

Java Regex

The **Java Regex** or Regular Expression is an API to *define a pattern for searching or manipulating strings*.

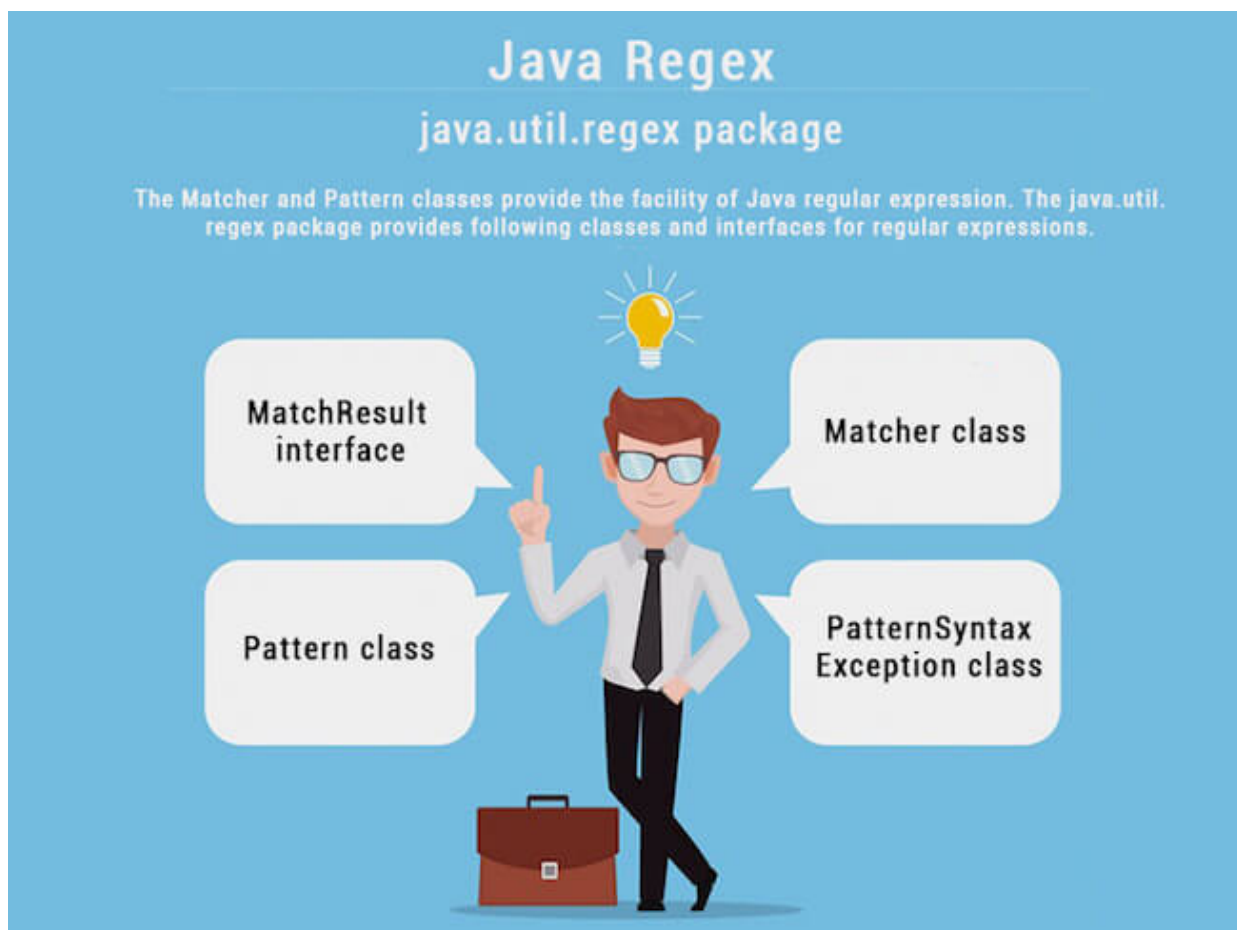
It is widely used to define the constraint on strings such as password and email validation. After learning Java regex tutorial, you will be able to test your regular expressions by the Java Regex Tester Tool.

Java Regex API provides 1 interface and 3 classes in **java.util.regex** package.

java.util.regex package

The Matcher and Pattern classes provide the facility of Java regular expression. The java.util.regex package provides following classes and interfaces for regular expressions.

1. MatchResult interface
2. Matcher class
3. Pattern class
4. PatternSyntaxException class



Matcher class

It implements the **MatchResult** interface. It is a *regex engine* which is used to perform match operations on a character sequence.

No.	Method	Description
1	boolean matches()	test whether the regular expression matches the pattern.
2	boolean find()	finds the next expression that matches the pattern.
3	boolean find(int start)	finds the next expression that matches the pattern from the given start number.
4	String group()	returns the matched subsequence.
5	int start()	returns the starting index of the matched subsequence.
6	int end()	returns the ending index of the matched subsequence.
7	int groupCount()	returns the total number of the matched subsequence.

Pattern class

It is the *compiled version of a regular expression*. It is used to define a pattern for the regex engine.

No.	Method	Description
1	static Pattern compile(String regex)	compiles the given regex and returns the instance of the Pattern.
2	Matcher matcher(CharSequence input)	creates a matcher that matches the given input with the pattern.
3	static boolean matches(String regex, CharSequence input)	It works as the combination of compile and matcher methods. It compiles the regular expression and matches the given input with the pattern.
4	String[] split(CharSequence input)	splits the given input string around matches of given pattern.
5	String pattern()	returns the regex pattern.

Example of Java Regular Expressions

There are three ways to write the regex example in Java.

```
import java.util.regex.*;
public class RegexExample1{
    public static void main(String args[]){
        //1st way
        Pattern p = Pattern.compile(".s");//. represents single character
        Matcher m = p.matcher("as");
        boolean b = m.matches();

        //2nd way
        boolean b2=Pattern.compile(".s").matcher("as").matches();

        //3rd way
        boolean b3 = Pattern.matches(".s", "as");

        System.out.println(b+ " "+b2+ " "+b3);
    }
}
```

Test it Now

Output

```
true true true
```

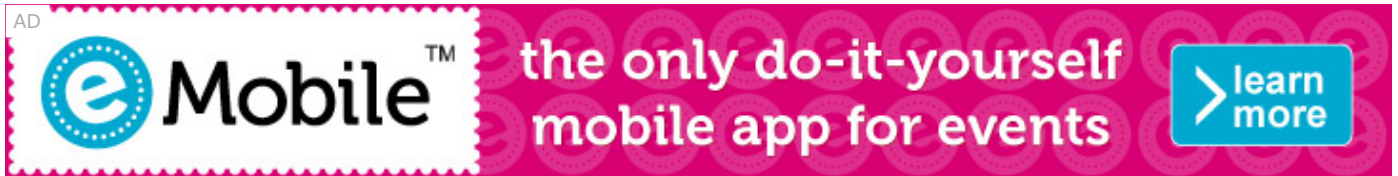
Regular Expression . Example

The . (dot) represents a single character.

```
import java.util.regex.*;
class RegexExample2{
    public static void main(String args[]){
        System.out.println(Pattern.matches(".s", "as"));//true (2nd char is s)
        System.out.println(Pattern.matches(".s", "mk"));//false (2nd char is not s)
        System.out.println(Pattern.matches(".s", "mst"));//false (has more than 2 char)
    }
}
```

```
System.out.println(Pattern.matches(".s", "amms")); //false (has more than 2 char)
System.out.println(Pattern.matches("..s", "mas")); //true (3rd char is s)
}}
```

Test it Now



Regex Character classes

No.	Character Class	Description
1	[abc]	a, b, or c (simple class)
2	[^abc]	Any character except a, b, or c (negation)
3	[a-zA-Z]	a through z or A through Z, inclusive (range)
4	[a-d[m-p]]	a through d, or m through p: [a-dm-p] (union)
5	[a-z&&[def]]	d, e, or f (intersection)
6	[a-z&&[^bc]]	a through z, except for b and c: [ad-z] (subtraction)
7	[a-z&&[^m-p]]	a through z, and not m through p: [a-lq-z](subtraction)

Regular Expression Character classes Example

```
import java.util.regex.*;
class RegexExample3{
public static void main(String args[]){
System.out.println(Pattern.matches("[amn]", "abcd")); //false (not a or m or n)
System.out.println(Pattern.matches("[amn]", "a")); //true (among a or m or n)
System.out.println(Pattern.matches("[amn]", "ammmna")); //false (m and a comes more than once)
}}
```

Test it Now

Regex Quantifiers

The quantifiers specify the number of occurrences of a character.

Regex	Description
X?	X occurs once or not at all
X+	X occurs once or more times
X*	X occurs zero or more times
X{n}	X occurs n times only
X{n,}	X occurs n or more times
X{y,z}	X occurs at least y times but less than z times

Regular Expression Character classes and Quantifiers Example

```
import java.util.regex.*;
class RegexExample4{
public static void main(String args[]){
System.out.println("? quantifier ....");
System.out.println(Pattern.matches("[amn]?", "a")); //true (a or m or n comes one time)
System.out.println(Pattern.matches("[amn]?", "aaa")); //false (a comes more than one time)
System.out.println(Pattern.matches("[amn]?", "aammmnn")); //false (a m and n comes more than one time)
System.out.println(Pattern.matches("[amn]?", "aazzta")); //false (a comes more than one time)
System.out.println(Pattern.matches("[amn]?", "am")); //false (a or m or n must come one time)

System.out.println("+ quantifier ....");
System.out.println(Pattern.matches("[amn]+", "a")); //true (a or m or n once or more times)
System.out.println(Pattern.matches("[amn]+", "aaa")); //true (a comes more than one time)
System.out.println(Pattern.matches("[amn]+", "aammmnn")); //true (a or m or n comes more than once)
System.out.println(Pattern.matches("[amn]+", "aazzta")); //false (z and t are not matching pattern)

System.out.println("* quantifier ....");
```

```
System.out.println(Pattern.matches("[amn]*", "ammmna")); //true (a or m or n may come zero or more times)

}}
```

Test it Now

Regex Metacharacters

The regular expression metacharacters work as shortcodes.

Regex	Description
.	Any character (may or may not match terminator)
\d	Any digits, short of [0-9]
\D	Any non-digit, short for [^0-9]
\s	Any whitespace character, short for [\t\n\x0B\f\r]
\S	Any non-whitespace character, short for [^\s]
\w	Any word character, short for [a-zA-Z_0-9]
\W	Any non-word character, short for [^\w]
\b	A word boundary
\B	A non word boundary

Regular Expression Metacharacters Example

```
import java.util.regex.*;
class RegexExample5{
public static void main(String args[]){
System.out.println("metacharacters d...."); \\d means digit

System.out.println(Pattern.matches("\\d", "abc")); //false (non-digit)
System.out.println(Pattern.matches("\\d", "1")); //true (digit and comes once)
System.out.println(Pattern.matches("\\d", "4443")); //false (digit but comes more than once)
```

```

System.out.println(Pattern.matches("\\d", "323abc")); //false (digit and char)

System.out.println("metacharacters D..."); \\D means non-digit

System.out.println(Pattern.matches("\\D", "abc")); //false (non-digit but comes more than once)
System.out.println(Pattern.matches("\\D", "1")); //false (digit)
System.out.println(Pattern.matches("\\D", "4443")); //false (digit)
System.out.println(Pattern.matches("\\D", "323abc")); //false (digit and char)
System.out.println(Pattern.matches("\\D", "m")); //true (non-digit and comes once)

System.out.println("metacharacters D with quantifier....");
System.out.println(Pattern.matches("\\D*", "mak")); //true (non-digit and may come 0 or more times)

}}

```

Test it Now

Regular Expression Question 1

```

/*Create a regular expression that accepts alphanumeric characters only.
Its length must be six characters long only.*/

import java.util.regex.*;
class RegexExample6{
public static void main(String args[]){
System.out.println(Pattern.matches("[a-zA-Z0-9]{6}", "arun32")); //true
System.out.println(Pattern.matches("[a-zA-Z0-9]{6}", "kkvarun32")); //false (more than 6 char)
System.out.println(Pattern.matches("[a-zA-Z0-9]{6}", "JA2Uk2")); //true
System.out.println(Pattern.matches("[a-zA-Z0-9]{6}", "arun$2")); //false ($ is not matched)
}}

```

Test it Now

Regular Expression Question 2

```

/*Create a regular expression that accepts 10 digit numeric characters

```

starting with 7, 8 or 9 only.*/

```
import java.util.regex.*;
class RegexExample7{
public static void main(String args[]){
System.out.println("by character classes and quantifiers ...");
System.out.println(Pattern.matches("[789]{1}[0-9]{9}", "9953038949")); //true
System.out.println(Pattern.matches("[789][0-9]{9}", "9953038949")); //true

System.out.println(Pattern.matches("[789][0-9]{9}", "99530389490")); //false (11 characters)
System.out.println(Pattern.matches("[789][0-9]{9}", "6953038949")); //false (starts from 6)
System.out.println(Pattern.matches("[789][0-9]{9}", "8853038949")); //true

System.out.println("by metacharacters ...");
System.out.println(Pattern.matches("[789]{1}\\d{9}", "8853038949")); //true
System.out.println(Pattern.matches("[789]{1}\\d{9}", "3853038949")); //false (starts from 3)

}}
```

Test it Now

Java Regex Finder Example

```
import java.util.regex.Pattern;
import java.util.Scanner;
import java.util.regex.Matcher;
public class RegexExample8{
    public static void main(String[] args){
        Scanner sc=new Scanner(System.in);
        while (true) {
            System.out.println("Enter regex pattern:");
            Pattern pattern = Pattern.compile(sc.nextLine());
            System.out.println("Enter text:");
            Matcher matcher = pattern.matcher(sc.nextLine());
            boolean found = false;
            while (matcher.find()) {
                System.out.println("I found the text " + matcher.group() + " starting at index " +
```



```
        matcher.start()+" and ending at index "+matcher.end());  
        found = true;  
    }  
    if(!found){  
        System.out.println("No match found.");  
    }  
}  
}  
}
```

Output:

```
Enter regex pattern: java  
Enter text: this is java, do you know java  
I found the text java starting at index 8 and ending at index 12  
I found the text java starting at index 26 and ending at index 30
```

[< Prev](#)[Next >](#)

AD

 **For Videos Join Our Youtube Channel: [Join Now](#)**

Feedback

- Send your Feedback to feedback@javatpoint.com

RMI (Remote Method Invocation)

The **RMI** (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM.

The RMI provides remote communication between the applications using two objects *stub* and *skeleton*.

Understanding stub and skeleton

RMI uses stub and skeleton object for communication with the remote object.

A **remote object** is an object whose method can be invoked from another JVM. Let's understand the stub and skeleton objects:

stub

The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

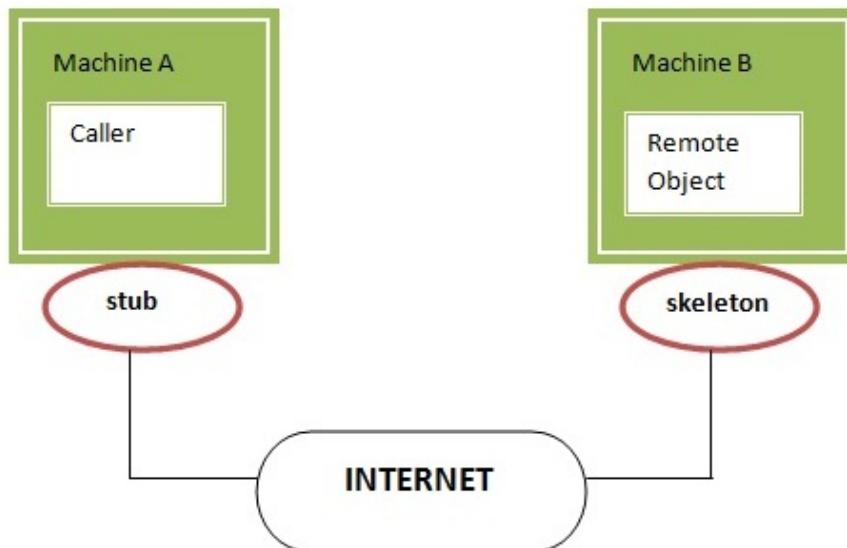
1. It initiates a connection with remote Virtual Machine (JVM),
2. It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),
3. It waits for the result
4. It reads (unmarshals) the return value or exception, and
5. It finally, returns the value to the caller.

skeleton

The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

1. It reads the parameter for the remote method
2. It invokes the method on the actual remote object, and
3. It writes and transmits (marshals) the result to the caller.

In the Java 2 SDK, an stub protocol was introduced that eliminates the need for skeletons.



Understanding requirements for the distributed applications

If any application performs these tasks, it can be distributed application.

.

1. The application need to locate the remote method
2. It need to provide the communication with the remote objects, and
3. The application need to load the class definitions for the objects.

The RMI application have all these features, so it is called the distributed application.

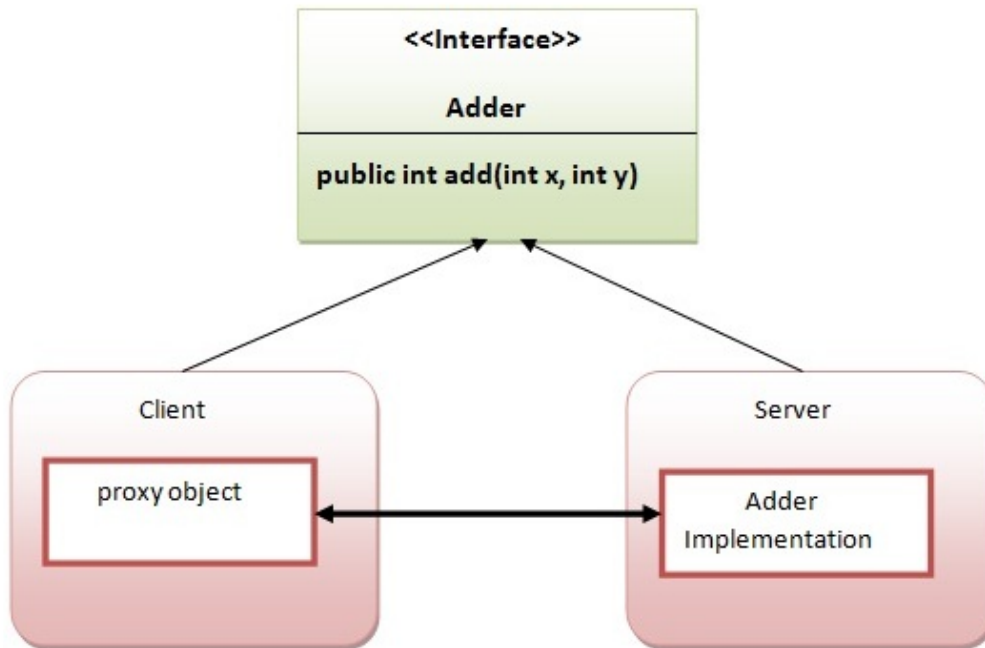
Java RMI Example

The is given the 6 steps to write the RMI program.

1. Create the remote interface
2. Provide the implementation of the remote interface
3. Compile the implementation class and create the stub and skeleton objects using the rmic tool
4. Start the registry service by rmiregistry tool
5. Create and start the remote application
6. Create and start the client application

RMI Example

In this example, we have followed all the 6 steps to create and run the rmi application. The client application need only two files, remote interface and client application. In the rmi application, both client and server interacts with the remote interface. The client application invokes methods on the proxy object, RMI sends the request to the remote JVM. The return value is sent back to the proxy object and then to the client application.



1) create the remote interface

For creating the remote interface, extend the Remote interface and declare the RemoteException with all the methods of the remote interface. Here, we are creating a remote interface that extends the Remote interface. There is only one method named add() and it declares RemoteException.

```
import java.rmi.*;
public interface Adder extends Remote{
    public int add(int x,int y)throws RemoteException;
}
```

2) Provide the implementation of the remote interface

Now provide the implementation of the remote interface. For providing the implementation of the Remote interface, we need to

- Either extend the UnicastRemoteObject class,

- or use the `exportObject()` method of the `UnicastRemoteObject` class

In case, you extend the `UnicastRemoteObject` class, you must define a constructor that declares `RemoteException`.

```
import java.rmi.*;
import java.rmi.server.*;
public class AdderRemote extends UnicastRemoteObject implements Adder{
    AdderRemote()throws RemoteException{
        super();
    }
    public int add(int x,int y){return x+y;}
}
```

3) create the stub and skeleton objects using the `rmic` tool.

Next step is to create stub and skeleton objects using the `rmi` compiler. The `rmic` tool invokes the RMI compiler and creates stub and skeleton objects.

```
rmic AdderRemote
```

4) Start the registry service by the `rmiregistry` tool

Now start the registry service by using the `rmiregistry` tool. If you don't specify the port number, it uses a default port number. In this example, we are using the port number 5000.

```
rmiregistry 5000
```

5) Create and run the server application

Now `rmi` services need to be hosted in a server process. The `Naming` class provides methods to get and store the remote object. The `Naming` class provides 5 methods.

<pre>public static java.rmi.Remote lookup(java.lang.String) throws java.rmi.NotBoundException, java.net.MalformedURLException, java.rmi.RemoteException;</pre>	It returns the reference of the remote object.
--	--

<pre>public static void bind(java.lang.String, java.rmi.Remote) throws java.rmi.AlreadyBoundException, java.net.MalformedURLException, java.rmi.RemoteException;</pre>	It binds the remote object with the given name.
<pre>public static void unbind(java.lang.String) throws java.rmi.RemoteException, java.rmi.NotBoundException, java.net.MalformedURLException;</pre>	It destroys the remote object which is bound with the given name.
<pre>public static void rebind(java.lang.String, java.rmi.Remote) throws java.rmi.RemoteException, java.net.MalformedURLException;</pre>	It binds the remote object to the new name.
<pre>public static java.lang.String[] list(java.lang.String) throws java.rmi.RemoteException, java.net.MalformedURLException;</pre>	It returns an array of the names of the remote objects bound in the registry.

In this example, we are binding the remote object by the name sonoo.

```
import java.rmi.*;
import java.rmi.registry.*;
public class MyServer{
public static void main(String args[]){
try{
Adder stub=new AdderRemote();
Naming.rebind("rmi://localhost:5000/sonoo",stub);
}catch(Exception e){System.out.println(e);}
}
}
```

6) Create and run the client application

At the client we are getting the stub object by the lookup() method of the Naming class and invoking the method on this object. In this example, we are running the server and client applications, in the same machine so we are using localhost. If you want to access the remote object from another machine, change the localhost to the host name (or IP address) where the remote object is located.



```
import java.rmi.*;
public class MyClient{
public static void main(String args[]){
try{
Adder stub=(Adder)Naming.lookup("rmi://localhost:5000/sonoo");
System.out.println(stub.add(34,4));
}catch(Exception e){}
}
}
```

download this example of rmi

For running **this** rmi example,

1) compile all the java files

```
javac *.java
```

2)create stub and skeleton object by rmic tool

```
rmic AdderRemote
```

3)start rmi registry in one command prompt

```
rmiregistry 5000
```

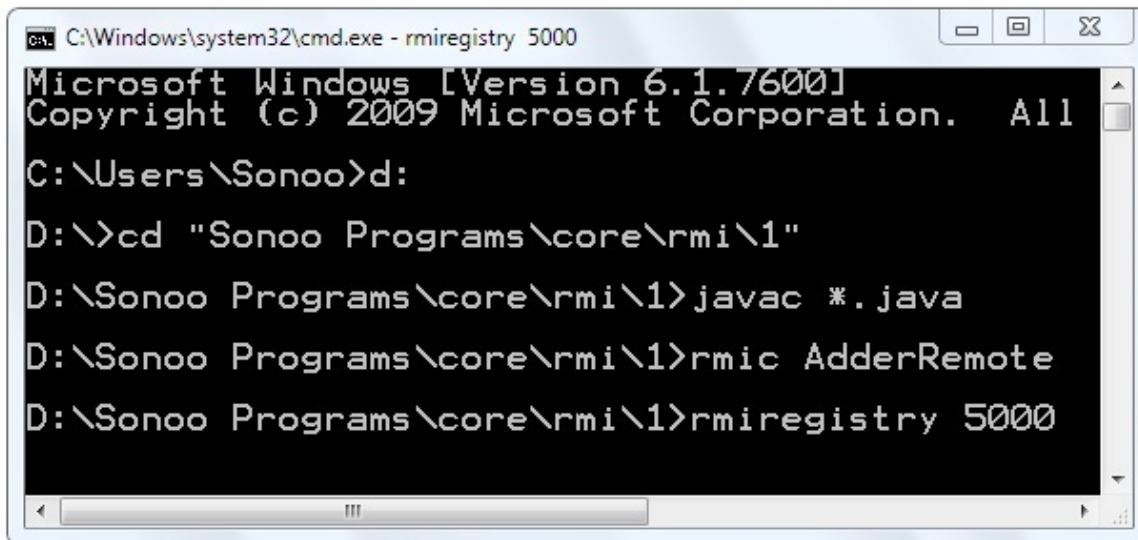
4)start the server in another command prompt

```
java MyServer
```

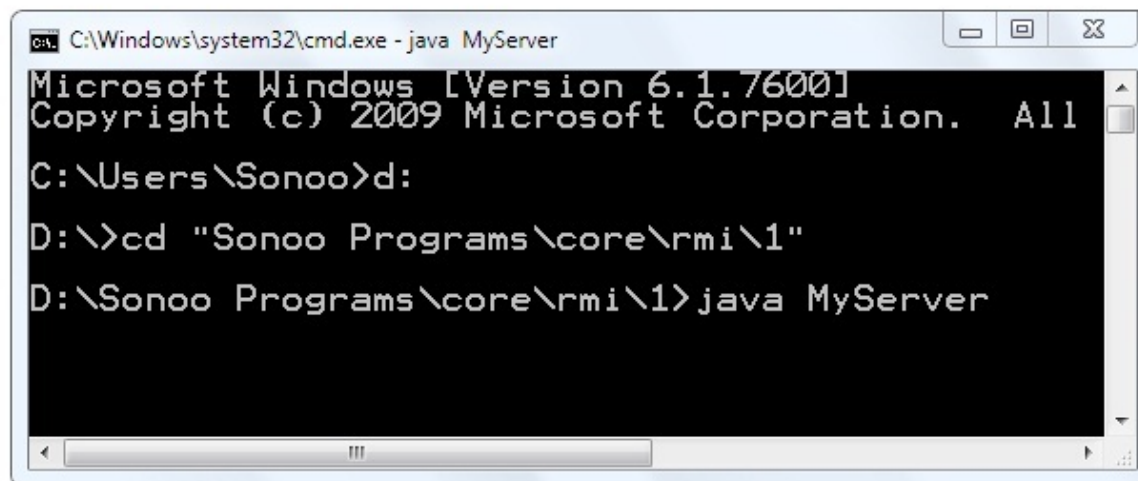
5)start the client application in another command prompt

```
java MyClient
```

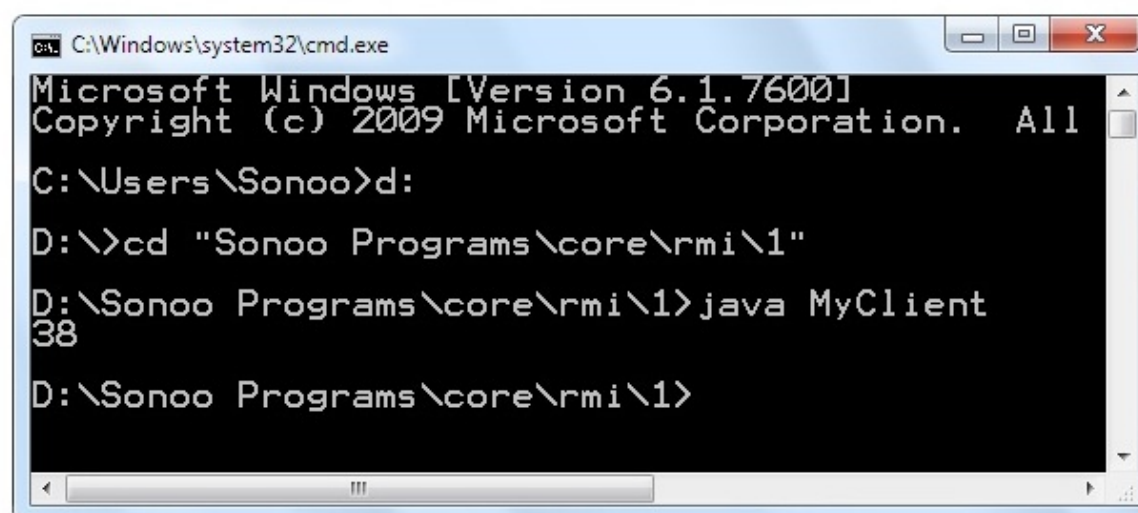
Output of this RMI example



```
C:\Windows\system32\cmd.exe - rmiregistry 5000
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All
C:\Users\Sonoo>d:
D:\>cd "Sonoo Programs\core\rmi\1"
D:\Sonoo Programs\core\rmi\1>javac *.java
D:\Sonoo Programs\core\rmi\1>rmic AdderRemote
D:\Sonoo Programs\core\rmi\1>rmiregistry 5000
```



```
C:\Windows\system32\cmd.exe - java MyServer
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All
C:\Users\Sonoo>d:
D:\>cd "Sonoo Programs\core\rmi\1"
D:\Sonoo Programs\core\rmi\1>java MyServer
```



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All
C:\Users\Sonoo>d:
D:\>cd "Sonoo Programs\core\rmi\1"
D:\Sonoo Programs\core\rmi\1>java MyClient
38
D:\Sonoo Programs\core\rmi\1>
```


Meaningful example of RMI application with database

Consider a scenario, there are two applications running in different machines. Let's say MachineA and MachineB, machineA is located in United States and MachineB in India. MachineB want to get list of all the customers of MachineA application.

Let's develop the RMI application by following the steps.




1) Create the table

First of all, we need to create the table in the database. Here, we are using Oracle10 database.

CUSTOMER400

TableDataIndexesModelConstraintsGrantsStatisticsUI DefaultsTriggersDependenciesSQL

QueryCount RowsInsert Row

EDIT	ACC_NO	FIRSTNAME	LASTNAME	EMAIL	AMOUNT
	67539876	James	Franklin	franklin1james@gmail.com	500000
	67534876	Ravi	Kumar	ravimalik@gmail.com	98000
	67579872	Vimal	Jaiswal	jaiswalvimal32@gmail.com	9380000
row(s) 1 - 3 of 3					

Download

2) Create Customer class and Remote interface

File: Customer.java

```
package com.javatpoint;

public class Customer implements java.io.Serializable{

    private int acc_no;

    private String firstname,lastname,email;

    private float amount;

    //getters and setters

}
```

Note: Customer class must be Serializable.

File: Bank.java

```
package com.javatpoint;

import java.rmi.*;
import java.util.*;

interface Bank extends Remote{
    public List<Customer> getCustomers()throws RemoteException;
}
```

3) Create the class that provides the implementation of Remote interface

File: BankImpl.java

```
package com.javatpoint;

import java.rmi.*;
import java.rmi.server.*;
import java.sql.*;
import java.util.*;

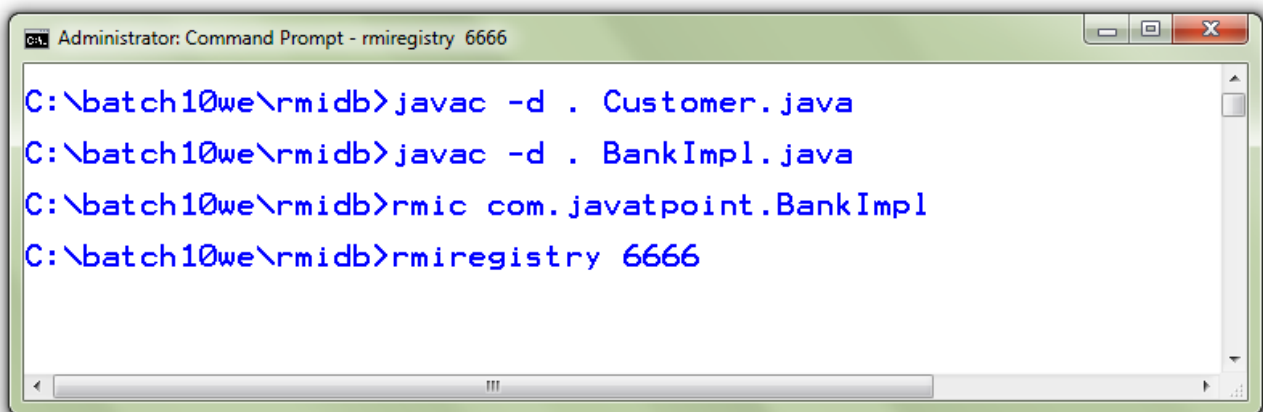
class BankImpl extends UnicastRemoteObject implements Bank{
    BankImpl()throws RemoteException{}

    public List<Customer> getCustomers(){
        List<Customer> list=new ArrayList<Customer>();
        try{
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","ora
            PreparedStatement ps=con.prepareStatement("select * from customer400");
            ResultSet rs=ps.executeQuery();

            while(rs.next()){
                Customer c=new Customer();
                c.setAcc_no(rs.getInt(1));
                c.setFirstname(rs.getString(2));
                c.setLastname(rs.getString(3));
                c.setEmail(rs.getString(4));
                c.setAmount(rs.getFloat(5));
                list.add(c);
            }
        }
    }
}
```

```
con.close();  
}catch(Exception e){System.out.println(e);}  
return list;  
}//end of getCustomers()  
}
```

4) Compile the class rmic tool and start the registry service by rmiregistry tool

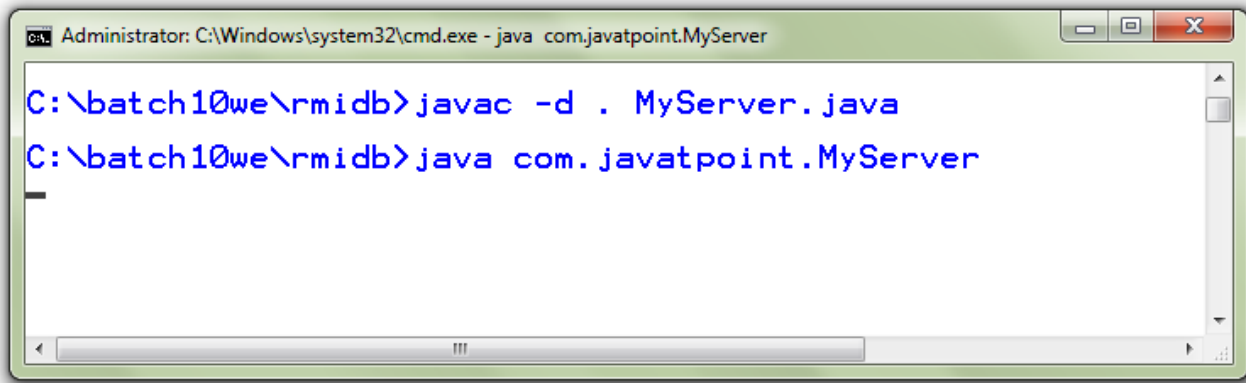


```
Administrator: Command Prompt - rmiregistry 6666  
C:\batch10we\rmidb>javac -d . Customer.java  
C:\batch10we\rmidb>javac -d . BankImpl.java  
C:\batch10we\rmidb>rmic com.javatpoint.BankImpl  
C:\batch10we\rmidb>rmiregistry 6666
```

5) Create and run the Server

File: MyServer.java

```
package com.javatpoint;  
import java.rmi.*;  
public class MyServer{  
  public static void main(String args[])throws Exception{  
    Remote r=new BankImpl();  
    Naming.rebind("rmi://localhost:6666/javatpoint",r);  
  }  
}
```



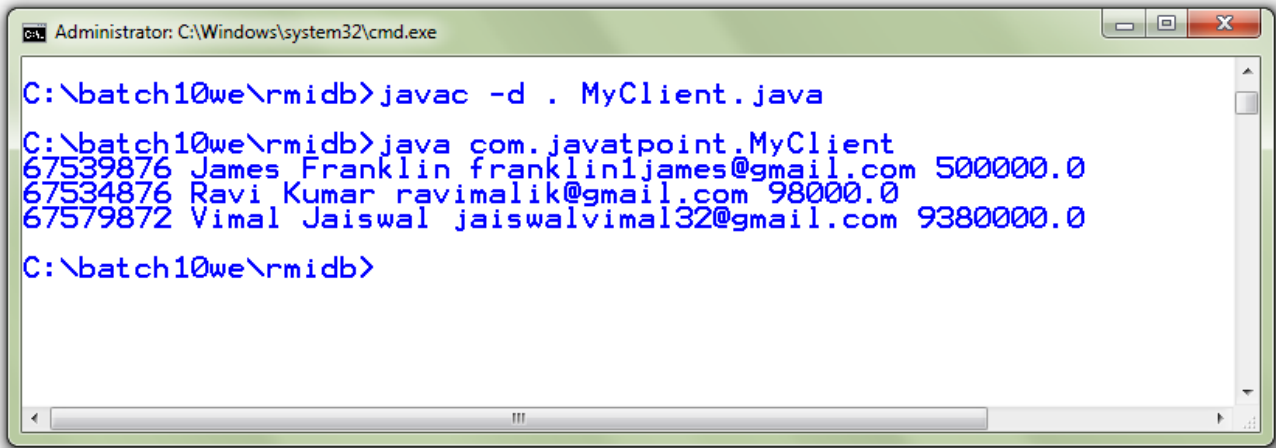
6) Create and run the Client

File: MyClient.java

```
package com.javatpoint;
import java.util.*;
import java.rmi.*;
public class MyClient{
    public static void main(String args[])throws Exception{
        Bank b=(Bank)Naming.lookup("rmi://localhost:6666/javatpoint");

        List<Customer> list=b.getCustomers();
        for(Customer c:list){
            System.out.println(c.getAcc_no()+" "+c.getFirstname()+" "+c.getLastname()
            +" "+c.getEmail()+" "+c.getAmount());
        }

    }
}
```



```
Administrator: C:\Windows\system32\cmd.exe

C:\batch10we\rmidb>javac -d . MyClient.java

C:\batch10we\rmidb>java com.javatpoint.MyClient
67539876 James Franklin franklin1james@gmail.com 500000.0
67534876 Ravi Kumar ravimalik@gmail.com 98000.0
67579872 Vimal Jaiswal jaiswalvimal32@gmail.com 9380000.0

C:\batch10we\rmidb>
```

download this example of rmi with database

← Prev

Next →

AD

 Youtube For Videos Join Our Youtube Channel: [Join Now](#)

Feedback

- Send your Feedback to feedback@javatpoint.com