## **What is Reflection?**

In brief, reflection is the ability of a program to examine and modify the structure and behavior of an object at runtime.

This concept is sometimes mixed with introspection. Introspection (Type introspection) is the ability of a program to examine the type or properties of an object at runtime. Therefore, it is a subset of reflection. Some languages support introspection, but do not support reflection, e.g., C++.

## **Why do we need reflection?**

Reflection enables us to do the following:

* Examine an object’s class at runtime
* Construct an object for a class at runtime
* Examine a class’s field and method at runtime
* Invoke any method of an object at runtime

For example, [JUnit](http://www.programcreek.com/2012/02/junit-tutorial-2-annotations/) use reflection to look through methods tagged with the @Test annotation, and then call those methods when running the unit test. (Here is a set of examples of [how to use JUnit](http://www.programcreek.com/2012/02/junit-tutorial-2-annotations/).)

#### **Example 1: Get class name from object**

|  |  |
| --- | --- |
| 01 | package myreflection; |
| 02 | import java.lang.reflect.Method; |

|  |  |
| --- | --- |
| 03 |  |
| 04 | public class ReflectionHelloWorld { |

|  |  |
| --- | --- |
| 05 | public static void main(String[] args){ |
| 06 | Foo f = new Foo(); |

|  |  |
| --- | --- |
| 07 | System.out.println(f.getClass().getName()); |
| 08 | } |

|  |  |
| --- | --- |
| 09 | } |
| 10 |  |

|  |  |
| --- | --- |
| 11 | class Foo { |
| 12 | public void print() { |

|  |  |
| --- | --- |
| 13 | System.out.println("abc"); |
| 14 | } |

|  |  |
| --- | --- |
| 15 | } |

**Output:**

|  |  |
| --- | --- |
| 1 | myreflection.Foo |

#### **Example 2: Invoke method on unknown object**

For the code example below, image the types of an object is unknown. By using reflection, the code can use the object and find out if the object has a method called “print” and then call it.

|  |  |
| --- | --- |
| 01 | package myreflection; |
| 02 | import java.lang.reflect.Method; |

|  |  |
| --- | --- |
| 03 |  |
| 04 | public class ReflectionHelloWorld { |

|  |  |
| --- | --- |
| 05 | public static void main(String[] args){ |
| 06 | Foo f = new Foo(); |

|  |  |
| --- | --- |
| 07 |  |
| 08 | Method method; |

|  |  |
| --- | --- |
| 09 | try { |
| 10 | method = f.getClass().getMethod("print", new Class<?>[0]); |

|  |  |
| --- | --- |
| 11 | method.invoke(f); |
| 12 | } catch (Exception e) { |

|  |  |
| --- | --- |
| 13 | e.printStackTrace(); |
| 14 | } |

|  |  |
| --- | --- |
| 15 | } |
| 16 | } |

|  |  |
| --- | --- |
| 17 |  |
| 18 | class Foo { |

|  |  |
| --- | --- |
| 19 | public void print() { |
| 20 | System.out.println("abc"); |

|  |  |  |
| --- | --- | --- |
| 21 | | } |
| 22 | | } |
| 1 | abc | |

#### **Example 3: Create object from Class instance**

|  |  |
| --- | --- |
| 01 | package myreflection; |
| 02 |  |

|  |  |
| --- | --- |
| 03 | public class ReflectionHelloWorld { |
| 04 | public static void main(String[] args){ |

|  |  |
| --- | --- |
| 05 | //create instance of "Class" |
| 06 | Class<?> c = null; |

|  |  |
| --- | --- |
| 07 | try{ |
| 08 | c=Class.forName("myreflection.Foo"); |

|  |  |
| --- | --- |
| 09 | }catch(Exception e){ |
| 10 | e.printStackTrace(); |

|  |  |
| --- | --- |
| 11 | } |
| 12 |  |

|  |  |
| --- | --- |
| 13 | //create instance of "Foo" |
| 14 | Foo f = null; |

|  |  |
| --- | --- |
| 15 |  |
| 16 | try { |

|  |  |
| --- | --- |
| 17 | f = (Foo) c.newInstance(); |
| 18 | } catch (Exception e) { |

|  |  |
| --- | --- |
| 19 | e.printStackTrace(); |
| 20 | } |

|  |  |
| --- | --- |
| 21 |  |
| 22 | f.print(); |

|  |  |
| --- | --- |
| 23 | } |
| 24 | } |

|  |  |
| --- | --- |
| 25 |  |
| 26 | class Foo { |

|  |  |
| --- | --- |
| 27 | public void print() { |
| 28 | System.out.println("abc"); |

|  |  |
| --- | --- |
| 29 | } |
| 30 | } |

#### **Example 4: Get constructor and create instance**

|  |  |
| --- | --- |
| 01 | package myreflection; |
| 02 |  |

|  |  |
| --- | --- |
| 03 | import java.lang.reflect.Constructor; |
| 04 |  |

|  |  |
| --- | --- |
| 05 | public class ReflectionHelloWorld { |
| 06 | public static void main(String[] args){ |

|  |  |
| --- | --- |
| 07 | //create instance of "Class" |
| 08 | Class<?> c = null; |

|  |  |
| --- | --- |
| 09 | try{ |
| 10 | c=Class.forName("myreflection.Foo"); |

|  |  |
| --- | --- |
| 11 | }catch(Exception e){ |
| 12 | e.printStackTrace(); |

|  |  |
| --- | --- |
| 13 | } |
| 14 |  |

|  |  |
| --- | --- |
| 15 | //create instance of "Foo" |
| 16 | Foo f1 = null; |

|  |  |
| --- | --- |
| 17 | Foo f2 = null; |
| 18 |  |

|  |  |
| --- | --- |
| 19 | //get all constructors |
| 20 | Constructor<?> cons[] = c.getConstructors(); |

|  |  |
| --- | --- |
| 21 |  |
| 22 | try { |

|  |  |
| --- | --- |
| 23 | f1 = (Foo) cons[0].newInstance(); |
| 24 | f2 = (Foo) cons[1].newInstance("abc"); |

|  |  |
| --- | --- |
| 25 | } catch (Exception e) { |
| 26 | e.printStackTrace(); |

|  |  |
| --- | --- |
| 27 | } |
| 28 |  |

|  |  |
| --- | --- |
| 29 | f1.print(); |
| 30 | f2.print(); |

|  |  |
| --- | --- |
| 31 | } |
| 32 | } |

|  |  |
| --- | --- |
| 33 |  |
| 34 | class Foo { |

|  |  |
| --- | --- |
| 35 | String s; |
| 36 |  |

|  |  |
| --- | --- |
| 37 | public Foo(){} |
| 38 |  |

|  |  |
| --- | --- |
| 39 | public Foo(String s){ |
| 40 | this.s=s; |

|  |  |
| --- | --- |
| 41 | } |
| 42 |  |

|  |  |
| --- | --- |
| 43 | public void print() { |
| 44 | System.out.println(s); |

|  |  |
| --- | --- |
| 45 | } |
| 46 | } |

**Output:**

|  |  |
| --- | --- |
| 1 | null |
| 2 |  |

|  |  |
| --- | --- |
| 3 | abc |

In addition, you can use Class instance to get implemented interfaces, super class, declared field, etc.

#### **Example 5: Change array size though reflection**

[view source](https://www.javacodegeeks.com/2013/09/java-reflection-tutorial.html#viewSource)[print](https://www.javacodegeeks.com/2013/09/java-reflection-tutorial.html#printSource)[?](https://www.javacodegeeks.com/2013/09/java-reflection-tutorial.html#about)

|  |  |
| --- | --- |
| 01 | package myreflection; |
| 02 |  |

|  |  |
| --- | --- |
| 03 | import java.lang.reflect.Array; |
| 04 |  |

|  |  |
| --- | --- |
| 05 | public class ReflectionHelloWorld { |
| 06 | public static void main(String[] args) { |

|  |  |
| --- | --- |
| 07 | int[] intArray = { 1, 2, 3, 4, 5 }; |
| 08 | int[] newIntArray = (int[]) changeArraySize(intArray, 10); |

|  |  |
| --- | --- |
| 09 | print(newIntArray); |
| 10 |  |

|  |  |
| --- | --- |
| 11 | String[] atr = { "a", "b", "c", "d", "e" }; |
| 12 | String[] str1 = (String[]) changeArraySize(atr, 10); |

|  |  |
| --- | --- |
| 13 | print(str1); |
| 14 | } |

|  |  |
| --- | --- |
| 15 |  |
| 16 | // change array size |

|  |  |
| --- | --- |
| 17 | public static Object changeArraySize(Object obj, int len) { |
| 18 | Class<?> arr = obj.getClass().getComponentType(); |

|  |  |
| --- | --- |
| 19 | Object newArray = Array.newInstance(arr, len); |
| 20 |  |

|  |  |
| --- | --- |
| 21 | //do array copy |
| 22 | int co = Array.getLength(obj); |

|  |  |
| --- | --- |
| 23 | System.arraycopy(obj, 0, newArray, 0, co); |
| 24 | return newArray; |

|  |  |
| --- | --- |
| 25 | } |
| 26 |  |

|  |  |
| --- | --- |
| 27 | // print |
| 28 | public static void print(Object obj) { |

|  |  |
| --- | --- |
| 29 | Class<?> c = obj.getClass(); |
| 30 | if (!c.isArray()) { |

|  |  |
| --- | --- |
| 31 | return; |
| 32 | } |

|  |  |
| --- | --- |
| 33 |  |
| 34 | System.out.println("\nArray length: " + Array.getLength(obj)); |

|  |  |
| --- | --- |
| 35 |  |
| 36 | for (int i = 0; i < Array.getLength(obj); i++) { |

|  |  |
| --- | --- |
| 37 | System.out.print(Array.get(obj, i) + " "); |
| 38 | } |

|  |  |
| --- | --- |
| 39 | } |
| 40 | } |

**Output:**

|  |  |
| --- | --- |
| 1 | Array length: 10 |
| 2 | 1 2 3 4 5 0 0 0 0 0 |

|  |  |
| --- | --- |
| 3 | Array length: 10 |
| 4 | a b c d e null null null null null |

## **Summary**

The above code examples shows a very small set of functions provided by Java reflection. Reading those examples may only give you a taste of Java reflection, you may want to [Read more information on Oracle website](http://docs.oracle.com/javase/tutorial/reflect/).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**next →**](https://www.javatpoint.com/new-instance()-method)[**← prev**](https://www.javatpoint.com/Applet-Communication)  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 we can 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.   1. **class** Simple{} 3. **class** Test{ 4. **public** **static** **void** main(String args[]){ 5. Class c=Class.forName("Simple"); 6. System.out.println(c.getName()); 7. } 8. }   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.   1. **class** Simple{} 3. **class** Test{ 4. **void** printName(Object obj){ 5. Class c=obj.getClass(); 6. System.out.println(c.getName()); 7. } 8. **public** **static** **void** main(String args[]){ 9. Simple s=**new** Simple(); 11. Test t=**new** Test(); 12. t.printName(s); 13. } 14. }   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 type also.   1. **class** Test{ 2. **public** **static** **void** main(String args[]){ 3. Class c = **boolean**.**class**; 4. System.out.println(c.getName()); 6. Class c2 = Test.**class**; 7. System.out.println(c2.getName()); 8. } 9. }   boolean  Test  Determining the class object  Following methods of Class class is 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.   1. **class** Simple{} 2. **interface** My{} 4. **class** Test{ 5. **public** **static** **void** main(String args[]){ 6. **try**{ 7. Class c=Class.forName("Simple"); 8. System.out.println(c.isInterface()); 10. Class c2=Class.forName("My"); 11. System.out.println(c2.isInterface()); 13. }**catch**(Exception e){System.out.println(e);} 15. } 16. }   false  true  Next Topics of Reflection API Tutorial  [newInstance() method](https://www.javatpoint.com/new-instance()-method)  [Understanding javap tool](https://www.javatpoint.com/understanding-javap-tool)  [creating javap tool](https://www.javatpoint.com/creating-javap-tool)  [creating appletviewer tool](https://www.javatpoint.com/creating-appletviewer-tool)  [Call private method from another class](https://www.javatpoint.com/how-to-call-private-method-from-another-class-in-java) |

# 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 it like Object class. You will learn about generics later.

### Example of newInstance() method

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

1. **class** Simple{
2. **void** message(){System.out.println("Hello Java");}
3. }
5. **class** Test{
6. **public** **static** **void** main(String args[]){
7. **try**{
8. Class c=Class.forName("Simple");
9. Simple s=(Simple)c.newInstance();
10. s.message();
12. }**catch**(Exception e){System.out.println(e);}
14. }
15. }

Output:Hello java