| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/Proxy.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/java/lang/reflect/ParameterizedType.html)   [**NEXT CLASS**](http://docs.google.com/java/lang/reflect/ReflectPermission.html) | [**FRAMES**](http://docs.google.com/index.html?java/lang/reflect/Proxy.html)    [**NO FRAMES**](http://docs.google.com/Proxy.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | [FIELD](#3znysh7) | [CONSTR](#2et92p0) | [METHOD](#tyjcwt) | DETAIL: [FIELD](#1t3h5sf) | [CONSTR](#2s8eyo1) | [METHOD](#3rdcrjn) |

## **java.lang.reflect**

Class Proxy

[java.lang.Object](http://docs.google.com/java/lang/Object.html)  
 **java.lang.reflect.Proxy**

**All Implemented Interfaces:** [Serializable](http://docs.google.com/java/io/Serializable.html)

public class **Proxy**extends [Object](http://docs.google.com/java/lang/Object.html)implements [Serializable](http://docs.google.com/java/io/Serializable.html)

Proxy provides static methods for creating dynamic proxy classes and instances, and it is also the superclass of all dynamic proxy classes created by those methods.

To create a proxy for some interface Foo:

InvocationHandler handler = new MyInvocationHandler(...);  
 Class proxyClass = Proxy.getProxyClass(  
 Foo.class.getClassLoader(), new Class[] { Foo.class });  
 Foo f = (Foo) proxyClass.  
 getConstructor(new Class[] { InvocationHandler.class }).  
 newInstance(new Object[] { handler });

or more simply:

Foo f = (Foo) Proxy.newProxyInstance(Foo.class.getClassLoader(),  
 new Class[] { Foo.class },  
 handler);

A *dynamic proxy class* (simply referred to as a *proxy class* below) is a class that implements a list of interfaces specified at runtime when the class is created, with behavior as described below. A *proxy interface* is such an interface that is implemented by a proxy class. A *proxy instance* is an instance of a proxy class. Each proxy instance has an associated *invocation handler* object, which implements the interface [InvocationHandler](http://docs.google.com/java/lang/reflect/InvocationHandler.html). A method invocation on a proxy instance through one of its proxy interfaces will be dispatched to the [invoke](http://docs.google.com/java/lang/reflect/InvocationHandler.html#invoke(java.lang.Object,%20java.lang.reflect.Method,%20java.lang.Object%5B%5D)) method of the instance's invocation handler, passing the proxy instance, a java.lang.reflect.Method object identifying the method that was invoked, and an array of type Object containing the arguments. The invocation handler processes the encoded method invocation as appropriate and the result that it returns will be returned as the result of the method invocation on the proxy instance.

A proxy class has the following properties:

* Proxy classes are public, final, and not abstract.
* The unqualified name of a proxy class is unspecified. The space of class names that begin with the string "$Proxy" should be, however, reserved for proxy classes.
* A proxy class extends java.lang.reflect.Proxy.
* A proxy class implements exactly the interfaces specified at its creation, in the same order.
* If a proxy class implements a non-public interface, then it will be defined in the same package as that interface. Otherwise, the package of a proxy class is also unspecified. Note that package sealing will not prevent a proxy class from being successfully defined in a particular package at runtime, and neither will classes already defined by the same class loader and the same package with particular signers.
* Since a proxy class implements all of the interfaces specified at its creation, invoking getInterfaces on its Class object will return an array containing the same list of interfaces (in the order specified at its creation), invoking getMethods on its Class object will return an array of Method objects that include all of the methods in those interfaces, and invoking getMethod will find methods in the proxy interfaces as would be expected.
* The [Proxy.isProxyClass](http://docs.google.com/java/lang/reflect/Proxy.html#isProxyClass(java.lang.Class)) method will return true if it is passed a proxy class-- a class returned by Proxy.getProxyClass or the class of an object returned by Proxy.newProxyInstance-- and false otherwise.
* The java.security.ProtectionDomain of a proxy class is the same as that of system classes loaded by the bootstrap class loader, such as java.lang.Object, because the code for a proxy class is generated by trusted system code. This protection domain will typically be granted java.security.AllPermission.
* Each proxy class has one public constructor that takes one argument, an implementation of the interface [InvocationHandler](http://docs.google.com/java/lang/reflect/InvocationHandler.html), to set the invocation handler for a proxy instance. Rather than having to use the reflection API to access the public constructor, a proxy instance can be also be created by calling the [Proxy.newInstance](http://docs.google.com/java/lang/reflect/Proxy.html#newProxyInstance(java.lang.ClassLoader,%20java.lang.Class%5B%5D,%20java.lang.reflect.InvocationHandler)) method, which combines the actions of calling [Proxy.getProxyClass](http://docs.google.com/java/lang/reflect/Proxy.html#getProxyClass(java.lang.ClassLoader,%20java.lang.Class...)) with invoking the constructor with an invocation handler.

A proxy instance has the following properties:

* Given a proxy instance proxy and one of the interfaces implemented by its proxy class Foo, the following expression will return true:  
   proxy instanceof Foo  
   and the following cast operation will succeed (rather than throwing a ClassCastException):  
   (Foo) proxy
* Each proxy instance has an associated invocation handler, the one that was passed to its constructor. The static [Proxy.getInvocationHandler](http://docs.google.com/java/lang/reflect/Proxy.html#getInvocationHandler(java.lang.Object)) method will return the invocation handler associated with the proxy instance passed as its argument.
* An interface method invocation on a proxy instance will be encoded and dispatched to the invocation handler's [invoke](http://docs.google.com/java/lang/reflect/InvocationHandler.html#invoke(java.lang.Object,%20java.lang.reflect.Method,%20java.lang.Object%5B%5D)) method as described in the documentation for that method.
* An invocation of the hashCode, equals, or toString methods declared in java.lang.Object on a proxy instance will be encoded and dispatched to the invocation handler's invoke method in the same manner as interface method invocations are encoded and dispatched, as described above. The declaring class of the Method object passed to invoke will be java.lang.Object. Other public methods of a proxy instance inherited from java.lang.Object are not overridden by a proxy class, so invocations of those methods behave like they do for instances of java.lang.Object.

### Methods Duplicated in Multiple Proxy Interfaces

When two or more interfaces of a proxy class contain a method with the same name and parameter signature, the order of the proxy class's interfaces becomes significant. When such a *duplicate method* is invoked on a proxy instance, the Method object passed to the invocation handler will not necessarily be the one whose declaring class is assignable from the reference type of the interface that the proxy's method was invoked through. This limitation exists because the corresponding method implementation in the generated proxy class cannot determine which interface it was invoked through. Therefore, when a duplicate method is invoked on a proxy instance, the Method object for the method in the foremost interface that contains the method (either directly or inherited through a superinterface) in the proxy class's list of interfaces is passed to the invocation handler's invoke method, regardless of the reference type through which the method invocation occurred.

If a proxy interface contains a method with the same name and parameter signature as the hashCode, equals, or toString methods of java.lang.Object, when such a method is invoked on a proxy instance, the Method object passed to the invocation handler will have java.lang.Object as its declaring class. In other words, the public, non-final methods of java.lang.Object logically precede all of the proxy interfaces for the determination of which Method object to pass to the invocation handler.

Note also that when a duplicate method is dispatched to an invocation handler, the invoke method may only throw checked exception types that are assignable to one of the exception types in the throws clause of the method in *all* of the proxy interfaces that it can be invoked through. If the invoke method throws a checked exception that is not assignable to any of the exception types declared by the method in one of the proxy interfaces that it can be invoked through, then an unchecked UndeclaredThrowableException will be thrown by the invocation on the proxy instance. This restriction means that not all of the exception types returned by invoking getExceptionTypes on the Method object passed to the invoke method can necessarily be thrown successfully by the invoke method.

**Since:** 1.3 **See Also:**[InvocationHandler](http://docs.google.com/java/lang/reflect/InvocationHandler.html), [Serialized Form](http://docs.google.com/serialized-form.html#java.lang.reflect.Proxy)

| **Field Summary** | |
| --- | --- |
| protected  [InvocationHandler](http://docs.google.com/java/lang/reflect/InvocationHandler.html) | [**h**](http://docs.google.com/java/lang/reflect/Proxy.html#h)            the invocation handler for this proxy instance. |

| **Constructor Summary** | |
| --- | --- |
| protected | [**Proxy**](http://docs.google.com/java/lang/reflect/Proxy.html#Proxy(java.lang.reflect.InvocationHandler))([InvocationHandler](http://docs.google.com/java/lang/reflect/InvocationHandler.html) h)            Constructs a new Proxy instance from a subclass (typically, a dynamic proxy class) with the specified value for its invocation handler. |

| **Method Summary** | |
| --- | --- |
| static [InvocationHandler](http://docs.google.com/java/lang/reflect/InvocationHandler.html) | [**getInvocationHandler**](http://docs.google.com/java/lang/reflect/Proxy.html#getInvocationHandler(java.lang.Object))([Object](http://docs.google.com/java/lang/Object.html) proxy)            Returns the invocation handler for the specified proxy instance. |
| static [Class](http://docs.google.com/java/lang/Class.html)<?> | [**getProxyClass**](http://docs.google.com/java/lang/reflect/Proxy.html#getProxyClass(java.lang.ClassLoader,%20java.lang.Class...))([ClassLoader](http://docs.google.com/java/lang/ClassLoader.html) loader, [Class](http://docs.google.com/java/lang/Class.html)<?>... interfaces)            Returns the java.lang.Class object for a proxy class given a class loader and an array of interfaces. |
| static boolean | [**isProxyClass**](http://docs.google.com/java/lang/reflect/Proxy.html#isProxyClass(java.lang.Class))([Class](http://docs.google.com/java/lang/Class.html)<?> cl)            Returns true if and only if the specified class was dynamically generated to be a proxy class using the getProxyClass method or the newProxyInstance method. |
| static [Object](http://docs.google.com/java/lang/Object.html) | [**newProxyInstance**](http://docs.google.com/java/lang/reflect/Proxy.html#newProxyInstance(java.lang.ClassLoader,%20java.lang.Class%5B%5D,%20java.lang.reflect.InvocationHandler))([ClassLoader](http://docs.google.com/java/lang/ClassLoader.html) loader, [Class](http://docs.google.com/java/lang/Class.html)<?>[] interfaces, [InvocationHandler](http://docs.google.com/java/lang/reflect/InvocationHandler.html) h)            Returns an instance of a proxy class for the specified interfaces that dispatches method invocations to the specified invocation handler. |

| **Methods inherited from class java.lang.**[**Object**](http://docs.google.com/java/lang/Object.html) |
| --- |
| [clone](http://docs.google.com/java/lang/Object.html#clone()), [equals](http://docs.google.com/java/lang/Object.html#equals(java.lang.Object)), [finalize](http://docs.google.com/java/lang/Object.html#finalize()), [getClass](http://docs.google.com/java/lang/Object.html#getClass()), [hashCode](http://docs.google.com/java/lang/Object.html#hashCode()), [notify](http://docs.google.com/java/lang/Object.html#notify()), [notifyAll](http://docs.google.com/java/lang/Object.html#notifyAll()), [toString](http://docs.google.com/java/lang/Object.html#toString()), [wait](http://docs.google.com/java/lang/Object.html#wait()), [wait](http://docs.google.com/java/lang/Object.html#wait(long)), [wait](http://docs.google.com/java/lang/Object.html#wait(long,%20int)) |

| **Field Detail** |
| --- |

### h

protected [InvocationHandler](http://docs.google.com/java/lang/reflect/InvocationHandler.html) **h**

the invocation handler for this proxy instance.

| **Constructor Detail** |
| --- |

### Proxy

protected **Proxy**([InvocationHandler](http://docs.google.com/java/lang/reflect/InvocationHandler.html) h)

Constructs a new Proxy instance from a subclass (typically, a dynamic proxy class) with the specified value for its invocation handler.

**Parameters:**h - the invocation handler for this proxy instance

| **Method Detail** |
| --- |

### getProxyClass

public static [Class](http://docs.google.com/java/lang/Class.html)<?> **getProxyClass**([ClassLoader](http://docs.google.com/java/lang/ClassLoader.html) loader,  
 [Class](http://docs.google.com/java/lang/Class.html)<?>... interfaces)  
 throws [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html)

Returns the java.lang.Class object for a proxy class given a class loader and an array of interfaces. The proxy class will be defined by the specified class loader and will implement all of the supplied interfaces. If a proxy class for the same permutation of interfaces has already been defined by the class loader, then the existing proxy class will be returned; otherwise, a proxy class for those interfaces will be generated dynamically and defined by the class loader.

There are several restrictions on the parameters that may be passed to Proxy.getProxyClass:

* All of the Class objects in the interfaces array must represent interfaces, not classes or primitive types.
* No two elements in the interfaces array may refer to identical Class objects.
* All of the interface types must be visible by name through the specified class loader. In other words, for class loader cl and every interface i, the following expression must be true:  
   Class.forName(i.getName(), false, cl) == i
* All non-public interfaces must be in the same package; otherwise, it would not be possible for the proxy class to implement all of the interfaces, regardless of what package it is defined in.
* For any set of member methods of the specified interfaces that have the same signature:
  + If the return type of any of the methods is a primitive type or void, then all of the methods must have that same return type.
  + Otherwise, one of the methods must have a return type that is assignable to all of the return types of the rest of the methods.
* The resulting proxy class must not exceed any limits imposed on classes by the virtual machine. For example, the VM may limit the number of interfaces that a class may implement to 65535; in that case, the size of the interfaces array must not exceed 65535.

If any of these restrictions are violated, Proxy.getProxyClass will throw an IllegalArgumentException. If the interfaces array argument or any of its elements are null, a NullPointerException will be thrown.

Note that the order of the specified proxy interfaces is significant: two requests for a proxy class with the same combination of interfaces but in a different order will result in two distinct proxy classes.

**Parameters:**loader - the class loader to define the proxy classinterfaces - the list of interfaces for the proxy class to implement **Returns:**a proxy class that is defined in the specified class loader and that implements the specified interfaces **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if any of the restrictions on the parameters that may be passed to getProxyClass are violated [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the interfaces array argument or any of its elements are null

### newProxyInstance

public static [Object](http://docs.google.com/java/lang/Object.html) **newProxyInstance**([ClassLoader](http://docs.google.com/java/lang/ClassLoader.html) loader,  
 [Class](http://docs.google.com/java/lang/Class.html)<?>[] interfaces,  
 [InvocationHandler](http://docs.google.com/java/lang/reflect/InvocationHandler.html) h)  
 throws [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html)

Returns an instance of a proxy class for the specified interfaces that dispatches method invocations to the specified invocation handler. This method is equivalent to:

Proxy.getProxyClass(loader, interfaces).  
 getConstructor(new Class[] { InvocationHandler.class }).  
 newInstance(new Object[] { handler });

Proxy.newProxyInstance throws IllegalArgumentException for the same reasons that Proxy.getProxyClass does.

**Parameters:**loader - the class loader to define the proxy classinterfaces - the list of interfaces for the proxy class to implementh - the invocation handler to dispatch method invocations to **Returns:**a proxy instance with the specified invocation handler of a proxy class that is defined by the specified class loader and that implements the specified interfaces **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if any of the restrictions on the parameters that may be passed to getProxyClass are violated [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the interfaces array argument or any of its elements are null, or if the invocation handler, h, is null

### isProxyClass

public static boolean **isProxyClass**([Class](http://docs.google.com/java/lang/Class.html)<?> cl)

Returns true if and only if the specified class was dynamically generated to be a proxy class using the getProxyClass method or the newProxyInstance method.

The reliability of this method is important for the ability to use it to make security decisions, so its implementation should not just test if the class in question extends Proxy.

**Parameters:**cl - the class to test **Returns:**true if the class is a proxy class and false otherwise **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if cl is null

### getInvocationHandler

public static [InvocationHandler](http://docs.google.com/java/lang/reflect/InvocationHandler.html) **getInvocationHandler**([Object](http://docs.google.com/java/lang/Object.html) proxy)  
 throws [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html)

Returns the invocation handler for the specified proxy instance.

**Parameters:**proxy - the proxy instance to return the invocation handler for **Returns:**the invocation handler for the proxy instance **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if the argument is not a proxy instance

| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/Proxy.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/java/lang/reflect/ParameterizedType.html)   [**NEXT CLASS**](http://docs.google.com/java/lang/reflect/ReflectPermission.html) | [**FRAMES**](http://docs.google.com/index.html?java/lang/reflect/Proxy.html)    [**NO FRAMES**](http://docs.google.com/Proxy.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | [FIELD](#3znysh7) | [CONSTR](#2et92p0) | [METHOD](#tyjcwt) | DETAIL: [FIELD](#1t3h5sf) | [CONSTR](#2s8eyo1) | [METHOD](#3rdcrjn) |

[Submit a bug or feature](http://bugs.sun.com/services/bugreport/index.jsp)

For further API reference and developer documentation, see [Java SE Developer Documentation](http://docs.google.com/webnotes/devdocs-vs-specs.html). That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.

Copyright 2006 Sun Microsystems, Inc. All rights reserved. Use is subject to [license terms](http://docs.google.com/legal/license.html). Also see the [documentation redistribution policy](http://java.sun.com/docs/redist.html).