wo new methods were added to the Collections utility class:

* [newSetFromMap(Map)](http://docs.oracle.com/javase/7/docs/api/java/util/Collections.html#newSetFromMap(java.util.Map)) - creates a general purpose Set implementation from a general purpose Map implementation.

There is no IdentityHashSet class, but instead, just use

Set<Object> identityHashSet=

Collections.newSetFromMap(

new IdentityHashMap<Object, Boolean>());

* [asLifoQueue(Deque)](http://docs.oracle.com/javase/7/docs/api/java/util/Collections.html#asLifoQueue(java.util.Deque)) - returns a view of a Deque as a Last-in-first-out (Lifo) Queue.

The Arrays utility class now has methods [copyOf](http://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html#copyOf(int[],%20int)) and [copyOfRange](http://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html#copyOfRange(int[],%20int,%20int)) that can efficiently resize, truncate, or copy subarrays for arrays of all types.

Before:

int[] newArray = new int[newLength];

System.arraycopy(oldArray, 0, newArray, 0, oldArray.length);

After:

int[] newArray = Arrays.copyOf(a, newLength);

<http://www.oracle.com/technetwork/java/javase/features-141434.html>

## Highlights of Technology Changes in Java SE 6

Choose a technology for links to further information.

* **[Collections Framework](http://download.oracle.com/javase/6/docs/technotes/guides/collections/changes6.html)**
* [**Deployment (Java Web Start and Java Plug-in)**](http://download.oracle.com/javase/6/docs/technotes/guides/deployment/enhancements.html)
* [**Drag and Drop**](http://www.oracle.com/technetwork/articles/javase/javase6-139238.html)
* [**Instrumentation**](http://download.oracle.com/javase/6/docs/technotes/guides/instrumentation/changes6.html)
* [**Internationalization Support**](http://download.oracle.com/javase/6/docs/technotes/guides/intl/index.html)
* [**I/O Support**](http://download.oracle.com/javase/6/docs/technotes/guides/io/enhancements.html#6)
* [**JAR (Java Archive Files) -**](http://download.oracle.com/javase/6/docs/technotes/guides/jar/changes6.html)An annotated list of changes between the 5.0 and 6.0 releases to APIs, the jar command, and the jar/zip implementation.
* **Java Web Start**
* **Java DB 10.2 JDBC4 Early Access**

The $JAVA\_HOME/db subdirectory contains class libraries for Java DB, Sun Microsystems's distribution of the Apache Derby database technology.

* + For information on Java DB, see [http://www.oracle.com/technetwork/java/javadb/](http://www.oracle.com/technetwork/java/javadb/index.html).
  + For information on Derby, see: <http://db.apache.org/derby/>.
  + For documentation on this version of Java DB, see <http://www.oracle.com/technetwork/java/javadb/documentation/index.html>.
  + For a description of the capabilities of this version of Java DB, see the release notes in the $JAVA\_HOME/db subdirectory.
  + This Early Access version of Java DB is built from Apache Derby 10.2.1.7 source code (revision 453926) using JDK 6 JDBC 4 APIs to build JDBC 4 driver code.
  + Comments regarding this version of Java DB can be sent to [javadb-ea@sun.com](mailto:javadb-ea@sun.com).

**DISCLAIMER:** THIS IS EARLY ACCESS SOFTWARE AND COMES WITH NO WARRANTIES OR SUPPORT. IT IS PROVIDED "AS IS". IT IS NOT FOR PRODUCTION USE.

* [**JMX (Java Management Extensions)**](http://download.oracle.com/javase/6/docs/technotes/guides/jmx/enhancements.html)- A list of JMX API changes between the J2SE 5.0 and Java SE 6 releases.
* [**JPDA (Java Platform Debugger Architecture)**](http://download.oracle.com/javase/6/docs/technotes/guides/jpda/enhancements.html)
* [**JVM TI (Java Virtual Machine Tool Interface)**](http://download.oracle.com/javase/6/docs/technotes/guides/jvmti/changes6.html)
* [**lang and util Packages**](http://download.oracle.com/javase/6/docs/technotes/guides/lang/index.html)
* [**Monitoring and Management for the Java Platform**](http://download.oracle.com/javase/6/docs/technotes/guides/management/enhancements.html)
* [**JConsole is Officially Supported in Java SE 6**](http://download.oracle.com/javase/6/docs/technotes/guides/management/enhancements.html#jconsole)
* [**Networking Features**](http://download.oracle.com/javase/6/docs/technotes/guides/net/enhancements-6.0.html)
* [**Performance**](http://download.oracle.com/javase/6/docs/technotes/guides/performance/speed.html)
* [**Reflection**](http://download.oracle.com/javase/6/docs/technotes/guides/reflection/enhancements.html)
* [**RMI (Remote Method Invocation)**](http://download.oracle.com/javase/6/docs/technotes/guides/rmi/relnotes.html)
* [**Scripting**](http://download.oracle.com/javase/6/docs/technotes/guides/scripting/index.html)
* [**Security**](http://download.oracle.com/javase/6/docs/technotes/guides/security/enhancements.html)
* [**Serialization of Objects**](http://download.oracle.com/javase/6/docs/technotes/guides/serialization/relnotes6.html)
* [**Swing**](http://www.oracle.com/technetwork/java/javase/tech/desktop-138473.html)
* [**VM (Java Virtual Machine)**](http://download.oracle.com/javase/6/docs/technotes/guides/vm/index.html#features)

# Java 8 Interface Default Methods

In the previous post we have seen [Lambda Expressions](http://www.java2novice.com/java-8/lambda-expression/). In this post we will see another java 8 new feature "Default Methods" in interfaces. Before Java 8, the interface only contains method signatures. With Java 8 new feature Default Methods or Defender Methods, you can include method body within the interface.

Lets start with an example:

|  |  |  |
| --- | --- | --- |
| [?](http://www.java2novice.com/java-8/interface-default-methods/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8 | package com.java2novice.dm;    public interface FirstInterface {        default void someMethod(){          System.out.println("from FirstInterface...");      }  } | |

Wait, what is **default** here? This is the new way of declaring the method body in Java 8 for an interface. As seen above, default methods comes along with implementation. You dont need to implement default methods if your class implements above interface, below is an example:

|  |  |  |
| --- | --- | --- |
| [?](http://www.java2novice.com/java-8/interface-default-methods/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8 | package com.java2novice.dm;    public class TestMe implements FirstInterface {        public static void main(String[] args) {          new TestMe().someMethod();      }  } | |

We can also override the default methods, but you dont need to specify **default** in the method signature.

|  |  |  |
| --- | --- | --- |
| [?](http://www.java2novice.com/java-8/interface-default-methods/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9 | package com.java2novice.dm;    public class TestMe implements FirstInterface {        @Override      public void someMethod(){          System.out.println("from TestMe class...");      }  } | |

**Why do we need to implement a method within the interface?**

Let's say you have an interface which has multiple methods, and multiple classes are implementing this interface. One of the method implementation can be common across the class, we can make that method as a default method, so that the implementation is common for all classes.

**How to work with existing interfaces?**

Second scenario where you have already existing application, for a new requirements we have to add a method to the existing interface. If we add new method then we need to implement it through out the implementation classes. By using the Java 8 default method we can add a default implementation of that method which resolves the problem.

**When working with multiple inheritance:**

If we have two interfaces, one with default method and another with just method signature (normal way of defining method in the interfaces).

|  |  |  |
| --- | --- | --- |
| [?](http://www.java2novice.com/java-8/interface-default-methods/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9 | public interface FirstInterface {      default void someMethod(){          System.out.println("Am from interface");      }  }    public interface SecondInterface {      void someMethod();  } | |

If any class implements two of the above interfaces, then someMethod() method became abstract method. We have to implement someMethod() method with in the implementation class.

If we have two interfaces, both are having same default method with different implementation, as shown below:

|  |  |  |
| --- | --- | --- |
| [?](http://www.java2novice.com/java-8/interface-default-methods/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11 | public interface FirstInterface {      default void someMethod(){          System.out.println("Am from first interface");      }  }    public interface SecondInterface {      default void someMethod(){          System.out.println("Am from second interface");      }  } | |

If any class implements above two interfaces, there will be an ambiguity. So we have to implement someMethod() method.

If we want to call the perticular default method implementation then you can call like <InterfaceName>.super.<method-name>() as shown below:

|  |  |  |
| --- | --- | --- |
| [?](http://www.java2novice.com/java-8/interface-default-methods/)   |  |  | | --- | --- | | 1  2  3  4  5  6 | public class TestMe implements FirstInterface,SecondInterface {        public void someMethod() {          FirstInterface.super.someMethod();      }  } | |

If we have two interfaces, both are having same default method with different implementation. But one interface extends other as shown below:

|  |  |  |
| --- | --- | --- |
| [?](http://www.java2novice.com/java-8/interface-default-methods/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11 | public interface FirstInterface {      default void someMethod(){          System.out.println("Am from first interface");      }  }    public interface SecondInterface extends FirstInterface{      default void someMethod(){          System.out.println("Am 2 from interface");      }  } | |

Here SecondInterface overrides the someMethod() method from FirstInterface.

**What is the difference between abstract class and interface?**

Abstract class can have constructor, where you need an object to call the methods in subclass. But in case of default method without any reference you can invoke the interface method, like InterfaceName.super.method()

- See more at: <http://www.java2novice.com/java-8/interface-default-methods/#sthash.KtmQAtn0.dpuf>



