Javadoc

Input

```
/**
 * Returns an Image object that can then be painted on the screen.
 * The url argument must specify an absolute {@link URL}. The
name
 * argument is a specifier that is relative to the url argument.
 * <p>
 * This method always returns immediately, whether or not the
 * image exists. When this applet attempts to draw the image on
 * the screen, the data will be loaded. The graphics primitives
 * that draw the image will incrementally paint on the screen.
 *
 * @param url an absolute URL giving the base location of the image
 * @param name the location of the image, relative to the url
argument
 * @return the image at the specified URL
 * @see Image
 */
public Image getImage(URL url, String name) {
   try { return getImage(new URL(url, name));
   } catch (MalformedURLException e) { return null; }
}
```

Output

getImage

public Image **getImage**(URL url, String name)

Returns an Image object that can then be painted on the screen. The url argument must specify an absolute URL. The name argument is a specifier that is relative to the url argument. This method always returns immediately, whether or not the image exists. When this applet attempts to draw the image on the screen, the data will be loaded. The graphics primitives that draw the image will incrementally paint on the screen.

Parameters:

url - an absolute URL giving the base location of the image

name - the location of the image, relative to the url argument

Returns:

the image at the specified URL

See Also:

Image

Tags

Include tags in the following order:

@author (classes and interfaces only, required)
@version (classes and interfaces only, required.)
@param (methods and constructors only)
@return (methods only)
@exception (@throws is a synonym added in Javadoc 1.2)
@see (additional references)
@since (since what version/ since when is it available?)
@serial (or @serialField or @serialData)
@deprecated (why is deprecated, since when, what to use)

Documentation generation

To generate the html documentation, run javadoc followed by the list of source files, which the documentation is to be generated for, in the command prompt (i.e. *javadoc [files]*).

javadoc also provides additional options which can be entered as switches following the javadoc command (i.e. *javadoc [options] [files]*).

javadoc options

Here are some basic javadoc options:

- ***-author*** **- generated documentation will include a author section**
- ***-classpath [path]*** **- specifies path to search for referenced .class files.**
- ***-classpathlist [path];[path];...;[path]*** **- specifies a list locations (separated by ";") to search for referenced .class files.**
- ***-d [path]*** **- specifies where generated documentation will be saved.**
- ***-private*** **- generated documentation will include private fields and methods (only public and protected ones are included by default).**
- ***-sourcepath [path]*** **- specifies path to search for .java files to generate documentation form.**
- ***-sourcepathlist [path];[path];...;[path]*** **- specifies a list locations (separated by ";") to search for .java files to generate documentation form.**
- ***-version*** **- generated documentation will include a version section**

Examples

Basic example that generates and saves documentation to the current directory (c:\MyWork) from A.java and B.java in current directory and all .java files in c:\OtherWork\.

c:\MyWork> javadoc A.java B.java c:\OtherWork*.java

More complex example with the generated documentation showing version information and private members from all .java files in c:\MySource\ and c:\YourSource\ which references files in c:\MyLib and saves it to c:\MyDoc.

c:\> javadoc -version -private -d c:\MyDoc
-sourcepathlist c:\MySource;c:\YourSource
-classpath c:\MyLib

More info

http://java.sun.com/j2se/javadoc/writingdoccomments/index.html

The javadoc tool does not directly document anonymous classes -- that is, their declarations and doc comments are ignored. If you want to document an anonymous class, the proper way to do so is in a doc comment of its outer class

Clonazione

Clonazione

La clonazione...

Ovvero: come costruire una copia

(probabilmente che ritorni true su equals?)

Metodo clone di Object

protected Object clone()
throws CloneNotSupportedException

Creates and returns a copy of this object. The precise meaning of "copy" may depend on the class of the object.

The general intent is that, for any object x,

- the expression: x.clone() != x will be true,
- and that the expression: x.clone().getClass() == x.getClass() will be true, but these are not absolute requirements.
- While it is typically the case that: x.clone().equals(x) will be true, this is not an absolute requirement.

```
public class Test {
  public static void main(String []a){new Test();}

  Test() {
     P p1=new P();
     p1.x=1;
     p1.y=2;

     P p2=p1.clone(); // NO! Metodo protected!
     System.out.println(p2);
  }
}
```

```
class P {
  int x; int y;
  public String toString() {
     return ("x="+x+" ; y="+y);
  }
}
```

clone per la classe P

```
class P implements Cloneable {
…
  public Object clone(){
     try {
       return super.clone();
     } catch (CloneNotSupportedException e) {
       System.err.println("Implementation error");
       System.exit(1);
     }
     return null; //qui non arriva mai  }
  }
}
```

Copia bit a bit

```
public class Test {
  public static void main(String []a){new Test();}
  Test() {
     P p1=new P(); p1.x=5; p1.y=6;
     P p2=p1;
     P p3=p1.clone();
     System.out.println(p1);
     System.out.println(p2);
     System.out.println(p3);
     p1.x=7
     System.out.println(p1);
     System.out.println(p2);
     System.out.println(p3);
  }
}
```

Main di test

```
x=5 ; y=6
x=5 ; y=6
x=5 ; y=6

x=7 ; y=6
x=7 ; y=6
x=5 ; y=6
```

Class V

```
class V  implements Cloneable   {
  int x[];
  V(int s) {
    x=new int[s];
    for (int k=0;k<x.length;k++) x[k]=k;
  }
  public String toString() {
    String s="";
    for (int k:i;) s=s+x[k]+" ";
    return s;
  }
 ... // clone definito come prima
}
```

Main di test

```
public class Test {
  public static void main(String []a){new Test();}

  Test() {
     V p1=new V(5);
     V p2=p1.clone();
     System.out.println(p1);
     System.out.println(p2);
     p1.x[0]=9;
     System.out.println(p1);
     System.out.println(p2);
  }
}
```

```
0 1 2 3 4
0 1 2 3 4

9 1 2 3 4
9 1 2 3 4
```

```
class V  implements Cloneable   {
  int x[];V(int s){…}public String toString(){…}
  public Object clone(){
    Object tmp=null;
    try {
      tmp=super.clone();
    } catch (CloneNotSupportedException e) {
      e.printStackTrace(); return null;
    }
    ((V)tmp).x=new int[x.length];
    for (int k:x)((V)tmp).x[k]=x[k];
    return tmp;
  }
}
```

Main di test

```
public class Test {
  public static void main(String []a){new Test();}

  Test() {
     V p1=new V(5);
     V p2=p1.clone();
     System.out.println(p1);
     System.out.println(p2);
     p1.x[0]=9;
     System.out.println(p1);
     System.out.println(p2);
  }
}
```

```
0 1 2 3 4
0 1 2 3 4
9 1 2 3 4
0 1 2 3 4
```

Shallow vs. Deep copy

super.clone()

Effettua una SHALLOW COPY

Per ottenere una DEEP COPY occorre modificane il risultato.

Ogni volta che ho delle referenze tra le variabili di istanza, devo chiedermi se voglio fare una copia della referenza o dell'oggetto!