# Java Reflection API

**Java Reflection** is a *process of examining or modifying the run time behavior of a class at run time.*

The **java.lang.Class** class provides many methods that can be used to get metadata, examine and change the run time behavior of a class.

The java.lang and java.lang.reflect packages provide classes for java reflection.

### Where it is used

The Reflection API is mainly used in:

- IDE (Integrated Development Environment) e.g., Eclipse, MyEclipse, NetBeans etc.
- Debugger
- Test Tools etc.

### *Do You Know?*

- How many ways can we get the instance of Class class?
- How to create the javap tool?
- How to create the appletviewer tool?
- How to access the private method from outside the class?

## java.lang.Class class

The java.lang.Class class performs mainly two tasks:

- provides methods to get the metadata of a class at run time.
- provides methods to examine and change the run time behavior of a class.

## Commonly used methods of Class class:

| Method | Description |
|---|---|

| | |
|---|---|
| 1) public String getName() | returns the class name |
| 2) public static Class forName(String className)throws ClassNotFoundException | loads the class and returns the reference of Class class. |
| 3) public Object newInstance()throws InstantiationException,IllegalAccessException | creates new instance. |
| 4) public boolean isInterface() | checks if it is interface. |
| 5) public boolean isArray() | checks if it is array. |
| 6) public boolean isPrimitive() | checks if it is primitive. |
| 7) public Class getSuperclass() | returns the superclass class reference. |
| 8) public Field[] getDeclaredFields()throws SecurityException | returns the total number of fields of this class. |
| 9) public Method[] getDeclaredMethods()throws SecurityException | returns the total number of methods of this class. |
| 10) public Constructor[] getDeclaredConstructors()throws SecurityException | returns the total number of constructors of this class. |
| 11) public Method getDeclaredMethod(String name,Class[] parameterTypes)throws NoSuchMethodException,SecurityException | returns the method class instance. |

## How to get the object of Class class?

There are 3 ways to get the instance of Class class. They are as follows:

- forName() method of Class class
- getClass() method of Object class
- the .class syntax

### 1) forName() method of Class class

- is used to load the class dynamically.
- returns the instance of Class class.

- It should be used if you know the fully qualified name of class.This cannot be used for primitive types.

Let's see the simple example of forName() method.

**FileName:** Test.java

```
class Simple{}

public class Test{
 public static void main(String args[]) throws Exception {
  Class c=Class.forName("Simple");
  System.out.println(c.getName());
 }
}
```

**Output:**

```
Simple
```

## 2) getClass() method of Object class

It returns the instance of Class class. It should be used if you know the type. Moreover, it can be used with primitives.

**FileName:** Test.java

```
class Simple{}

class Test{
  void printName(Object obj){
  Class c=obj.getClass();
  System.out.println(c.getName());
  }
  public static void main(String args[]){
   Simple s=new Simple();

   Test t=new Test();
   t.printName(s);
 }
}
```

**Output:**

```
Simple
```

### 3) The .class syntax

If a type is available, but there is no instance, then it is possible to obtain a Class by appending ".class" to the name of the type. It can be used for primitive data types also.

**FileName:** Test.java

```
class Test{
  public static void main(String args[]){
   Class c = boolean.class;
   System.out.println(c.getName());

   Class c2 = Test.class;
   System.out.println(c2.getName());
 }
}
```

**Output:**

```
        boolean
        Test
```

## Determining the class object

The following methods of Class class are used to determine the class object:

**1) public boolean isInterface():** determines if the specified Class object represents an interface type.

**2) public boolean isArray():** determines if this Class object represents an array class.

**3) public boolean isPrimitive():** determines if the specified Class object represents a primitive type.

Let's see the simple example of reflection API to determine the object type.

**FileName:** Test.java

```
class Simple{}
interface My{}

class Test{
 public static void main(String args[]){
  try{
   Class c=Class.forName("Simple");
   System.out.println(c.isInterface());

   Class c2=Class.forName("My");
   System.out.println(c2.isInterface());

  }catch(Exception e){System.out.println(e);}

 }
}
```

**Output:**

```
false
true
```

## Pros and Cons of Reflection

Java reflection should always be used with caution. While the reflection provides a lot of advantages, it has some disadvantages too. Let's discuss the advantages first.

**Pros:** Inspection of interfaces, classes, methods, and fields during runtime is possible using reflection, even without using their names during the compile time. It is also possible to call methods, instantiate a clear or to set the value of fields using reflection. It helps in the creation of Visual Development Environments and class browsers which provides aid to the developers to write the correct code.

**Cons:** Using reflection, one can break the principles of encapsulation. It is possible to access the private methods and fields of a class using reflection. Thus, reflection may leak important data to the outside world, which is dangerous. For example, if one access the private members of a class and sets null value to it, then the other user of the same class can get the NullReferenceException, and this behaviour is not expected.

Another demerit is the overhead in performance. Since the types in reflection are resolved dynamically, JVM (Java Virtual Machine) optimization cannot take place. Therefore, the operations performed by reflections are usually slow.

## Conclusion

Because of the above-mentioned cons, it is generally advisable to avoid using reflection. It is an advanced feature that should only be used by programmers or developers who have a good knowledge of the basics of the language. Always remember! Whenever reflection is used, the security of the application is compromised.

## Next Topics of Reflection API Tutorial

newInstance() method

Understanding javap tool

creating javap tool

creating appletviewer tool

Call private method from another class

← Prev Next →

## newInstance() method

The **newInstance()** method of **Class** class and **Constructor** class is used to create a new instance of the class.

The newInstance() method of Class class can invoke zero-argument constructor, whereas newInstance() method of Constructor class can invoke any number of arguments. So, Constructor class is preferred over Class class.

### Syntax of newInstance() method of Class class

**public T newInstance()throws InstantiationException,IllegalAccessException**

Here T is the generic version. You can think of it as Object class. You will learn about generics later.

### newInstance() Method Example-1

Let's see the simple example to use newInstance() method.

**FileName:** Test.java

```
class Simple{
 void message(){System.out.println("Hello Java");}
}

class Test{
 public static void main(String args[]){
  try{
  Class c=Class.forName("Simple");
  Simple s=(Simple)c.newInstance();
  s.message();

  }catch(Exception e){System.out.println(e);}

 }
}
```

**Output:**

```
Hello java
```

### newInstance() method Example-2

We have learned in the introductory part of this topic that the newInstance() method of the Class class can only invoke the parameterless constructor. Let's understand the same with the help of an example.

**FileName:** ReflectionExample1.java

```
// important import statements
import static java.lang.System.out;
import java.lang.reflect.*;
import javax.swing.*;

public class ReflectionExample1
{
// Allowing Java Virtual Machine to handle the ClassNotFoundException
// main method
public static void main(String argvs[]) throws ClassNotFoundException
{

Object ob = null;
Class classDefinition = Class.forName("javax.swing.JLabel");
ob = classDefinition.newInstance();

// instance variable for holding the instance of the class
JLabel l1;

// checking whether the created object ob is
// the instance of JLabel or not.
// If yes, do the typecasting; otherwise, terminate the method
if(ob instanceof JLabel)
{
l1 = (JLabel)ob;
}
else
{
return;
}

// reaching here means the typecasting has been done
// now we can invoke the getText() method
out.println(l1.getText());



}
}
```

**Output:**

```
/ReflectionExample1.java:15: error: unreported exception InstantiationException; must be caught or declared to be throw
ob = classDefinition.newInstance();
                                ^
Note: /ReflectionExample1.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
1 error
```

**Explanation:** The newInstance() method of the Class class only invokes the zero-argument constructor. However, we need to invoke the non-zero argument constructor for that, we need to use the newInstance() method of the class Constructor.

The following program shows how one can use the newInstance() method of the class Constructor to avoid the exception that has come in the above example.

**FileName:** ReflectionExample2.java

```
// important import statements
import static java.lang.System.out;
import java.lang.reflect.*;
import javax.swing.*;

public class ReflectionExample2
{
```

```
// main method
public static void main(String argvs[])
{
try
{
Class[] t = { String.class };
Class classDef = Class.forName("javax.swing.JLabel");

// getting the constructor
Constructor cons = classDef.getConstructor(t);

// setting the label
Object[] objct = { "My JLabel in Reflection."};

// getting the instance by invoking the correct constructor this time
Object ob = cons.newInstance(objct);



// instance variable for holding the instance of the class
JLabel l1;

// checking whether the created object ob is
// the instance of JLabel or not.
// If yes, do the typecasting; otherwise, exit from the method
if(ob instanceof JLabel)
{
l1 = (JLabel)ob;
}
else
{
// exiting from the method using the return statement
return;
}

// if the control reaches here, then it means the typecasting has been done
// now we can print the label of the JLabel instance
out.println(l1.getText());
}
// relevant catch block for handling the raised exceptions.
catch (InstantiationException ie)
{
out.println(ie);
}
catch (IllegalAccessException ie)
{
System.out.println(ie);
}

catch (InvocationTargetException ie)
{
out.println(ie);
}
catch (ClassNotFoundException e)
{
```

```
out.println(e);
}

catch (NoSuchMethodException e)
{
out.println(e);
}
}
}
```

**Output:**

```
My JLabel in Reflection.
```

<<Prev Next>>

## Feedback

- Send your Feedback to feedback@javatpoint.com

## Help Others, Please Share

## Learn Latest Tutorials

| | | | | | |
|---|---|---|---|---|---|
| Splunk tutorial<br>Splunk | SPSS tutorial<br>SPSS | Swagger tutorial<br>Swagger | T-SQL tutorial<br>Transact-SQL | Tumblr tutorial<br>Tumblr | React tutorial<br>ReactJS |
| Regex tutorial<br>Regex | Reinforcement learning tutorial<br>Reinforcement Learning | R Programming tutorial<br>R Programming | RxJS tutorial<br>RxJS | React Native tutorial<br>React Native | Python Design Patterns<br>Python Design Patterns |
| Python Pillow tutorial<br>Python Pillow | Python Turtle tutorial<br>Python Turtle | Keras tutorial<br>Keras | | | |

# Understanding javap tool

The **javap command** disassembles a class file. The javap command displays information about the fields, constructors and methods present in a class file.

## Syntax to use javap tool

Let's see how to use javap tool or command.

```
javap fully_class_name
```

## Example to use javap tool

```
javap java.lang.Object
```

**Output:**

```
Compiled from "Object.java"
public class java.lang.Object {
  public java.lang.Object();
  public final native java.lang.Class<?> getClass();
  public native int hashCode();
  public boolean equals(java.lang.Object);
  protected native java.lang.Object clone() throws java.lang.CloneNotSupportedException;
  public java.lang.String toString();
  public final native void notify();
  public final native void notifyAll();
  public final native void wait(long) throws java.lang.InterruptedException;
  public final void wait(long, int) throws java.lang.InterruptedException;
  public final void wait() throws java.lang.InterruptedException;
  protected void finalize() throws java.lang.Throwable;
  static {};
}
```

## Another example to use javap tool for your class

Let's use the javap command for our java file.

**FileName:** Simple.java

```
class Simple{
public static void main(String args[]){
System.out.println("hello java");
}
}
```

Now let's use the javap tool to disassemble the class file.

```
javap Simple
```

**Output:**

```
Compiled from "Simple.java"
class Simple {
  Simple();
  public static void main(java.lang.String[]);
}
```

## javap -c command

You can use the javap -c command to see disassembled code. The code that reflects the java bytecode.

```
javap -c Simple
```

**Output:**

```
Compiled from "Simple.java"
class Simple {
  Simple();
    Code:
       0: aload_0
       1: invokespecial #1                  // Method java/lang/Object."":()V
       4: return

  public static void main(java.lang.String[]);
    Code:
       0: getstatic     #2                 // Field java/lang/System.out:Ljava/io/PrintStream;
       3: ldc           #3                  // String hello java
       5: invokevirtual #4 // Method java/io/PrintStream.println:(Ljava/lang/String;)V
       8: return
}
```

# Options of javap tool

The important options of javap tool are as follows.

| Option | Description |
|---|---|
| -help | prints the help message. |
| -l | prints line number and local variable |
| -c | disassembles the code |
| -s | prints internal type signature |
| -sysinfo | shows system info (path, size, date, MD5 hash) |
| -constants | shows static final constants |

| | |
|---|---|
| -version | shows version information |

Let's see how one can use these options with the help of an example. For the following file (ABC.java) we will use the above-mentioned options.

**FileName:** ABC.java

```
public class ABC
{
// main method
public static void main(String argvs[])
{
// declaring an integer array
int arr[] = {6, 7, 8, 6, 8, 0, 4};

// caculating size of the array
int size = arr.length;

// printing size of the array
System.out.println("The size of the array is " + size );

System.out.println("The 8th index of the array is " + arr[8] );

}
}
```

**Command:** javap -c ABC

**Output:**

```
Compiled from "ABC.java"
public class ABC {
  public ABC();
    Code:
       0: aload_0
       1: invokespecial #1                  // Method java/lang/Object."":()V
       4: return

  public static void main(java.lang.String[]);
    Code:
       0: bipush        7
       2: newarray       int
       4: dup
       5: iconst_0
       6: bipush        6
       8: iastore
       9: dup
      10: iconst_1
      11: bipush        7
      13: iastore
      14: dup
      15: iconst_2
      16: bipush        8
      18: iastore
      19: dup
      20: iconst_3
      21: bipush        6
      23: iastore
      24: dup
      25: iconst_4
      26: bipush        8
      28: iastore
      29: dup
      30: iconst_5
      31: iconst_0
      32: iastore
      33: dup
      34: bipush        6
      36: iconst_4
      37: iastore
      38: astore_1
      39: aload_1
      40: arraylength
      41: istore_2
      42: getstatic     #7                  // Field java/lang/System.out:Ljava/io/PrintStream;
      45: iload_2
      46: invokedynamic #13,  0             // InvokeDynamic #0:makeConcatWithConstants:(I)Ljava/lang/String;
      51: invokevirtual #17                 // Method java/io/PrintStream.println:(Ljava/lang/String;)V
      54: getstatic     #7                  // Field java/lang/System.out:Ljava/io/PrintStream;
      57: aload_1
      58: bipush        8
      60: iaload
      61: invokedynamic #23,  0             // InvokeDynamic #1:makeConcatWithConstants:(I)Ljava/lang/String;
      66: invokevirtual #17                 // Method java/io/PrintStream.println:(Ljava/lang/String;)V
      69: return
}
```

**Command:** javap -l ABC

**Output:**

```
Compiled from "ABC.java"
public class ABC {
  public ABC();
    LineNumberTable:
      line 1: 0

  public static void main(java.lang.String[]);
    LineNumberTable:
      line 6: 0
      line 9: 39
      line 12: 42
      line 14: 54
      line 16: 69
}
```

**Command:** javap -s ABC

**Output:**

```
Compiled from "ABC.java"
public class ABC {
  public ABC();
    descriptor: ()V

  public static void main(java.lang.String[]);
    descriptor: ([Ljava/lang/String;)V
}
```

**Command:** javap -sysinfo ABC

**Output:**

```
Classfile /C:/Users/Nikhil Kumar/Documents/ABC.class
  Last modified Sep 11, 2021; size 970 bytes
  SHA-256 checksum 576adf03386399a4691e0ce5b6c5aa5d964b082a1a61299bac5632942e413312
  Compiled from "ABC.java"
public class ABC {
  public ABC();
  public static void main(java.lang.String[]);
}
```

**Command:** javap -constants ABC

**Output:**

```
Compiled from "ABC.java"
public class ABC {
  public ABC();
  public static void main(java.lang.String[]);
}
```

**Command:** javap -version ABC

**Output:**

```
14
Compiled from "ABC.java"
public class ABC {
  public ABC();
  public static void main(java.lang.String[]);
}
```

<<Prev Next>>

## Feedback

- Send your Feedback to feedback@javatpoint.com

## Help Others, Please Share

f 🐦 P

## Learn Latest Tutorials

| | | | | |
|---|---|---|---|---|
| Splunk tutorial<br>Splunk | SPSS tutorial<br>SPSS | Swagger tutorial<br>Swagger | T-SQL tutorial<br>Transact-SQL | Tumblr tutorial<br>Tumblr |
| React tutorial<br>ReactJS | Regex tutorial<br>Regex | Reinforcement learning tutorial<br>Reinforcement Learning | R Programming tutorial<br>R Programming | RxJS tutorial<br>RxJS |
| React Native tutorial<br>React Native | Python Design Patterns<br>Python Design Patterns | Python Pillow tutorial<br>Python Pillow | Python Turtle tutorial<br>Python Turtle | Keras tutorial<br>Keras |

## Preparation

# Creating a program that works as javap tool

Following methods of **java.lang.Class** class can be used to display the metadata of a class.

| Method | Description |
|---|---|
| public Field[] getDeclaredFields()throws SecurityException | returns an array of Field objects reflecting all the fields declared by the class or interface represented by this Class object. |
| public Constructor[] getDeclaredConstructors()throws SecurityException | returns an array of Constructor objects reflecting all the constructors declared by the class represented by this Class object. |
| public Method[] getDeclaredMethods()throws SecurityException | returns an array of Method objects reflecting all the methods declared by the class or interface represented by this Class object. |

## Example of creating javap tool

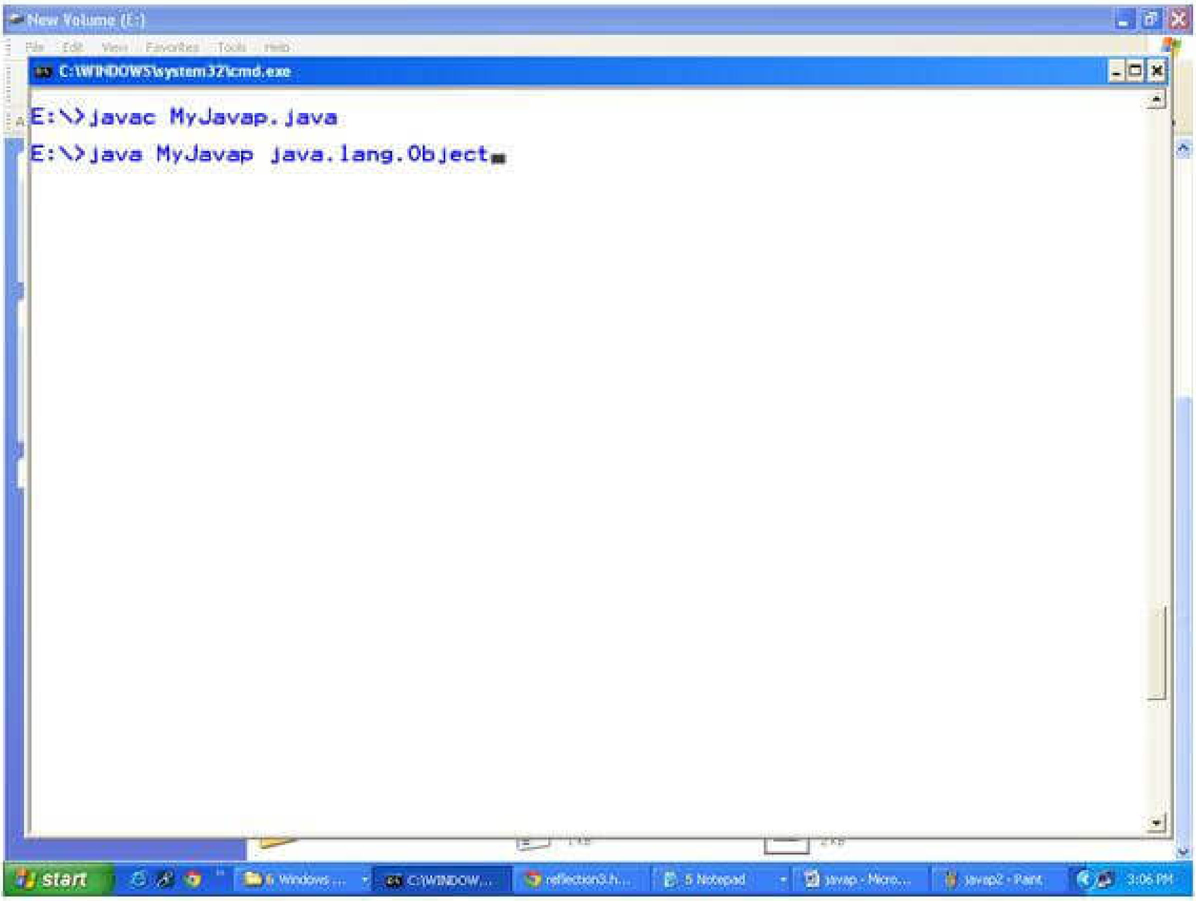
Let's create a program that works like javap tool.

```
import java.lang.reflect.*;

public class MyJavap{
  public static void main(String[] args)throws Exception {
   Class c=Class.forName(args[0]);

   System.out.println("Fields........");
   Field f[]=c.getDeclaredFields();
   for(int i=0;i<f.length;i++)
      System.out.println(f[i]);

   System.out.println("Constructors........");
   Constructor con[]=c.getDeclaredConstructors();
   for(int i=0;i<con.length;i++)
      System.out.println(con[i]);

      System.out.println("Methods........");
```

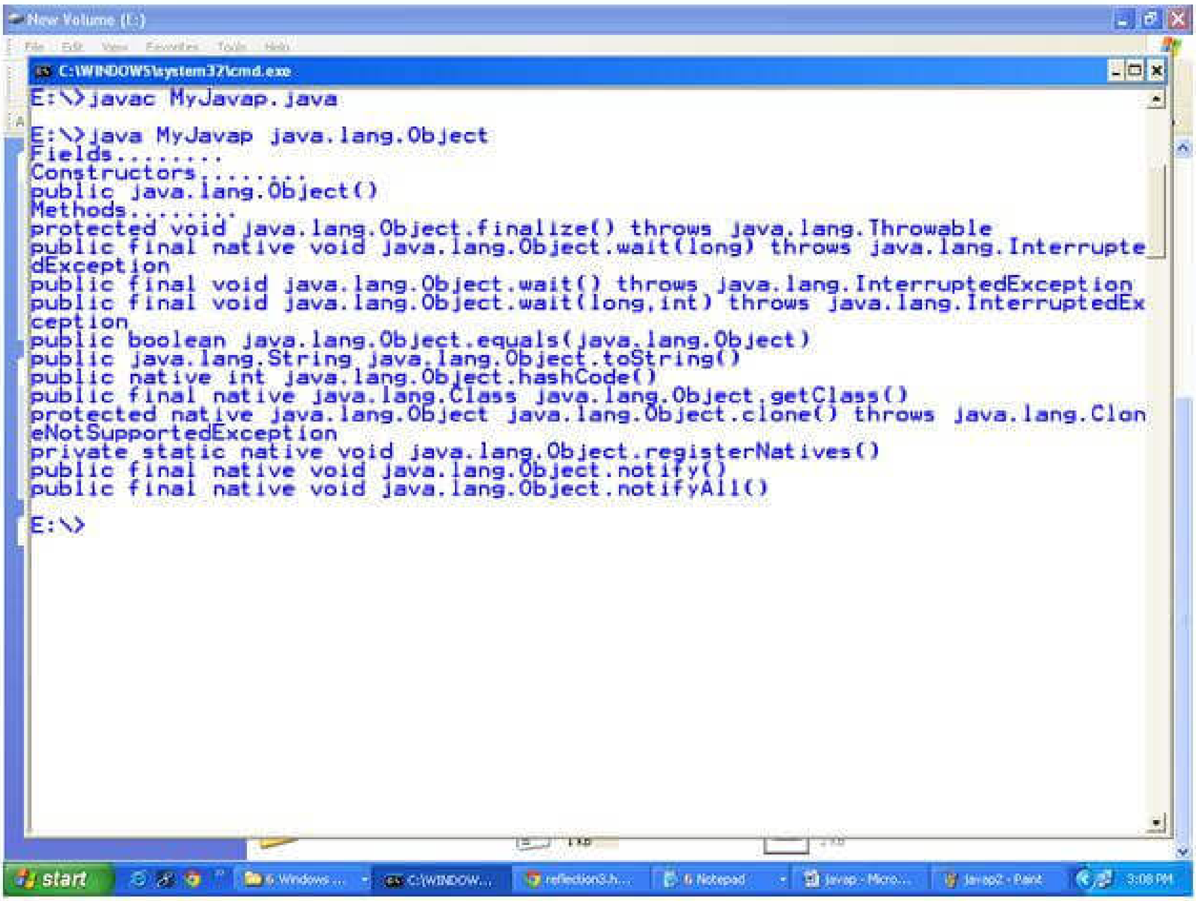
```
    Method m[]=c.getDeclaredMethods();
    for(int i=0;i<m.length;i++)
      System.out.println(m[i]);
  }
}
```

At runtime, you can get the details of any class, it may be user-defined or pre-defined class.

## Output:



```
E:\>javac MyJavap.java

E:\>java MyJavap java.lang.Object
```



```
E:\>javac MyJavap.java

E:\>java MyJavap java.lang.Object
Fields........
Constructors........
public java.lang.Object()
Methods........
protected void java.lang.Object.finalize() throws java.lang.Throwable
public final native void java.lang.Object.wait(long) throws java.lang.Interrupte
dException
public final void java.lang.Object.wait() throws java.lang.InterruptedException
public final void java.lang.Object.wait(long,int) throws java.lang.InterruptedEx
ception
public boolean java.lang.Object.equals(java.lang.Object)
public java.lang.String java.lang.Object.toString()
public native int java.lang.Object.hashCode()
public final native java.lang.Class java.lang.Object.getClass()
protected native java.lang.Object java.lang.Object.clone() throws java.lang.Clon
eNotSupportedException
private static native void java.lang.Object.registerNatives()
public final native void java.lang.Object.notify()
public final native void java.lang.Object.notifyAll()

E:\>
```

# Creating your own appletviewer

As you know well that appletviewer tool creates a frame and displays the output of applet in the frame.You can also create your frame and display the applet output.

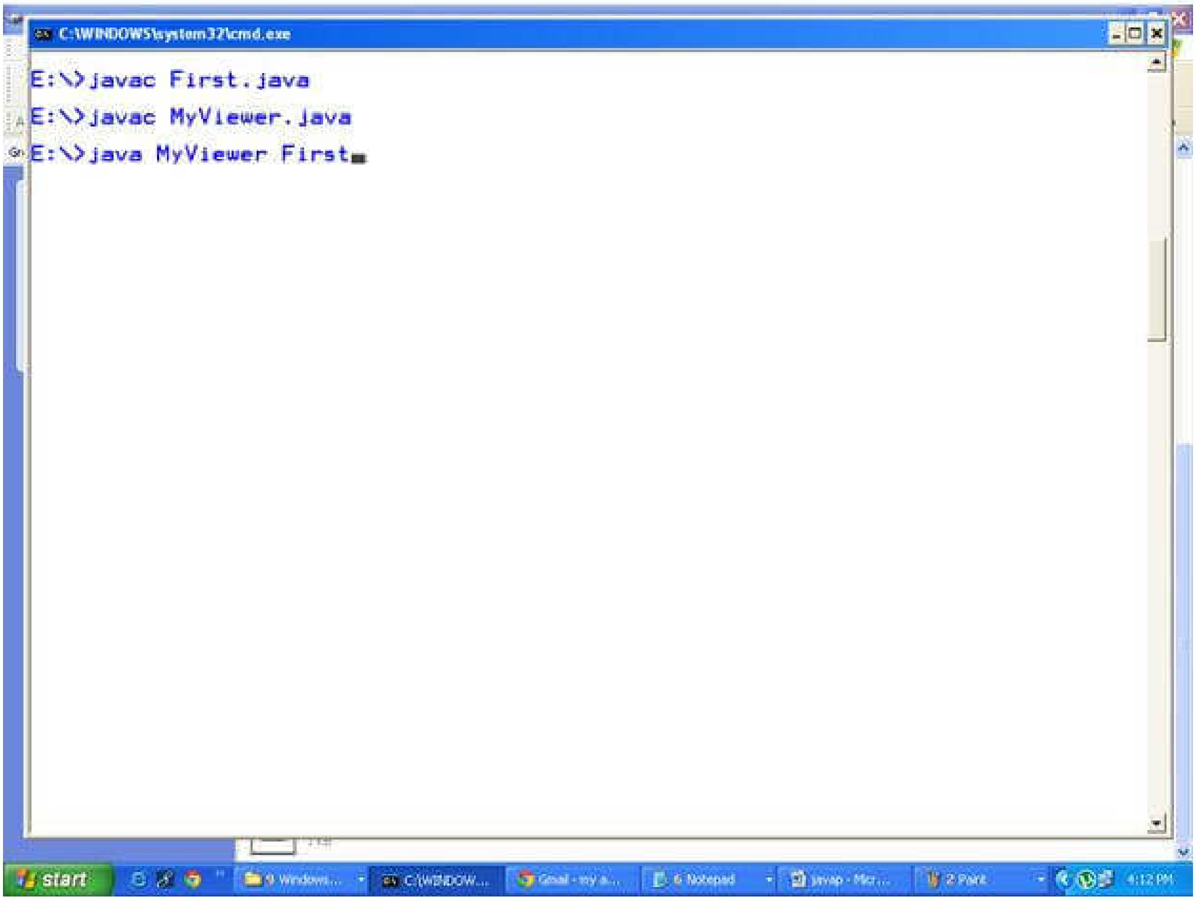
## Example that works like appletviewer tool

Let's see the simple example that works like appletviewer tool. This example displays applet on the frame.

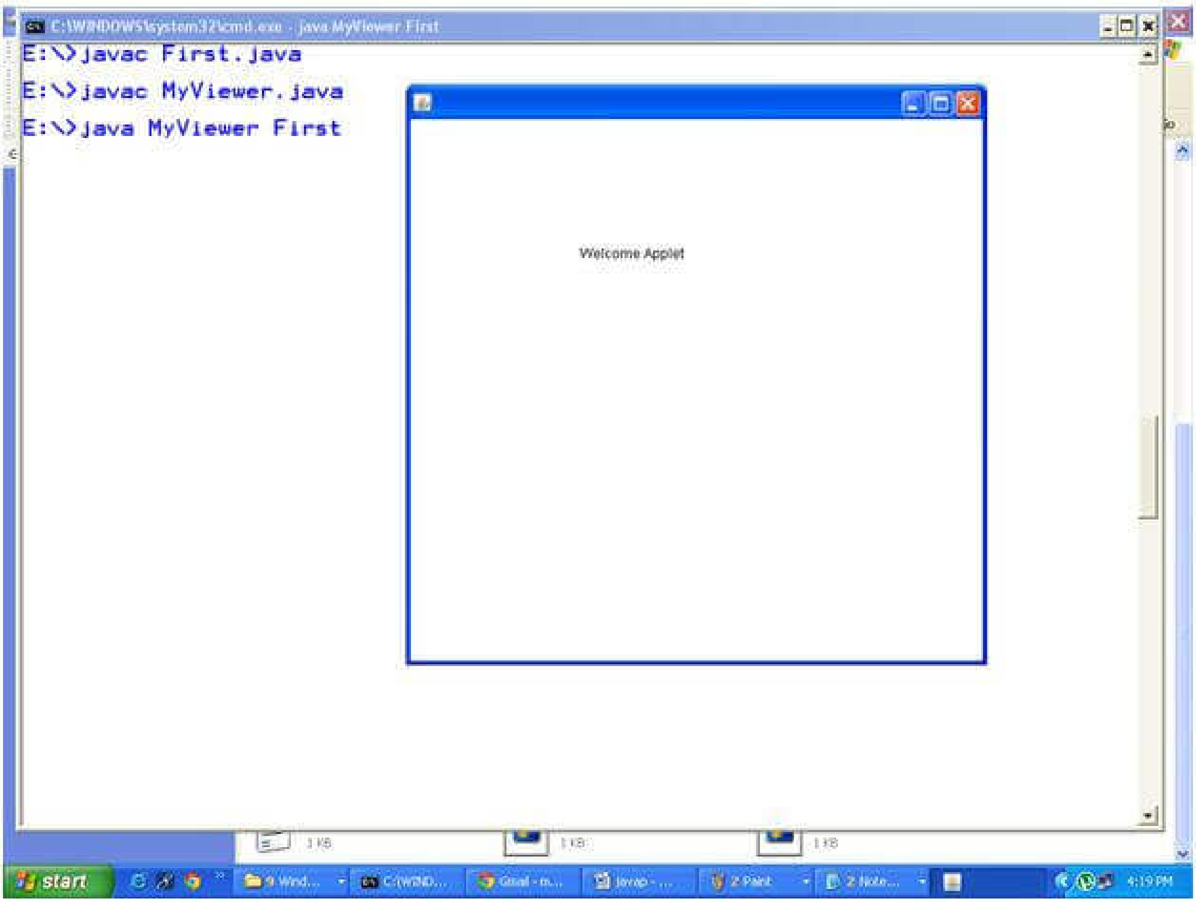
```
import java.applet.Applet;
import java.awt.Frame;
import java.awt.Graphics;

public class MyViewer extends Frame{
 public static void main(String[] args) throws Exception{
  Class c=Class.forName(args[0]);

  MyViewer v=new MyViewer();
  v.setSize(400,400);
  v.setLayout(null);
  v.setVisible(true);

  Applet a=(Applet)c.newInstance();
  a.start();
  Graphics g=v.getGraphics();
  a.paint(g);
  a.stop();

 }

}
```

```
//simple program of applet

import java.applet.Applet;
import java.awt.Graphics;

public class First extends Applet{

  public void paint(Graphics g){g.drawString("Welcome",50, 50);}
}
```

## Output:



```
E:\>javac First.java
E:\>javac MyViewer.java
E:\>java MyViewer First
```



```
E:\>javac First.java
E:\>javac MyViewer.java
E:\>java MyViewer First
```

# How to call private method from another class in java

You can call the private method from outside the class by changing the runtime behaviour of the class.

With the help of **java.lang.Class** class and **java.lang.reflect.Method** class, we can call a private method from any other class.

## Required methods of Method class

**1) public void setAccessible(boolean status) throws SecurityException** sets the accessibility of the method.

**2) public Object invoke(Object method, Object... args) throws IllegalAccessException, IllegalArgumentException, InvocationTargetException** is used to invoke the method.

## Required method of Class class

**1) public Method getDeclaredMethod(String name,Class[] parameterTypes)throws NoSuchMethodException,SecurityException:** returns a Method object that reflects the specified declared method of the class or interface represented by this Class object.

## Example of calling private method from another class

Let's see the simple example to call private method from another class.

*File: A.java*

```
public class A {
  private void message(){System.out.println("hello java"); }
}
```

*File: MethodCall.java*

```
import java.lang.reflect.Method;
public class MethodCall{
public static void main(String[] args)throws Exception{

    Class c = Class.forName("A");
    Object o= c.newInstance();
    Method m =c.getDeclaredMethod("message", null);
    m.setAccessible(true);
    m.invoke(o, null);
}
}
```

**Output:**

```
hello java
```

## Another example to call parameterized private method from another class

Let's see the example to call parameterized private method from another class

*File: A.java*

```
class A{
private void cube(int n){System.out.println(n*n*n);}
}
```

*File: M.java*

```
import java.lang.reflect.*;
class M{
public static void main(String args[])throws Exception{
Class c=A.class;
Object obj=c.newInstance();

Method m=c.getDeclaredMethod("cube",new Class[]{int.class});
m.setAccessible(true);
m.invoke(obj,4);
}}
```

**Output:**

```
64
```

## Accessing Private Constructors of a class

We know that constructors of a class are a special kind of method this is used to instantiate the class. To access the private constructor, we use the method getDeclaredConstructor(). The getDeclaredConstructor() is used to access a parameterless as well as a parametrized constructor of a class. The following example shows the same.

**FileName:** PvtConstructorDemo.java

```
// important import statements
import java.lang.reflect.Constructor;
import java.lang.reflect.Modifier;
import java.lang.reflect.InvocationTargetException;

class Vehicle
{

// private fields of the class Vehicle
private Integer vId;
private String vName;

// parameterless constructor
private Vehicle()
{


}

// parameterized constructor
private Vehicle(Integer vId, String vName)
{
  this.vId = vId;
  this.vName = vName;
}

// setter methods of the class Vehicle
```

```
public void setVehicleId(Integer vId)
{
  this.vId = vId;
}

public void setVehicleName(String vName)
{
  this.vName = vName;
}



// getter methods of the class Vehicle
public Integer getVehicleId()
{
  return vId;
}

public String getVehicleName()
{
 return vName;
}
}




public class PvtConstructorDemo
{
// the createObj() method is used to create an object of
// the Vehicle class using the parameterless constructor.
public void craeteObj(int vId, String vName) throws InstantiationException, IllegalAccessException
IllegalArgumentException, InvocationTargetException, NoSuchMethodException
{
// using the parametereless contructor
Constructor<Vehicle> constt = Vehicle.class.getDeclaredConstructor();

constt.setAccessible(true);
Object obj = constt.newInstance();
if (obj instanceof Vehicle)
{
 Vehicle v = (Vehicle)obj;
  v.setVehicleId(vId);
  v.setVehicleName(vName);
    System.out.println("Vehicle Id: " +  v.getVehicleId());
    System.out.println("Vehicle Name: " +  v.getVehicleName());
}
}

// the craeteObjByConstructorName() method is used to create an object
// of the Vehicle class using the parameterized constructor.
public void craeteObjByConstructorName(int vId, String vName) throws NoSuchMethodException
InstantiationException, IllegalAccessException, IllegalArgumentException, InvocationTargetExceptio
{

// using the parameterized contructor
Constructor<Vehicle> constt = Vehicle.class.getDeclaredConstructor(Integer.class, String.class);

if (Modifier.isPrivate(constt.getModifiers()))
{
constt.setAccessible(true);

Object obj = constt.newInstance(vId, vName);
if(obj instanceof Vehicle)
{
    Vehicle v = (Vehicle)obj;
    System.out.println("Vehicle Id: " +  v.getVehicleId());
    System.out.println("Vehicle Name: " + v.getVehicleName());
}
}
}




// delegating the responsibility to Java Virtual Machine (JVM) to handle the raised
// exception
// main method
public static void main(String argvs[]) throws InstantiationException,
IllegalAccessException, IllegalArgumentException, InvocationTargetException,
NoSuchMethodException, SecurityException
{

  // creating an object of the class PvtConstructorDemo
  PvtConstructorDemo ob = new PvtConstructorDemo();
  ob.craeteObj(20, "Indica");
  System.out.println(" -------------------------- ");
  ob.craeteObjByConstructorName(30, "Alto");
}
}
```

**Output:**

```
Vehicle Id: 20
Vehicle Name: Indica
 --------------------------
Vehicle Id: 30
Vehicle Name: Alto
```

← Prev Next →