

[JAVA TUTORIAL](#)[#INDEX POSTS](#)[#INTERVIEW QUESTIONS](#)[RESOURCES](#)[HIRE ME](#)[DOWNLOAD ANDROID APP](#)[CONTRIBUTE](#)[HOME](#) » [JAVA](#) » [DESIGN PATTERNS](#) » [JAVA DESIGN PATTERNS – EXAMPLE TUTORIAL](#)

# Java Design Patterns – Example Tutorial

MAY 2, 2018 BY [PANKAJ](#) — [144 COMMENTS](#)

**Design Patterns** are very popular among software developers. A design pattern is a well described solution to a common software problem. I have written extensively on **java design patterns**. You can download PDF eBook (130+ pages) by subscribing to our newsletter.

## Java Design Patterns

Some of the benefits of using design patterns are:

1. Design Patterns are already defined and provides **industry standard approach** to solve a recurring problem, so it saves time if we sensibly use the design pattern. There are many java design patterns that we can use in our java based projects.
2. Using design patterns promotes **reusability** that leads to more **robust** and highly maintainable code. It helps in reducing total cost of ownership (TCO) of the software product.
3. Since design patterns are already defined, it makes our code easy to understand and debug. It leads to faster development and new members of team understand it easily.



**Java Design Patterns** are divided into three categories – **creational**, **structural**, and **behavioral** design patterns. This post serves as an index for all the java design patterns articles I have written so far.

- **Creational Design Patterns**
  - Singleton Pattern
  - Factory Pattern
  - Abstract Factory Pattern
  - Builder Pattern
  - Prototype Pattern
- **Structural Design Patterns**
  - Adapter Pattern
  - Composite Pattern
  - Proxy Pattern
  - Flyweight Pattern
  - Facade Pattern
  - Bridge Pattern
  - Decorator Pattern
- **Behavioral Design Patterns**
  - Template Method Pattern
  - Mediator Pattern
  - Chain of Responsibility Pattern
  - Observer Pattern
  - Strategy Pattern
  - Command Pattern
  - State Pattern
  - Visitor Pattern
  - Interpreter Pattern
  - Iterator Pattern
  - Memento Pattern

## Design Patterns Video Tutorials

Recently I started video tutorials on Design Patterns and they are uploaded on YouTube. Please subscribe to my YouTube channel as I am planning to upload a lot more videos on Core Java, Spring Framework etc.

## Creational Design Patterns

Creational design patterns provide solution to instantiate a object in the best possible way for specific situations.

### 1. Singleton Pattern

**Singleton** pattern restricts the instantiation of a class and ensures that only one instance of the class exists in the java virtual machine. It seems to be a very simple design pattern but when it comes to implementation, it comes with a lot of implementation concerns. The implementation of Singleton pattern has always been a controversial topic among developers. Check out **Singleton Design Pattern** to learn about different ways to implement Singleton pattern and pros and cons of each of the method. This is one of the most discussed java design patterns.

### 2. Factory Pattern

Factory design pattern is used when we have a super class with multiple sub-classes and based on input, we need to return one of the sub-class. This pattern take out the responsibility of instantiation of a class from client program to the factory class. We can apply Singleton pattern on Factory class or make the factory method static. Check out **Factory Design Pattern** for example program and **factory pattern** benefits. This is one of the most widely used java design pattern.

### 3. Abstract Factory Pattern

Abstract Factory pattern is similar to Factory pattern and it's factory of factories. If you are familiar with factory design pattern in java, you will notice that we have a single Factory class that returns the different sub-classes based on the input provided and factory class uses if-else or switch statement to achieve this.

# APIs for Java Developers

Create, Convert, Report and Print: DOC, XLS, PDF, PPT, Image and more [aspose.com](https://aspose.com)



In Abstract Factory pattern, we get rid of if-else block and have a factory class for each sub-class and then an Abstract Factory class that will return the sub-class based on the input factory class. Check out **Abstract Factory Pattern** to know how to implement this pattern with example program.

## 4. Builder Pattern

This pattern was introduced to solve some of the problems with Factory and Abstract Factory design patterns when the Object contains a lot of attributes. Builder pattern solves the issue with large number of optional parameters and inconsistent state by providing a way to build the object step-by-step and provide a method that will actually return the final Object. Check out **Builder Pattern** for example program and classes used in JDK.

## 5. Prototype Pattern

Prototype pattern is used when the Object creation is a costly affair and requires a lot of time and resources and you have a similar object already existing. So this pattern provides a mechanism to copy the original object to a new object and then modify it according to our needs. This pattern uses java cloning to copy the object.

Prototype design pattern mandates that the Object which you are copying should provide the copying feature. It should not be done by any other class. However whether to use shallow or deep copy of the Object properties depends on the requirements and it's a design decision. Check out **Prototype Pattern** for sample program.

# Structural Design Patterns

Structural patterns provide different ways to create a class structure, for example using inheritance and composition to create a large object from small objects.

## 1. Adapter Pattern

Adapter design pattern is one of the structural design pattern and it's used so that two unrelated interfaces can work together. The object that joins these unrelated interface is called an Adapter. As a real life example, we can think of a mobile charger as an adapter because mobile battery needs 3 volts to charge but the normal socket produces either 120V (US) or 240V (India). So the mobile charger works as an adapter between mobile charging socket and the wall socket. Check out **Adapter Pattern** for example program and it's usage in Java.

## 2. Composite Pattern

Composite pattern is one of the Structural design pattern and is used when we have to represent a part-whole hierarchy. When we need to create a structure in a way that the objects in the structure has to be treated the same way, we can apply composite design pattern.

Lets understand it with a real life example – A diagram is a structure that consists of Objects such as Circle, Lines, Triangle etc and when we fill the drawing with color (say Red), the same color also gets applied to the Objects in the drawing. Here drawing is made up of different parts and they all have same operations. Check out [Composite Pattern](#) article for different component of composite pattern and example program.

### 3. Proxy Pattern

Proxy pattern intent is to "Provide a surrogate or placeholder for another object to control access to it". The definition itself is very clear and proxy pattern is used when we want to provide controlled access of a functionality.

Let's say we have a class that can run some command on the system. Now if we are using it, it's fine but if we want to give this program to a client application, it can have severe issues because client program can issue command to delete some system files or change some settings that you don't want. Check out [Proxy Pattern](#) post for the example program with implementation details.

### 4. Flyweight Pattern

Flyweight design pattern is used when we need to create a lot of Objects of a class. Since every object consumes memory space that can be crucial for low memory devices, such as mobile devices or embedded systems, flyweight design pattern can be applied to reduce the load on memory by sharing objects. String Pool implementation in java is one of the best example of Flyweight pattern implementation. Check out [Flyweight Pattern](#) article for sample program and implementation process.

### 5. Facade Pattern

Facade Pattern is used to help client applications to easily interact with the system. Suppose we have an application with set of interfaces to use MySQL/Oracle database and to generate different types of reports, such as HTML report, PDF report etc. So we will have different set of interfaces to work with different types of database. Now a client application can use these interfaces to get the required database connection and generate reports. But when the complexity increases or the interface behavior names are confusing, client application will find it difficult to manage it. So we can apply Facade pattern here and provide a wrapper interface on top of the existing interface to help client application. Check out [Facade Pattern](#) post for implementation details and sample program.

### 6. Bridge Pattern

When we have interface hierarchies in both interfaces as well as implementations, then bridge design pattern is used to decouple the interfaces from implementation and hiding the implementation details

from the client programs. Like Adapter pattern, it's one of the Structural design pattern.

The implementation of bridge design pattern follows the notion to prefer Composition over inheritance. Check out [Bridge Pattern](#) post for implementation details and sample program.

## 7. Decorator Pattern

Decorator design pattern is used to modify the functionality of an object at runtime. At the same time other instances of the same class will not be affected by this, so individual object gets the modified behavior. Decorator design pattern is one of the structural design pattern (such as Adapter Pattern, Bridge Pattern, Composite Pattern) and uses abstract classes or interface with composition to implement.

We use inheritance or composition to extend the behavior of an object but this is done at compile time and it's applicable to all the instances of the class. We can't add any new functionality or remove any existing behavior at runtime – this is when Decorator pattern comes into picture. Check out [Decorator Pattern](#) post for sample program and implementation details.

# Behavioral Design Patterns

Behavioral patterns provide solution for the better interaction between objects and how to provide loose coupling and flexibility to extend easily.

## 1. Template Method Pattern

Template Method is a behavioral design pattern and it's used to create a method stub and deferring some of the steps of implementation to the subclasses. Template method defines the steps to execute an algorithm and it can provide default implementation that might be common for all or some of the subclasses.

Suppose we want to provide an algorithm to build a house. The steps need to be performed to build a house are – building foundation, building pillars, building walls and windows. The important point is that we can't change the order of execution because we can't build windows before building the foundation. So in this case we can create a template method that will use different methods to build the house. Check out [Template Method Pattern](#) post for implementation details with example program.

## 2. Mediator Pattern

Mediator design pattern is used to provide a centralized communication medium between different objects in a system. Mediator design pattern is very helpful in an enterprise application where multiple objects are interacting with each other. If the objects interact with each other directly, the system components are tightly-coupled with each other that makes maintainability cost higher and not flexible to extend easily. Mediator pattern focuses on provide a mediator between objects for communication and help in implementing loose-coupling between objects.

Air traffic controller is a great example of mediator pattern where the airport control room works as a mediator for communication between different flights. Mediator works as a router between objects and it can have its own logic to provide way of communication. Check out [Mediator Pattern](#) post for implementation details with example program.

### 3. Chain of Responsibility Pattern

Chain of responsibility pattern is used to achieve loose coupling in software design where a request from client is passed to a chain of objects to process them. Then the object in the chain will decide themselves who will be processing the request and whether the request is required to be sent to the next object in the chain or not.

We know that we can have multiple catch blocks in a try-catch block code. Here every catch block is kind of a processor to process that particular exception. So when any exception occurs in the try block, it's sent to the first catch block to process. If the catch block is not able to process it, it forwards the request to next object in chain i.e next catch block. If even the last catch block is not able to process it, the exception is thrown outside of the chain to the calling program.

ATM dispense machine logic can be implemented using [Chain of Responsibility Pattern](#), check out the linked post.

### 4. Observer Pattern

Observer design pattern is useful when you are interested in the state of an object and want to get notified whenever there is any change. In observer pattern, the object that watch on the state of another object are called **Observer** and the object that is being watched is called **Subject**.

Java provides inbuilt platform for implementing Observer pattern through `java.util.Observable` class and `java.util.Observer` interface. However it's not widely used because the implementation is really simple and most of the times we don't want to end up extending a class just for implementing Observer pattern as java doesn't provide multiple inheritance in classes.

Java Message Service (JMS) uses Observer pattern along with Mediator pattern to allow applications to subscribe and publish data to other applications. Check out [Observer Pattern](#) post for implementation details and example program.

### 5. Strategy Pattern

Strategy pattern is used when we have multiple algorithm for a specific task and client decides the actual implementation to be used at runtime.

Strategy pattern is also known as Policy Pattern. We defines multiple algorithms and let client application pass the algorithm to be used as a parameter. One of the best example of this pattern is `Collections.sort()` method that takes `Comparator` parameter. Based on the different implementations of `Comparator` interfaces, the Objects are getting sorted in different ways.

Check out [Strategy Pattern](#) post for implementation details and example program.

## 6. Command Pattern

Command Pattern is used to implement loose coupling in a request-response model. In command pattern, the request is sent to the invoker and *invoker* pass it to the encapsulated *command* object. Command object passes the request to the appropriate method of *Receiver* to perform the specific action.

Let's say we want to provide a File System utility with methods to open, write and close file and it should support multiple operating systems such as Windows and Unix.

To implement our File System utility, first of all we need to create the receiver classes that will actually do all the work. Since we code in terms of java interfaces, we can have FileSystemReceiver interface and its implementation classes for different operating system flavors such as Windows, Unix, Solaris etc. Check out [Command Pattern](#) post for the implementation details with example program.

## 7. State Pattern

State design pattern is used when an Object change its behavior based on its internal state.

If we have to change the behavior of an object based on its state, we can have a state variable in the Object and use if-else condition block to perform different actions based on the state. State pattern is used to provide a systematic and loose-coupled way to achieve this through Context and State implementations.

Check out [State Pattern](#) post for implementation details with example program.

## 8. Visitor Pattern

Visitor pattern is used when we have to perform an operation on a group of similar kind of Objects. With the help of visitor pattern, we can move the operational logic from the objects to another class.

For example, think of a Shopping cart where we can add different type of items (Elements), when we click on checkout button, it calculates the total amount to be paid. Now we can have the calculation logic in item classes or we can move out this logic to another class using visitor pattern. Let's implement this in our example of visitor pattern. Check out [Visitor Pattern](#) post for implementation details.

## 9. Interpreter Pattern

is used to defines a grammatical representation for a language and provides an interpreter to deal with this grammar.



The best example of this pattern is java compiler that interprets the java source code into byte code that is understandable by JVM. Google Translator is also an example of interpreter pattern where the input can be in any language and we can get the output interpreted in another language.

Check out [Interpreter Pattern](#) post for example program.

## 10. **Iterator Pattern**

Iterator pattern is one of the behavioral pattern and it's used to provide a standard way to traverse through a group of Objects. Iterator pattern is widely used in [Java Collection Framework](#) where Iterator interface provides methods for traversing through a collection.

Iterator pattern is not only about traversing through a collection, we can provide different kind of iterators based on our requirements. Iterator pattern hides the actual implementation of traversal through the collection and client programs just use iterator methods. Check out [Iterator Pattern](#) post for example program and implementation details.

## 11. **Memento Pattern**

Memento design pattern is used when we want to save the state of an object so that we can restore later on. Memento pattern is used to implement this in such a way that the saved state data of the object is not accessible outside of the object, this protects the integrity of saved state data.

Memento pattern is implemented with two objects – Originator and Caretaker. Originator is the object whose state needs to be saved and restored and it uses an inner class to save the state of Object. The inner class is called Memento and it's private, so that it can't be accessed from other objects.

Check out [Memento Pattern](#) for sample program and implementation details.

That's all for different design patterns in java, this post intent is to provide an index to browse all of them easily.

FILED UNDER: [DESIGN PATTERNS](#)

### About Pankaj

If you have come this far, it means that you liked what you are reading. Why not reach little more and connect with me directly on [Google Plus](#), [Facebook](#) or [Twitter](#). I would love to hear your thoughts and opinions on my articles directly.

Recently I started creating video tutorials too, so do check out my videos on [Youtube](#).

[« Visitor Design Pattern in Java](#)

[Java Dependency Injection – DI Design Pattern Example Tutorial »](#)

## Comments

### Rimpi says

[JULY 4, 2018 AT 8:49 PM](#)

B. B

LL. BL

UUU. BLU

EEEE. BLUE

JJJJJ BLUEJ

How to print this pattern

[Reply](#)

**Abbin Varghese says**

MAY 15, 2018 AT 3:13 AM

Very nice article. Thanks for all the information.

One thing i missed in these is that, the explanations are according to the package structure. I think first we should think about the use case, and then the pattern.

For eg. "Mediator design pattern is very helpful in an enterprise application where multiple objects are interacting with each other." But when do we decide objects should interact with each other and why not Observer Pattern or Visitor Pattern? What we should or shouldn't use this patterns from a real life scenario ?

If these informations are added, this will be a perfect article.. ☐

[Reply](#)**KRISHNA KANT says**

JANUARY 8, 2018 AT 10:41 AM

How can I download ebook for of this tutorial ?

[Reply](#)**Iqbal Hossain says**

MAY 13, 2018 AT 7:53 PM

How can I download ebook for of this tutorial ?

[Reply](#)**Abdullah says**

MAY 22, 2018 AT 11:53 PM

How can I download ebook for of this tutorial ?

[Reply](#)**pratk says**

JULY 5, 2018 AT 8:55 AM

How can I download ebook for of this tutorial?

[Reply](#)**Pankaj says**

JULY 5, 2018 AT 9:39 AM

Subscribe to the newsletter and you will get the eBook in the email.

**Kiran Kanaparthi says**

DECEMBER 31, 2017 AT 2:00 PM

Can You sort the Design pattern names Alphabetically(just like the in GOF book), then it will be easier to follow them and memorize.

[Reply](#)**xubing says**

NOVEMBER 27, 2017 AT 7:16 PM

could you send the pdf ebook for me

[Reply](#)**Uday says**

NOVEMBER 13, 2017 AT 8:16 AM

How do we adapt the version changes of common library like Apache in our existing code ?

For example , if I am using method of some common library in multiple classes and method signature get changed in upgrade version ?

what is the best solution not to make changes in all classes rather then in single point ?

My guess : Creat a separate class and use the common method inside the class and use this class reference everywhere .

[Reply](#)**Nishant says**

JUNE 1, 2018 AT 3:57 AM

Adapter patter serves best here , the client program would adapt to the changes via the adapter class.

[Reply](#)**Nishant says**

JUNE 1, 2018 AT 3:59 AM

An adapter pattern would server best here. I was a bit surprised as the Apache libs are usually backward compatible.

[Reply](#)**novapattern.com says**

NOVEMBER 9, 2017 AT 1:11 PM

This is by far the most comprehensive tutorial / guide I've come across.

[Reply](#)**Emanuel says**

AUGUST 9, 2017 AT 6:09 PM

How to do this please is very important

WINDHOEK

WINDHOE

WINDHO

WINDH

WIND

WIN

WI

W

L

LI

LIN

LING

LINGU

LINGUA

[Reply](#)**SURAJ says**

SEPTEMBER 5, 2017 AT 1:50 AM

```
s="WINDHOEK"
```

```
t="LINGUA"
```

```
for i in range(0,len(s)):
```

```
for j in range(0,len(s)-i):
```

```
print(s[j],end=" ")
```

```
if i!=len(s)-1:
```

```
print()
```

```
for i in range(0,len(t)+1):
```

```
for j in range(0,i):
```

```
print(tlj,end=" ")
```

```
print()
```

[Reply](#)**Roy El Asmar says**

OCTOBER 5, 2017 AT 2:39 AM

```
public class MyClass {  
    public static void main(String args[]) {  
        String s1="WINDHOEK";  
        String s2="LINGUA";  
        for(int i=s1.length()-1;i>=0;i--){  
            System.out.println(s1.substring(0,i+1));  
        }  
        for(int j=0;j<=s2.length()-1;j++){  
            System.out.println(s2.substring(0,j+1));  
        }  
    }  
}
```

[Reply](#)**Abhishek says**

MAY 31, 2018 AT 7:50 PM

```
String a="WINDHOEK";  
String b="LINGUA";  
String s="";  
int temp=a.length();  
for(int i=a.length()-1;i>=0;i--)  
{  
    for(int j=0;j<=i;j++)  
    {  
        s=s+a.charAt(j);  
    }  
    System.out.println(s);  
    s="";  
}  
s="";  
for(int i=0;i<0;j--)  
{  
    s=b.charAt(j)+s;  
}
```

```
System.out.println(s);
```

```
s="";
```

```
}
```

[Reply](#)**Geethu says**

JUNE 27, 2017 AT 11:09 PM

Hi Pankaj,

Consider a situation where our management has decided to integrate a new jar instead of old one. The new jar provides the same set of operations which was used by old jar, but the number of arguments or the datatype of the argument ( header parameter of the function) is different from the existing. It will be very difficult to change the method parameters across the application.

Which design pattern suits well in this case with minimum number of modification in the application?

Thanks,

Geethu

[Reply](#)**Amit Bhatia says**

AUGUST 10, 2017 AT 4:34 AM

Adapter

[Reply](#)**Stefan says**

MAY 9, 2017 AT 7:17 AM

Slight typo on the section for Bridge Pattern it says " When we have interface hierarchies in both interfaces as well as implementations, then builder design pattern "..... but we are talking about the Bridget Pattern and Structural Patterns

[Reply](#)**Pankaj says**

MAY 9, 2017 AT 9:03 AM

Thanks for the tip, corrected the typo error.

[Reply](#)

**Arsallmam says**

FEBRUARY 10, 2017 AT 2:44 PM

Great blog

[Reply](#)**Luke Fan says**

FEBRUARY 7, 2017 AT 9:20 PM

It's good for me. Thank you for these.

[Reply](#)**tathagata bhattacharya says**

DECEMBER 28, 2016 AT 2:23 AM

I need help.Can you help me in this:

15 14 13 12 11

10 9 8 7

6 5 4

3 2

1

[Reply](#)**Anonymous says**

JANUARY 23, 2017 AT 3:39 AM

```
void main()
{
    int i,j,a=15;
    for(i =1;i=1;j--)
    {
        printf("%d",a);
        a--;
    }
    printf("\n");
}
```

[Reply](#)**Maruthi says**

JUNE 20, 2017 AT 10:19 PM



```
void main()
{
int i,j,a=15;
for(i =5;i<1;i--)
{
for(j =1;j<i;j++)
{
printf("%d",a);
a--;
}
printf("\n");
}
}
```

[Reply](#)

#### **Srikanth says**

DECEMBER 5, 2017 AT 1:37 AM

```
public static void main(String[] args) {
Scanner s = new Scanner(System.in);
System.out.println("Enter the number");
int value = Integer.parseInt(s.next());
int numberOfNumbers = 0;
int count=1;
int row =1;
for(int i = 1;i<=value;i++){
row =i;
if(i%2!=0){
numberOfNumbers = i * count;
} else {
numberOfNumbers = (i*count)+count;
count++;
}
if(numberOfNumbers == value){
break;
} else if (numberOfNumbers > value){
value=numberOfNumbers;
break;
}
}
System.out.println("all "+numberOfNumbers+" numbers to be displayed in number of rows is "+row);
for(int i=row; i>=1;i--){
for(int rows=1;rows<=i;rows++){
```

```
System.out.print(numberOfNumbers+" ");  
}  
System.out.println();  
}  
}
```

[Reply](#)**anjali says**

NOVEMBER 16, 2016 AT 9:29 PM

Hi i need the ebook  
java design patterns

[Reply](#)**Nikhil Aggarwal says**

DECEMBER 31, 2016 AT 3:52 AM

Download Head first design patterns book.

[Reply](#)**Ranga Kalyan says**

SEPTEMBER 7, 2016 AT 5:38 PM

This blog post on design patterns is great! I particularly like your links that show examples. Very helpful. I use a lot of design patterns in my Java work. I come to your blog often, to clarify their differences. Thank you for posting it.

Ranga

[Reply](#)**Ranga says**

AUGUST 28, 2016 AT 1:43 PM

Great post.. Thanks!

[Reply](#)**Balaji says**

AUGUST 23, 2016 AT 6:45 AM

Hi Pankaj,

I just want to thank you. The work you are doing so great, it's making developers life easier.

Thank a lot for your great work again.

One query i have that might be wrong but just to clarify:

1. In Decorator Pattern – We are using inheritance or aggregation to extend the behavior of an object.... and not the "Composition", I think Aggregation is correct to say here than Composition.

what you say?

[Reply](#)

**a says**

AUGUST 10, 2016 AT 7:00 AM

how to print

1

1 2 1

1 2 3 2 1

1 2 3 4 3 2 1

1 2 3 2 1

1 2 1

1

[Reply](#)

**Lalit Sharma says**

AUGUST 11, 2016 AT 5:29 AM

```
public void printStars(int num, int limit)
```

```
{
```

```
    if (num > limit)
```

```
        return;
```

```
    else
```

```
    {
```

```
        for (int q = 1; q <= num; q++){
```

```
            System.out.print(q);
```

```
        }
```

```
        System.out.print("\n");
```

```
        printStars(num +1, limit);
```

```
        for (int q = 1; q <= num; q++){
```

```
            System.out.print(q);
```

```
        }
```

```
        System.out.print("\n");
```

```
    }
```

```
}
```

[Reply](#)**Veera says**

JANUARY 18, 2017 AT 5:03 AM

```
public void printStars(int num, int limit)
{
    int count =0;
    if (num > limit)
        return;
    else
    {
        for (int q = 1; q <=limit; q++)
        {
            System.out.print(q-1);
            count++;
        }
        if(count==(limit-1)){
        }
        else{
            System.out.print("\n");
        }
        printStars(num +1, limit);
        for (int q = 1; q <=limit; q++)
        {
            if(count==(limit-1)){
            }
            else{
                System.out.print((q-1));
            }
        }
        if(count==(limit-1)){
            System.out.print("\n");
        }
        else{
            System.out.print("\n");
        }
    }
}
```

[Reply](#)**Veera says**

JANUARY 18, 2017 AT 5:04 AM

Please use the above code to display the required format

[Reply](#)

**vinay says**

SEPTEMBER 25, 2017 AT 9:23 PM

still i am getting errors in program

[Reply](#)

**Padam Dhariwal says**

AUGUST 9, 2016 AT 7:45 AM

Very nice articles.

[Reply](#)

**Prashanth says**

JULY 18, 2016 AT 12:46 AM

Brother, add examples or link it in another page. It would be quite helpful.

[Reply](#)

**Prashanth says**

JULY 18, 2016 AT 12:48 AM

Nice examples and easily understandable.

Seems links are not working only for me.

[Reply](#)

**Pankaj says**

JULY 18, 2016 AT 1:38 AM

can you tell me which link is not working?

[Reply](#)

**Vikram says**

JULY 16, 2016 AT 10:32 AM

Very Nice article and well written with very good examples

[Reply](#)**Atul says**

JULY 7, 2016 AT 2:54 AM

Pankaj,

Write a program to print this series.

129876

245335

483728

092872

387763

[Reply](#)**Hayawan says**

NOVEMBER 24, 2016 AT 12:07 PM

```
public class numbers () {  
    system.out.println (129876 "+" 245335 "+" 483728 "+" 092872 "+" 387763);  
    return null;  
}
```

[Reply](#)**Hayawan says**

NOVEMBER 24, 2016 AT 12:41 PM

```
public class Numbers {  
    public static void main(String[] args) {  
        System.out.println(129876);  
        System.out.println(245335);  
        System.out.println(483728);  
        System.out.println(992872);  
    }  
};
```

this is the right way, i was rushing before, sorry.

[Reply](#)

**chetan says**

JUNE 14, 2016 AT 3:09 AM

Hi Pankaj,

Singleton Design Pattern hyperlink not working.

[Reply](#)**Pankaj says**

JUNE 14, 2016 AT 5:29 AM

Thanks Chetan for noticing and pointing out, I was doing some post update and mistakenly removed the link. I have corrected it. Thanks again.

[Reply](#)**Shailendra says**

JULY 2, 2016 AT 2:22 PM

0 1 2 3

0 1 2

0 1

0

0 1

0 1 2

0 1 2 3

[Reply](#)**Shailendra says**

JULY 2, 2016 AT 2:31 PM

4

3 4

2 3 4

1 2 3 4

0 1 2 3 4

1 2 3 4

2 3 4

3 4

4

[Reply](#)

**Boovaragan says**

JUNE 10, 2016 AT 6:58 AM

Hi Sir,

what about Service Locator patter?

under which category will it come?

[Reply](#)**Vijay Kumar says**

MAY 20, 2016 AT 3:56 AM

Awesome articles on Design patterns so far I have read.

[Reply](#)**Pankaj says**

MAY 28, 2016 AT 10:46 AM

Thanks Vijay, make sure to get the eBook by subscribing to the newsletter.

[Reply](#)**Java Learner says**

APRIL 26, 2016 AT 5:43 AM

Guru,

This is great, no words the way you wrote the article.

Love it, Love it, Love it.

Thanks

[Reply](#)**Pankaj says**

MAY 28, 2016 AT 10:46 AM

You made my day brother.

[Reply](#)**Don says**

APRIL 4, 2016 AT 11:24 PM



Help full

[Reply](#)

**Pankaj says**

MAY 28, 2016 AT 10:46 AM

Thanks Don, appreciate the nice words.

[Reply](#)

**ejakhan says**

MARCH 27, 2016 AT 12:48 AM

1 2 3 4 5

10 9 8 7 6

11 12 13 14 15

20 19 18 17 16

[Reply](#)

**Pankaj says**

MAY 28, 2016 AT 10:47 AM

I don't understand what you are asking, and how it's anywhere related to java design patterns.

[Reply](#)

**shreya says**

MARCH 26, 2016 AT 8:26 AM

1 3 5 7 9

3 5 7 9 1

5 7 9 1 3

7 9 1 3 5

9 1 3 5 7

help please!!!!

[Reply](#)

**Suyash says**

MAY 23, 2016 AT 12:37 AM

```
int[] arr = new int[]{1, 3, 5, 7, 9};
```

```
for (int i = 0; i < 5; i++) {
```

```
for (int j = 0; j < arr.length; j++) {  
    System.out.print(arr[j] % 10 + " ");  
    arr[j] = arr[j] + 2;  
}  
System.out.println("\n");  
}
```

[Reply](#)**Pankaj says**

MAY 28, 2016 AT 10:48 AM

Thanks Suyash for responding and helping Shreya.

[Reply](#)**maresh says**

MARCH 16, 2016 AT 11:36 PM

wow Super Pankaj ,very easy to understand

[Reply](#)**Pankaj says**

MAY 28, 2016 AT 10:48 AM

Thanks Mahesh, get the eBook too. ☐

[Reply](#)**Santeepa says**

FEBRUARY 29, 2016 AT 6:40 AM

plzz tell me how to do this

NO Data Pattern

1 5

3 5 10

5 5 10 15

[Reply](#)**Vishnu says**

MARCH 13, 2016 AT 11:27 PM

```
public class Pattern {  
    static int c = 1;  
    static int p = 5;  
    static int x = 2;  
    public static void main(String[] args) {  
        for (int i = 2; i < 5; i++) {  
            for (int j = 0; j < x; j++) {  
                if (j == 0)  
                    System.out.print(c);  
                else {  
                    System.out.print(p);  
                    p += 5;  
                }  
                System.out.print("\t");  
            }  
            c += 2;  
            System.out.print("\n");  
            x++;  
            p = 5;  
        }  
    }  
}
```

[Reply](#)

**Pankaj says**

MAY 28, 2016 AT 10:49 AM

Thanks Vishnu for responding and quick solution, I hope it helps Santeepa.

[Reply](#)

**khushi says**

APRIL 19, 2017 AT 8:45 PM

plz solve it anybody

if user gives(3,2)as an input then the output should be

7 9 11

3 5

1

1

3 5

plz solve this question in and share thiis code.

[Reply](#)**Kush says**

JANUARY 15, 2016 AT 11:50 AM

Very simple examples and nicely explained. It would not have been possible to learn design patterns without your book. Thanks a ton for your effort sir.

[Reply](#)**Pankaj says**

MAY 28, 2016 AT 10:50 AM

Wow, one of the nicest comment I have got recently.

[Reply](#)**Chirag Sharma says**

DECEMBER 27, 2015 AT 9:54 PM

There should be some real time problem also as a example to complete this tutorial.

[Reply](#)**Pankaj says**

MAY 28, 2016 AT 10:52 AM

If I will provide real life problem, it will become too specific and too lengthy. Also there are many ways to solve a real life problem, it depends on your project specific situation. You should learn these design patterns and use them wisely. You should not use a design pattern just because you know it.

Always remember, simple code is the best code.

[Reply](#)**Vidita Daga says**

OCTOBER 19, 2015 AT 9:33 AM

Thank you so much...This is really helpful to learn.

[Reply](#)

**Pankaj says**

MAY 28, 2016 AT 10:52 AM

Appreciate your lovely words, Vidita. ☐

[Reply](#)**suresh atta says**

SEPTEMBER 4, 2015 AT 7:28 PM

Simply awesome. I like your write ups and examples.

[Reply](#)**Pankaj says**

MAY 28, 2016 AT 10:53 AM

Thanks Suresh, I hope you will like my eBook for design patterns too. It's absolutely free for my email subscribers.

[Reply](#)**Himalaya says**

APRIL 30, 2015 AT 8:03 AM

Hi, Blog information is really nice. Please provide a "Scroll to Top" option in your website. Many thanks ☐

[Reply](#)**Pankaj says**

MAY 28, 2016 AT 10:54 AM

Hmm, nice suggestion. I want to do that but it will add some more resources and slow down page speed. I have kept bare minimum things to make sure page load fast for everyone. ☐

[Reply](#)**Ranga says**

MARCH 23, 2015 AT 10:16 PM

Great work. I'm a great believer that understanding the context where a Design Pattern is applicable is more important than the actual implementation details. This article gives a good overview. I think this video will also be a good guide. [Video](#)

[Reply](#)**Pankaj says**

MAY 28, 2016 AT 10:57 AM

Thanks for the nice words Ranga.

[Reply](#)**Aryan says**

JANUARY 12, 2015 AT 12:46 PM

Nice article, good examples, easy to follow. I am using it to prepare for Software Engineering exam which is in two days. Thank you.

[Reply](#)**Pankaj says**

MAY 28, 2016 AT 10:57 AM

I am replying late but I wish you did well.

[Reply](#)**sree says**

JANUARY 12, 2015 AT 12:32 AM

is MVC pattern belongs to which category?

[Reply](#)**Pankaj says**

MAY 28, 2016 AT 10:58 AM

It belongs to Java EE patterns.

[Reply](#)**Neeraj says**

JANUARY 8, 2015 AT 3:05 AM

In One For Loop

```
public class Test {  
    public static void main(String arg[]){  
        int c =1;  
        int d =1;  
        int till =15;  
        for(int b=till;b>0;b--){  
            System.out.print(b+" ");  
            if(d==c){  
                System.out.println("");  
                c++;  
                d=0;  
            }  
            d++;  
        }  
    }  
}
```

[Reply](#)

**Pankaj says**

JANUARY 8, 2015 AT 5:00 AM

What is your query here?

[Reply](#)

**Atmprakash Sharma says**

NOVEMBER 20, 2014 AT 10:07 PM

nice ..well manged data...i read most of your blog and get confident in java....keep it up....

[Reply](#)

**Pankaj says**

MAY 28, 2016 AT 10:59 AM

Thanks Atmprakash, good to hear from a regular read.

[Reply](#)

**Oleg says**

OCTOBER 28, 2014 AT 5:11 AM

It would be creat to see tutotials about DAO and MVC patterns and how to use them with JDBC.

[Reply](#)**Pankaj says**

MAY 28, 2016 AT 11:01 AM

Added to my TODO list, will publish it soon.

[Reply](#)**abhinav says**

SEPTEMBER 28, 2016 AT 7:44 AM

still waiting for the same.

[Reply](#)**j says**

OCTOBER 23, 2014 AT 5:46 PM

Simple explanations on common design patterns. Thanks!

[Reply](#)**Pankaj says**

MAY 28, 2016 AT 11:01 AM

You are welcome 'J'. ☐

[Reply](#)**Tanmoy says**

OCTOBER 21, 2014 AT 11:25 PM

how to do this program

1 2 3 4 5

6 7 8 9

10 11 12

13 14

15

[Reply](#)



**rao arsalan says**

DECEMBER 22, 2014 AT 3:53 AM

here is your program!!!!!!

```
class Pattern{
public static void main(String a[]){
int k=5;
int i,j;
int z;
for( i=1;i<=15;i=z){
for( j=i;j<=k;j++){
System.out.print(j);}
System.out.println();
k=(k+j)-i-1;
z=j;
}
}
}
```

[Reply](#)**Ganesh Ahiwale says**

DECEMBER 23, 2014 AT 4:11 AM

```
cnt=1;
for(i=n,i<0;n-)
{
for(j=0;j<i;j++)
{
System.out.print(cnt);
}
System.out.println();
cnt=cnt+1;
}
```

[Reply](#)**Pankaj says**

MAY 28, 2016 AT 11:02 AM

Thanks guys for pitching in and helping Tanmoy.

[Reply](#)

**jayesh says**

OCTOBER 15, 2014 AT 3:32 AM

how to do

1

24

369

481216

[Reply](#)**Vijay Bharwani says**

JANUARY 5, 2015 AT 2:47 AM

I am not sure whether you solved it or not. But it is pretty simple. Below is the program for this pattern

```
public static void main(String args[]) {  
    int lines = 4;  
    for (int i = 1; i <= lines; i++) {  
        for (int j = 1; j <= i; j++) {  
            System.out.print(i*j);  
        }  
        System.out.println();  
    }  
}
```

[Reply](#)**Pankaj says**

MAY 28, 2016 AT 11:02 AM

i hope your program helped Jayesh.

[Reply](#)**Ashakant says**

OCTOBER 6, 2014 AT 11:00 AM

This is also very nice simple explanation :

```
public interface Duck  
{  
    public void quack();  
}
```

```
public void fly() ;
}

public interface Turkey
{
    public void gobble() ;
    public void fly() ;
}

public class TurkeyAdapter implements Duck
{
    private Turkey mTurkey;
    public void quack() {
        // TODO Auto-generated method stub
        mTurkey.gobble() ;
    }
    public void fly() {
        // TODO Auto-generated method stub
        for(int i=0;i<5;i++)
            mTurkey.fly() ;
    }
}

TurkeyAdapter link between Duck and Turkey
```

[Reply](#)**Pankaj says**

MAY 28, 2016 AT 11:06 AM

You need implementation of Turkey interface and then inject it to TurkeyAdapter private variable mTurkey by setter method or through constructor. My 2 cents, however I am not sure if there was a query here. 😊

[Reply](#)**Chidambar Dorairaj says**

SEPTEMBER 8, 2014 AT 12:11 AM

Awesome tutorials Pankaj. Keep up the good work.

[Reply](#)**Pankaj says**

MAY 28, 2016 AT 11:06 AM

Thanks Chidambar, appreciate nice words.

[Reply](#)

**groovy says**

AUGUST 19, 2014 AT 10:02 PM

Some developers becomes so obsessed with design patterns that they over-engineer.

[Reply](#)**vijay says**

SEPTEMBER 16, 2014 AT 8:39 AM

But for most, it is a good learning.

[Reply](#)**Pankaj says**

MAY 28, 2016 AT 11:07 AM

Design patterns are good to know, but don't use it just because you know it. See your requirement and if there is a benefit in using design pattern, then only apply it.

Simple code is the Best Code. ☐

[Reply](#)**Rama says**

AUGUST 4, 2014 AT 1:57 AM

Really good for refreshing the concepts. Thanks a lot ☐

[Reply](#)**Pankaj says**

MAY 28, 2016 AT 11:08 AM

Yeah I know, I also come by this once in a while to refresh my learning too. ☐

[Reply](#)**Marquis says**

JULY 14, 2014 AT 8:31 AM

2 words Thank you!

[Reply](#)

**Pankaj says**

MAY 28, 2016 AT 11:08 AM

2 words – “Appreciate it” ☐

[Reply](#)

**Cristian Manoliu says**

JUNE 24, 2014 AT 3:32 AM

Hello

I've been struggling with design patterns, but this document explains them so well.

Thank you for the lesson.

Best regards,

Cristian

[Reply](#)

**Pankaj says**

MAY 28, 2016 AT 11:09 AM

Hey Cristian, thanks for nice words. You can get my Design Pattern PDF eBook too by subscribing to my email newsletter.

[Reply](#)

**Hossein Moradi says**

JUNE 17, 2014 AT 11:52 PM

Thanks a lot for the instructive tutorial...

[Reply](#)

**Pankaj says**

MAY 28, 2016 AT 11:09 AM

You are welcome brother.

[Reply](#)

**jitendra says**

JUNE 17, 2014 AT 9:42 AM

This is excellent tutorial with best & simplified examples, compared to other tutorials on design pattern its easy to understand with great knowledge. The description given about each pattern helps you to decide which pattern should be used where. I read it like a story book.

[Reply](#)**Pankaj says**

MAY 28, 2016 AT 11:10 AM

Thanks Jitendra, you just made me smile. ☺

[Reply](#)**jerry says**

MAY 29, 2014 AT 9:04 PM

very useful. thanks a lot.

[Reply](#)**Pankaj says**

MAY 28, 2016 AT 11:11 AM

Thanks Jerry, I hope Tom is not giving you a lot of trouble these days. ☺

[Reply](#)**Pranay says**

APRIL 29, 2014 AT 6:21 AM

great work...

[Reply](#)**Pankaj says**

MAY 28, 2016 AT 11:11 AM

Thanks Pranay.

[Reply](#)

**Delli Babs says**

APRIL 25, 2014 AT 7:07 PM

This is excellent work sir !!! really helpful

[Reply](#)**Pankaj says**

MAY 28, 2016 AT 11:11 AM

You are welcome Delli.

[Reply](#)**chandrani says**

APRIL 20, 2014 AT 7:13 AM

Great work!

Thanks.

[Reply](#)**Pankaj says**

MAY 28, 2016 AT 11:11 AM

You are welcome Chandrani.

[Reply](#)**SRK says**

APRIL 9, 2014 AT 10:34 AM

Excellent site to know about all design patterns.

[Reply](#)**Pankaj says**

MAY 28, 2016 AT 11:12 AM

I tried to cover as much as possible, to the best of my knowledge. ☐

[Reply](#)

**job@basware.com says**

MARCH 12, 2014 AT 12:45 PM

This is a really good blog, and I always follow your blog whenever I need any clearance.

[Reply](#)

**Pankaj says**

MAY 28, 2016 AT 11:12 AM

Thanks for appreciation.

[Reply](#)

**Ravikumar says**

MARCH 5, 2014 AT 1:48 PM

Excellent Explanation, Wonderful examples to understood easily... Thanks

[Reply](#)

**Pankaj says**

MAY 28, 2016 AT 11:13 AM

You are welcome Ravikumar.

[Reply](#)

**Usha says**

MARCH 4, 2014 AT 6:35 AM

One of the best article on Design pattern, Thanks.

[Reply](#)

**Pankaj says**

MAY 28, 2016 AT 11:13 AM

Thanks Usha for kind words.

[Reply](#)



**Varun says**

JANUARY 22, 2014 AT 7:25 AM

This tutorial was very helpful ; thanks Pankaj ...waiting for J2EE Design Patterns tutorial

[Reply](#)

**Pankaj says**

MAY 28, 2016 AT 11:13 AM

I have added it on my TODO list, hope to start soon on them.

[Reply](#)

**Naveen says**

AUGUST 28, 2013 AT 4:17 AM

Hi your blog is really good, I request you to post more on struts2 from basics to advance.

[Reply](#)

**Pankaj says**

MAY 28, 2016 AT 11:14 AM

Struts2 tutorials are posted, i hope you find them useful.

[Reply](#)

**Syam says**

AUGUST 27, 2013 AT 1:16 AM

Hi Pankaj,

Thanks a lot for the tutorial.

Could you please also include J2EE Design Patterns also. (MVC, Business Delegates..)

[Reply](#)

**Pankaj says**

AUGUST 27, 2013 AT 7:17 PM

They are coming next, mostly in next month.

[Reply](#)

**Nestor says**

AUGUST 17, 2013 AT 4:35 AM

Hey your blog is really good. congratulation

[Reply](#)**Deepika Gurav says**

SEPTEMBER 22, 2015 AT 11:41 PM

how to print

1

0 1

1 0 1

0 1 0 1

1 0 1 0 1

[Reply](#)**jay says**

NOVEMBER 4, 2015 AT 9:09 PM

```
public class Pattern {  
    public static void main(String[] args) {  
        for(int i=1;i<6;i++){  
            for(int j=1;j<i;j++){  
                if((i+j)%2==0){  
                    System.out.print("0");  
                }  
                else  
                {  
                    System.out.print("1");  
                }  
            }  
            System.out.println();  
        }  
    }  
}
```

[Reply](#)**Pankaj says**

MAY 28, 2016 AT 11:14 AM

Thanks for the help buddy.

[Reply](#)

**parth says**

AUGUST 10, 2017 AT 10:26 PM

thanks

[Reply](#)**Ymg says**

MARCH 2, 2017 AT 9:56 AM

You can refer this code

class demo

```
{
public static void main(String args[])
{
int i,j,zero=0,one=1;
for(i=1;i<=j;)
{
if(j==1 || j==3 || j==5)
{
System.out.print(one+ "\t");
}
else if(j==2 || j==4)
{
System.out.print(zero+ "\t");
}
}
System.out.print("\n");
}
}
```

[Reply](#)

## Leave a Reply

Your email address will not be published. Required fields are marked \*

Comment

Name \*

Email \*

☐

Save my name, email, and website in this browser for the next time I comment.

---

[DOWNLOAD ANDROID APP](#)

---



---

[DESIGN PATTERNS TUTORIAL](#)

---

## Java Design Patterns

### Creational Design Patterns

- > [Singleton](#)
- > [Factory](#)
- > [Abstract Factory](#)
- > [Builder](#)
- > [Prototype](#)

### Structural Design Patterns

- > [Adapter](#)

- > [Composite](#)
- > [Proxy](#)
- > [Flyweight](#)
- > [Facade](#)
- > [Bridge](#)
- > [Decorator](#)

## Behavioral Design Patterns

- > [Template Method](#)
- > [Mediator](#)
- > [Chain of Responsibility](#)
- > [Observer](#)
- > [Strategy](#)
- > [Command](#)
- > [State](#)
- > [Visitor](#)
- > [Interpreter](#)
- > [Iterator](#)
- > [Memento](#)

## Miscellaneous Design Patterns

- > [Dependency Injection](#)
- > [Thread Safety in Java Singleton](#)

RECOMMENDED TUTORIALS

### Java Tutorials

- > [Java IO](#)
- > [Java Regular Expressions](#)
- > [Multithreading in Java](#)
- > [Java Logging](#)
- > [Java Annotations](#)
- > [Java XML](#)
- > [Collections in Java](#)
- > [Java Generics](#)
- > [Exception Handling in Java](#)
- > [Java Reflection](#)
- > [Java Design Patterns](#)
- > [JDBC Tutorial](#)

### Java EE Tutorials

- > [Servlet JSP Tutorial](#)
- > [Struts2 Tutorial](#)
- > [Spring Tutorial](#)
- > [Hibernate Tutorial](#)
- > [Primefaces Tutorial](#)
- > [Apache Axis 2](#)
- > [JAX-RS](#)
- > [Memcached Tutorial](#)

**Create Your Own Website**

---

**Build Business Web Site**

---

**Learn Programming Online**

---

**Interior Design Software**

---

**Python Programming Tutorial**

---

**Java Design Patterns**

---

**Java Help Forum**

---

**Java for Beginners**

