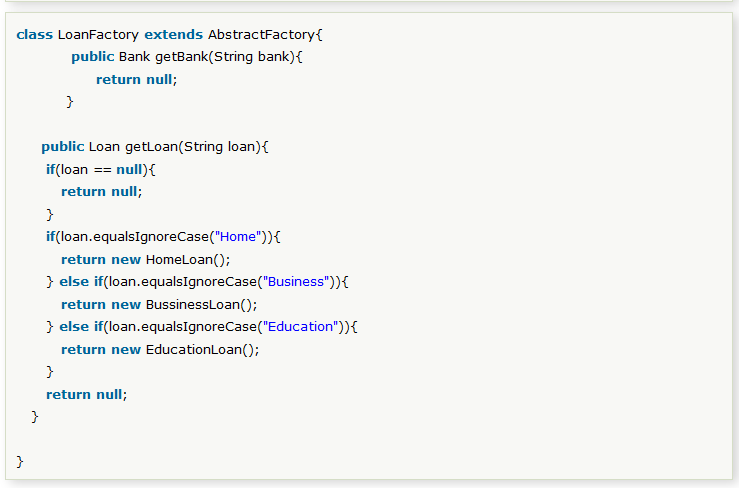


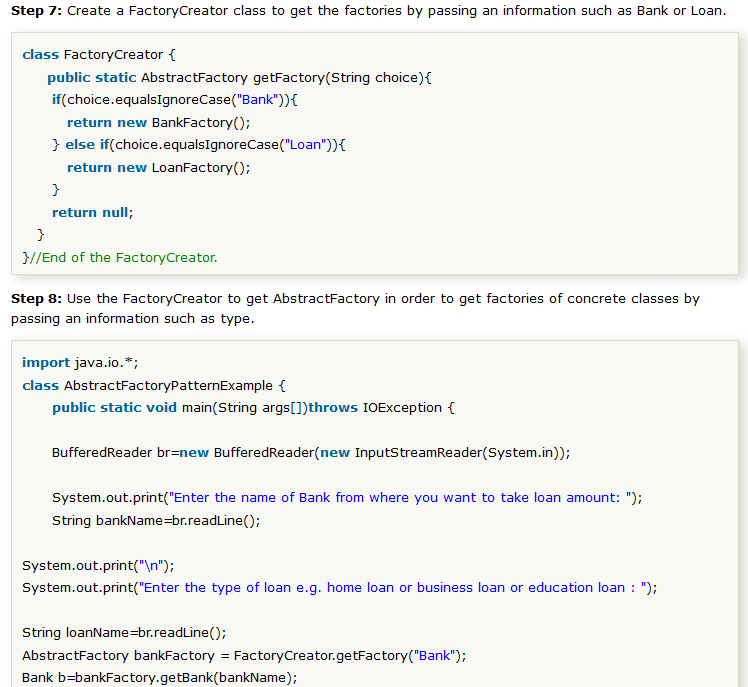




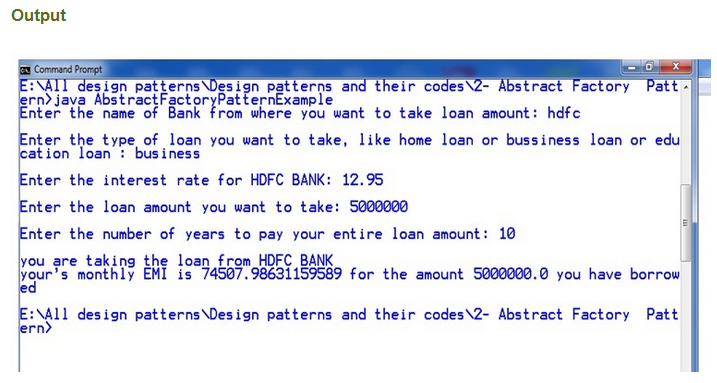












# Singleton design pattern in Java

Singleton Pattern says that just **"define a class that has only one instance and provides a global point of access to it".**

In other words, a class must ensure that only single instance should be created and single object can be used by all other classes.

There are two forms of singleton design pattern

* **Early Instantiation:** creation of instance at load time.
* **Lazy Instantiation:** creation of instance when required.

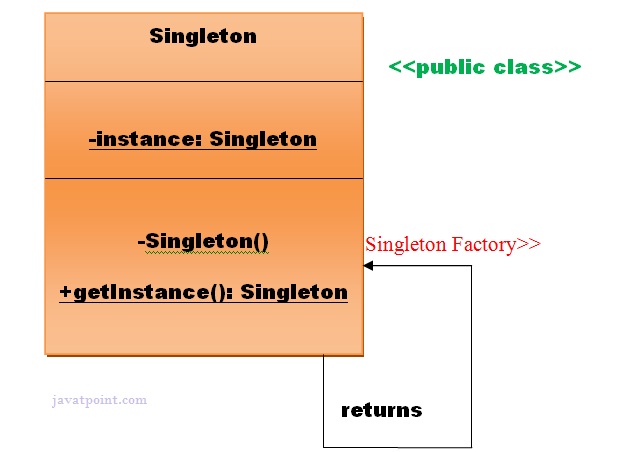
#### Advantage of Singleton design pattern

* Saves memory because object is not created at each request. Only single instance is reused again and again.

#### Usage of Singleton design pattern

* Singleton pattern is mostly used in multi-threaded and database applications. It is used in logging, caching, thread pools, configuration settings etc.

#### Uml of Singleton design pattern



#### How to create Singleton design pattern?

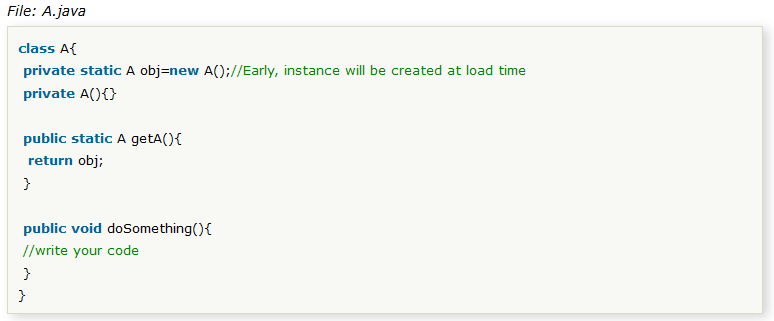
To create the singleton class, we need to have static member of class, private constructor and static factory method.

* **Static member:** It gets memory only once because of static, itcontains the instance of the Singleton class.
* **Private constructor:** It will prevent to instantiate the Singleton class from outside the class.
* **Static factory method:** This provides the global point of access to the Singleton object and returns the instance to the caller.

### Understanding early Instantiation of Singleton Pattern

In such case, we create the instance of the class at the time of declaring the static data member, so instance of the class is created at the time of classloading.

Let's see the example of singleton design pattern using early instantiation.

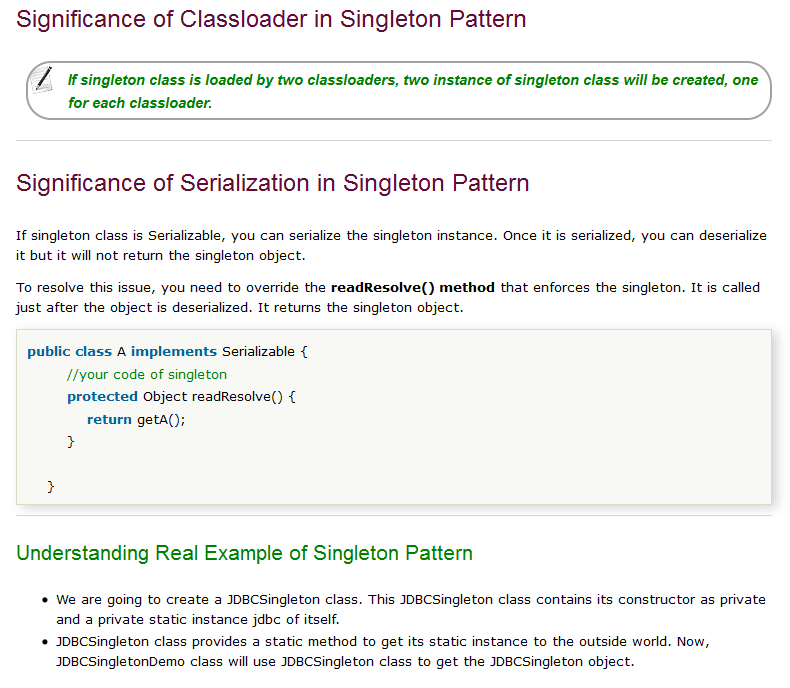


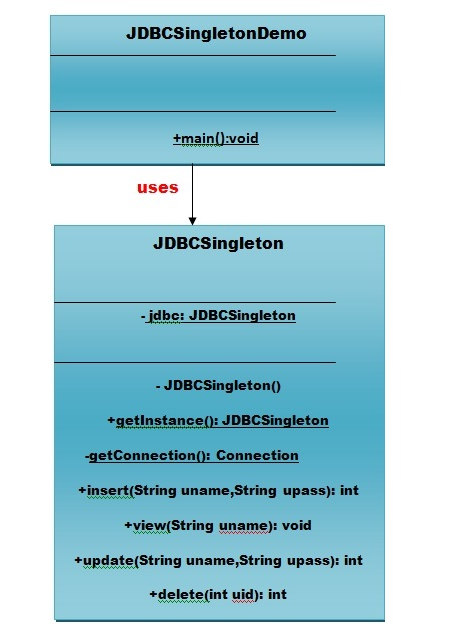
### Understanding lazy Instantiation of Singleton Pattern

In such case, we create the instance of the class in synchronized method or synchronized block, so instance of the class is created when required.

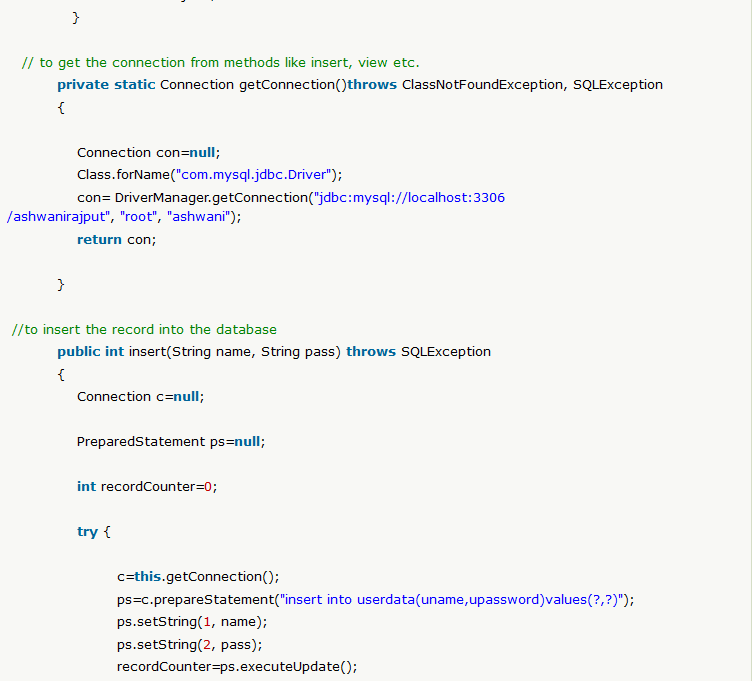
Let's see the simple example of singleton design pattern using lazy instantiation.





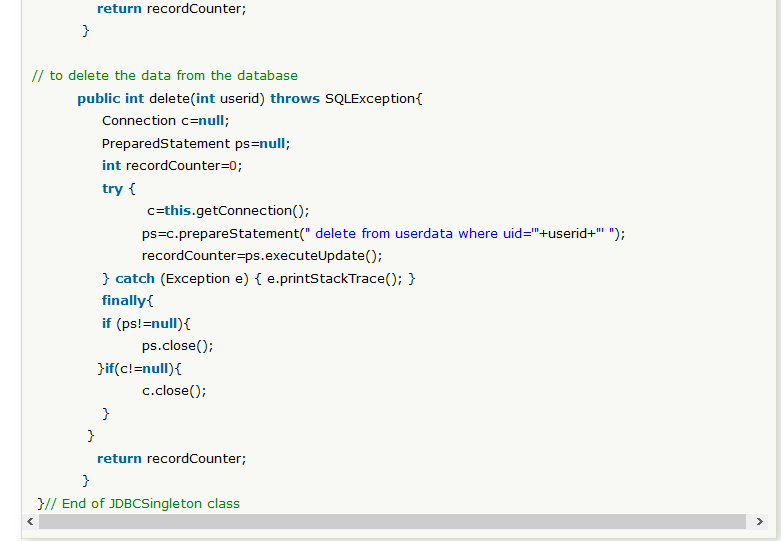










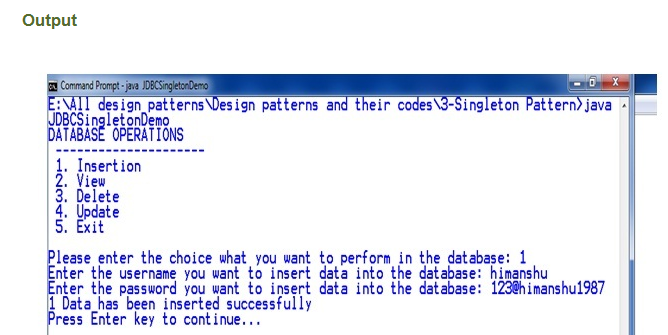


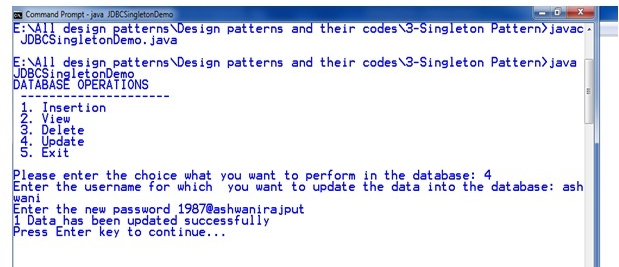


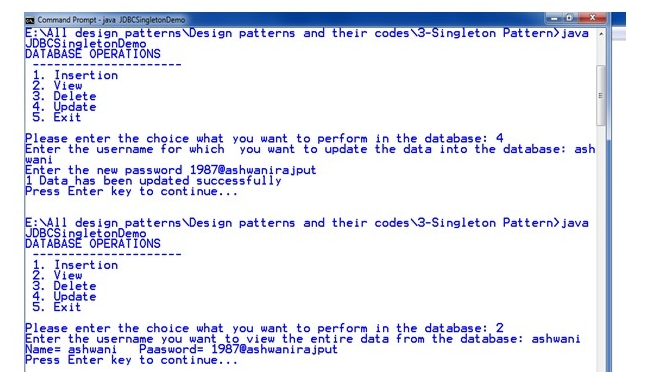












# Prototype Design Pattern

Prototype Pattern says that **cloning of an existing object instead of creating new one and can also be customized as per the requirement**.

This pattern should be followed, if the cost of creating a new object is expensive and resource intensive.

#### Advantage of Prototype Pattern

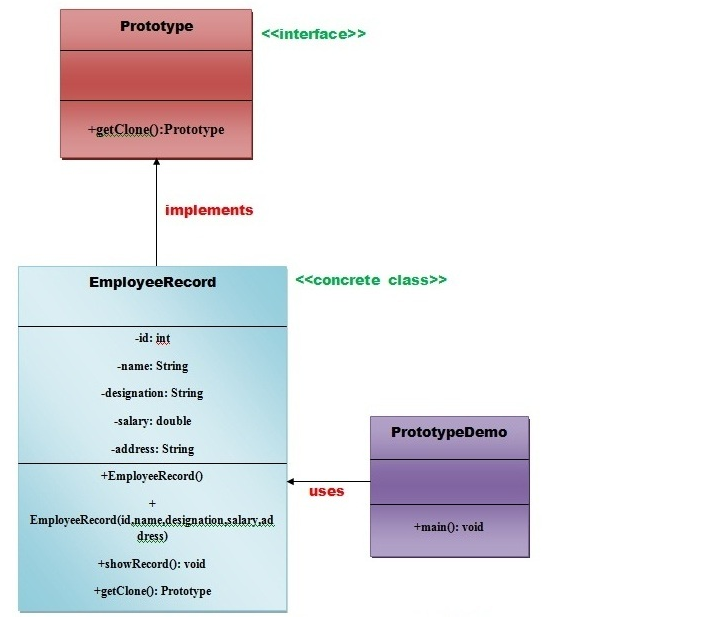
The main advantages of prototype pattern are as follows:

* It reduces the need of sub-classing.
* It hides complexities of creating objects.
* The clients can get new objects without knowing which type of object it will be.
* It lets you add or remove objects at runtime.

#### Usage of Prototype Pattern

* When the classes are instantiated at runtime.
* When the cost of creating an object is expensive or complicated.
* When you want to keep the number of classes in an application minimum.
* When the client application needs to be unaware of object creation and representation.

#### UML for Prototype Pattern



* We are going to create **an interface Prototype** that contains a method **getClone()** of **Prototype type.**
* Then, we create **a concrete class EmployeeRecord** which implements **Prototype interface** that does the cloning of EmployeeRecord object.
* **PrototypeDemo class** will uses this concrete class **EmployeeRecord.**

#### Example of Prototype Design Pattern

