## Java Reflection

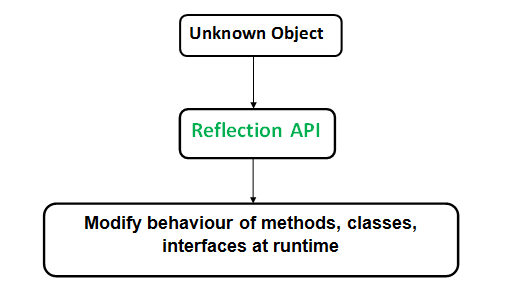
1. **Reflection API**
2. **newInstance() method**
3. [**javap tool**](https://www.javatpoint.com/understanding-javap-tool)
4. **creating javap tool**
5. **Call private method**

**Reference:**  
<https://docs.oracle.com/javase/tutorial/reflect/index.html>

Reflection in Java

Reflection is an API which is used to examine or modify the behaviour of methods, classes, interfaces at runtime.

* The required classes for reflection are provided under java.lang.reflect package.
* Reflection gives us information about the class to which an object belongs and also the methods of that class which can be executed by using the object.
* Through reflection we can invoke methods at runtime irrespective of the access specifier used with them.

  
Reflection can be used to get information about –

1. **Class** The getClass() method is used to get the name of the class to which an object belongs.
2. **Constructors** The getConstructors() method is used to get the public constructors of the class to which an object belongs.
3. **Methods** The getMethods() method is used to get the public methods of the class to which an objects belongs.

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 access the private method from outside the 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. |

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 behaviour of a class.

Commonly used methods of Class class:

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

# 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

# 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.

1. javap fully\_class\_name

#### **Example to use javap tool**

1. javap java.lang.Object

**Output:**

1. Compiled from "Object.java"
2. **public** **class** java.lang.Object {
3. **public** java.lang.Object();
4. **public** **final** **native** java.lang.Class<?> getClass();
5. **public** **native** **int** hashCode();
6. **public** **boolean** equals(java.lang.Object);
7. **protected** **native** java.lang.Object clone() **throws** java.lang.CloneNotSupportedException;
8. **public** java.lang.String toString();
9. **public** **final** **native** **void** notify();
10. **public** **final** **native** **void** notifyAll();
11. **public** **final** **native** **void** wait(**long**) **throws** java.lang.InterruptedException;
12. **public** **final** **void** wait(**long**, **int**) **throws** java.lang.InterruptedException;
13. **public** **final** **void** wait() **throws** java.lang.InterruptedException;
14. **protected** **void** finalize() **throws** java.lang.Throwable;
15. **static** {};
16. }

#### **Another example to use javap tool for your class**

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

1. **class** Simple{
2. **public** **static** **void** main(String args[]){
3. System.out.println("hello java");
4. }
5. }

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

1. javap Simple

**Output:**

1. Compiled from ".java"
2. **class** Simple {
3. Simple();
4. **public** **static** **void** main(java.lang.String[]);
5. }

#### **javap -c command**

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

1. javap -c Simple

**Output:**

1. Compiled from ".java"
2. **class** Simple {
3. Simple();
4. Code:
5. 0: aload\_0
6. 1: invokespecial #1                  // Method java/lang/Object."<init>":()V
7. 4: **return**
9. **public** **static** **void** main(java.lang.String[]);
10. Code:
11. 0: getstatic     #2                  // Field java/lang/System.out:Ljava/io/PrintStream;
12. 3: ldc           #3                  // String hello java
13. 5: invokevirtual #4                  // Method java/io/PrintStream.println:(Ljava/lang/String;)V
14. 8: **return**
15. }

## **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 |

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.

1. **import** java.lang.reflect.\*;
3. **public** **class** MyJavap{
4. **public** **static** **void** main(String[] args)**throws** Exception {
5. Class c=Class.forName(args[0]);
7. System.out.println("Fields........");
8. Field f[]=c.getDeclaredFields();
9. **for**(**int** i=0;i<f.length;i++)
10. System.out.println(f[i]);
12. System.out.println("Constructors........");
13. Constructor con[]=c.getDeclaredConstructors();
14. **for**(**int** i=0;i<con.length;i++)
15. System.out.println(con[i]);
17. System.out.println("Methods........");
18. Method m[]=c.getDeclaredMethods();
19. **for**(**int** i=0;i<m.length;i++)
20. System.out.println(m[i]);
21. }
22. }

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

Output:

# creating a program that works like javap toolcreating a program that works like javap toolHow 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.

By the help of **java.lang.Class** class and **java.lang.reflect.Method** class, we can call 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*

1. **public** **class** A {
2. **private** **void** message(){System.out.println("hello java"); }
3. }

*File: MethodCall.java*

1. **import** java.lang.reflect.Method;
2. **public** **class** MethodCall{
3. **public** **static** **void** main(String[] args)**throws** Exception{
5. Class c = Class.forName("A");
6. Object o= c.newInstance();
7. Method m =c.getDeclaredMethod("message", **null**);
8. m.setAccessible(**true**);
9. m.invoke(o, **null**);
10. }
11. }

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*

1. **class** A{
2. **private** **void** cube(**int** n){System.out.println(n\*n\*n);}
3. }

*File: M.java*

1. **import** java.lang.reflect.\*;
2. **class** M{
3. **public** **static** **void** main(String args[])**throws** Exception{
4. Class c=A.**class**;
5. Object obj=c.newInstance();
7. Method m=c.getDeclaredMethod("cube",**new** Class[]{**int**.**class**});
8. m.setAccessible(**true**);
9. m.invoke(obj,4);
10. }}

Output:64