Proxy objects

1. Proxy objects

A proxy class is a class that implements a list of interfaces specified at compile time. The proxy object typically holds a reference to an internal object that implements the same interfaces (or parts of them). The proxy object is implemented by delegating method calls to the internal object. This is the same principle as implemented by the class `java.lang.reflect.Proxy`, which created proxy objects dynamically at runtime using Java reflection.

Compared to the standard Proxy class, the generator has the obvious disadvantage, that you have to specify the implemented interfaces at runtime. On the other hand, it allows both to select the proxy objects super class and the derivation of subclasses. In fact the derivation of a subclass is much more obvious, simple and faster than the use of an InvocationHandler.

The proxy generator is implemented by the class ProxyGenerator. Use of the ProxyGenerator is demonstrated in the Ant target "generate.proxy".

The proxy generator Ant task supports the following attributes:

Name	Description	Required Default
classpathRef	Reference to a class path, being used to load Java classes or sources. See the "type" attribute in the nested element "implementedInterface" below for details. Use of the "classpathRef" attribute is mutually exclusive with the nested element "classpath".	No Ant's class path
destDir	Path of the directory, where the generated sources are being created. A package structure will be created below. In other words, if a class "org.apache.Foo" is generated, then it will be found in the file	No Current directory

	${destDir}/org/apache/Foo.class.	
extendedClass	Specifies the fully qualified name of a class, which is being extended by the generated sources.	No java.lang.Object

The proxy generator Ant task also supports the following nested elements:

Name	Description	Required Default
classPath	Specifies a path, which is being used for loading Java classes or sources. See the "type" attribute in the nested element "implementedInterface" below for details. The "classpath" element is mutually exclusive with the "classpathRef" attribute.	No Ant's class path
implementedInterface	Specifies an additional interface being implemented. This element must have an attribute "interface" with the fully qualified name of a Java interface which is being implemented by the generated classes. The generator needs to determine the methods specified by the interface. If the element has an attribute "type" with the value "Reflection", then the specified interface will be loaded as a compiled Java class. Otherwise, if the attribute "type" has the value "Source", then the generator attempts to load a Java source file with the same name from the class path and parse it. Otherwise, if the "type" attribute is missing, it will attempt both "Reflection" and "Source", in that order.	Yes, at least one

2. Multiple Inheritance

Multiple inheritance is a design pattern which is not so easy to implement in Java - unless you use the ProxyGenerator. This is demonstrated by the JUnit test MultipleInheritanceTest, which creates a subclass of `java.util.Observable`, that also inherits from `java.util.ArrayList`. The example implements a list, which notifies its observers whenever the `add()` method is invoked.