Automatic wrapper generator

The purpose of this document is to describe the automatic wrapper generator for developers of applications for the SLK/Jigsaw system at Cornell. The generator described here allows for easy creation of three types of wrappers: callee-protected (CP) wrappers, caller-callee protected (CCP) wrappers and thread switch (TS) wrappers. All three kinds are described below, together with the way the generator is used.

Obtaining the code

The code is available from SourceSafe (go to Java, then Jigsaw, then wrgen).

Copyright and such

The code is based on the Toba system, from the Univerity of Arizona. In particular, we have used their code to parse the class file format. The wrapper generator as it is now is not to be distributed outside of Cornell. When the right time comes, we'll worry about the proper headers, disclaimers etc. Now the code is still "under development".

Problems

In case of bugs and other problems contact grzes@cs.cornell.edu. Also, since the generator has been tested on a very limited number of classes only, it may turn out that enhancements and changes are necessary. Send such comments as well.

Running the program

The program is invoked as follows:

java Wrgen <class-file> <wrapper-type> [-nM/-yM list-of methods] [-nI/-yI list-of-interfaces]

where class-file is a name of a Java class file, wrapper type is one of: CP, CCP and TS, and list-of-methods is a list of method names. Specifying -nM means that no methods in the wrapper will be generated for listed methods; -yM means that only the specified method names will have their counterparts in the wrapper class. The default is that all public methods of the class are taken care of. Similarly, one can request some interfaces to be dropped from the list of interfaces the wrapper will implement (option -nI) or request that only some interfaces be implemented by the wrapper (option -yI). Being able to control interfaces implemented by wrappers is important, since dropping methods may

cause the wrapper not to implement all interfaces of wrapped class.

Output

The program produces an approproate wrapper source code, sent to standard output. The name of the resulting class is a concatenation of the name of the original class and a wrapper type.

Remarks

Although the types of supported wrappers differ in many respects (describel later in this document), there is a number of features that all wrappers share, as listed below..

Interfaces

The wrapper generator considers all interfaces of a given class, unless requested otherwise via the -nI/-yI option.

Methods

The wrapper generator considers all public methods of a given class, unless requested otherwise by the -nM/-yM option.

Fields

No fields from a wrapped class appear in its wrappers, even if the fields are public.

Constructors

Exacly one constructor (a "copying constructor") is generated for every wrapper class. Its function is to assign its parameter (of the type of wrapped class) to a private fileld.

Exceptions

Every method in the wrapper is declared to throw all exceptions declared in the "throw" clause of the corresponding "wrapped" method.

Inheriting method modifiers

The following method modifiers are present in a method of the wrapper if and only if they are present in the "wrapped" method: **synchronized, abstract, final, static**. However, the TS wrapper type cannot implement static wrapping methods.

Class path

Your *classpath* environment variable must contain a directory name with unpacked base java classes

(from java, io, util and net packages). Directories containing superclasses of the class for which a wrapper is being generated need to appear in classpath as well. This is necessary for traversing inheritance chains.

Wrapper types

The current version of the wrapper generator supports currently three kinds of wrappers, described below. The description is followed by several simple examples.

Callee-protected (CP) wrapper

It is the simplest kind of wrapper. Every method appearing in the wrapper makes a direct call to the appropriate method of the wrapped object. The CP wrappers are useful for two reasons:

- by omitting some methods, a copy of the same object with different list of accessible methods can be given out to different users
- a CP wrapper can be used as a basic block to which more code is added before calling methods of the wrapped object and after return from them. Examples include adding statistics, accounting and verification code.

Caller/callee protected (CCP) wrapper

This wrapper type is used when caller's threads are of the type SlkThread**.** Before caling a method on a wrapped object, the caller's thread takes on temporarily (i.e. until the return from the call) null rights - this ensures that the callee cannot perform certain operations on the thread. Similar advantages as in the case of CP wrappers apply here as well.

Thread switch (TS) wrapper

This wrapper suspends the caller's thread until a wrapper's thread services the request. This ensures that neither the caller can do any damage to the callee (for example, by killing the servicing thread and leaving the wrapped object in an inconsistent state) nor that the callee can harm the caller (for example, by storing the caller's thread Thread object and manipulating it).

Examples

The usage of wrapper generator is illustrated on a simple example that uses two interfaces and two classes shown below:

```
interface If1 {
    public Object foo1(Object o, int i) throws IOException, ClassNotFoundException ;
}

public interface If2 {
    public void foo2(Vector v, Hashtable h);
}
```

```
public class Cl1 implements If1 {

    public Object foo1(Object o, int i) throws IOException, ClassNotFoundException {
        return null;
    }
    public static synchronized void bar() {}
}

public class Cl2 extends Cl1 implements If2 {
    public void foo2(Vector v, Hashtable h) {
    }
    public static int fun(int a, int b, int c) { return a+b+c; }
}
```

The CP wrapper for objects of class Cl2 with removed fun() method has been generated with the following command: java Wrgen Cl2 CP -n fun. The code obtained is listed below.

```
public class Cl2CPWrapper implements If2, If1
{

   private Cl2 o;

   public void foo2(java.util.Vector arg1, java.util.Hashtable arg2)
   {
      o.foo2(arg1, arg2);
   }

   public java.lang.Object foo1(java.lang.Object arg1, int arg2)
      throws java.io.IOException,java.lang.ClassNotFoundException
   {
      java.lang.Object retVal = o.foo1(arg1, arg2);
      return retVal;
   }

   public static synchronized void bar()
   {
      Cl2.bar();
   }

   public int hashCode()
   {
      int retVal = o.hashCode();
      return retVal;
   }

   public boolean equals(java.lang.Object arg1)
   {
      boolean retVal = o.equals(arg1);
      return retVal;
   }

   public java.lang.String toString()
   {
      java.lang.String retVal = o.toString();
      return retVal;
   }

   public Cl2CPWrapper(Cl2 o)
   {
      this.o = o;
```

```
   }
}
```

The CPP wrapper for the Cl2 class, also with fun() removed, looks like this:

```
import java.servlet.*;
import java.io.*;

public class Cl2CCPWrapper implements If2, If1
{

   private Cl2 o;

   Cl2CCPWrapper(Cl2 o)
   {
      this.o = o;
   }

   public void foo2(java.util.Vector arg1, java.util.Hashtable arg2)
   {
      SlkThread currentThread = (SlkThread) Thread.currentThread();
      ThreadRights prevRights = currentThread.currentRights;

      try {
         synchronized (currentThread.switchLock) {
            currentThread.currentRights = null;
         }
         o.foo2(arg1, arg2);
      }
      finally {
         synchronized (currentThread.switchLock) {
            currentThread.currentRights = prevRights;
         }
      }
   }

   public java.lang.Object foo1(java.lang.Object arg1, int arg2)
      throws java.io.IOException,java.lang.ClassNotFoundException
   {
      SlkThread currentThread = (SlkThread) Thread.currentThread();
      ThreadRights prevRights = currentThread.currentRights;

      try {
         synchronized (currentThread.switchLock) {
            currentThread.currentRights = null;
         }
         java.lang.Object retVal = o.foo1(arg1, arg2);
         return retVal;
      }
      finally {
         synchronized (currentThread.switchLock) {
            currentThread.currentRights = prevRights;
         }
      }
   }

   public static synchronized void bar()
   {
      SlkThread currentThread = (SlkThread) Thread.currentThread();
      ThreadRights prevRights = currentThread.currentRights;
```

```
   try {
      synchronized (currentThread.switchLock) {
         currentThread.currentRights = null;
      }
      Cl2.bar();
   }
   finally {
      synchronized (currentThread.switchLock) {
         currentThread.currentRights = prevRights;
      }
   }
}

public int hashCode()
{
   SlkThread currentThread = (SlkThread) Thread.currentThread();
   ThreadRights prevRights = currentThread.currentRights;

   try {
      synchronized (currentThread.switchLock) {
         currentThread.currentRights = null;
      }
      int retVal = o.hashCode();
      return retVal;
   }
   finally {
      synchronized (currentThread.switchLock) {
         currentThread.currentRights = prevRights;
      }
   }
}

public boolean equals(java.lang.Object arg1)
{
   SlkThread currentThread = (SlkThread) Thread.currentThread();
   ThreadRights prevRights = currentThread.currentRights;

   try {
      synchronized (currentThread.switchLock) {
         currentThread.currentRights = null;
      }
      boolean retVal = o.equals(arg1);
      return retVal;
   }
   finally {
      synchronized (currentThread.switchLock) {
         currentThread.currentRights = prevRights;
      }
   }
}

public java.lang.String toString()
{
   SlkThread currentThread = (SlkThread) Thread.currentThread();
   ThreadRights prevRights = currentThread.currentRights;

   try {
      synchronized (currentThread.switchLock) {
         currentThread.currentRights = null;
      }
      java.lang.String retVal = o.toString();
```

```
         return retVal;
      }
      finally {
         synchronized (currentThread.switchLock) {
            currentThread.currentRights = prevRights;
         }
      }
   }

}
```

Finally, the TS wrapper for Cl2 without foo() is listed below. Note that since static methods cannot be wrapped by TS wrappers, bar had to be removed from the allowed methods. The resulting command line was: java Wrgen TS -n fun bar.

```
import java.servlet.*;
import java.io.*;


public class Cl2TSWrapper implements Runnable, If2, If1{
   private Cl2 o;

   public Cl2TSWrapper(Cl2 o)
   {
      this.o = o;

      if (workerThread == false) {
         Thread t = new Thread(this);
         workerThread = true;
         t.start();
         Thread.currentThread().yield();
      }
   }

   private static boolean workerThread = false;
   private boolean done = false;
   private int whichCall = -1;
   private final Object sa = new Object();
   private final Object sb = new Object();

   private final int foo2_CALL = 0;
   private java.util.Vector foo2_ARG1;
   private java.util.Hashtable foo2_ARG2;
   public  synchronized void foo2(java.util.Vector arg1, java.util.Hashtable arg2)
   {
      try {
         synchronized (sa) {
            done = false;
            foo2_ARG1 = arg1;
            foo2_ARG2 = arg2;
            whichCall = foo2_CALL;
            sa.notify();
         }
         synchronized (sb) { while (!done) sb.wait(); }
      }
      catch (InterruptedException ie) {
         ie.printStackTrace();
      }
   }
```

```
private final int foo1_CALL = 3;
private java.lang.Object foo1_ARG1;
private int foo1_ARG2;
private java.lang.Object foo1_RET;
private static java.io.IOException foo1java_io_IOException = null;
private static java.lang.ClassNotFoundException foo1java_lang_ClassNotFoundExcept
public  synchronized java.lang.Object foo1(java.lang.Object arg1, int arg2)
   throws java.io.IOException,java.lang.ClassNotFoundException
{
   try {
      synchronized (sa) {
         done = false;
         foo1_ARG1 = arg1;
         foo1_ARG2 = arg2;
         foo1java_io_IOException = null;
         foo1java_lang_ClassNotFoundException = null;
         whichCall = foo1_CALL;
         sa.notify();
      }
      synchronized (sb) { while (!done) sb.wait(); }
   }
   catch (InterruptedException ie) {
      ie.printStackTrace();
   }
   if (foo1java_io_IOException != null) throw foo1java_io_IOException;
   if (foo1java_lang_ClassNotFoundException != null) throw foo1java_lang_ClassNot
   return foo1_RET;
}

private final int hashCode_CALL = 5;
private int hashCode_RET;
public  synchronized int hashCode()
{
   try {
      synchronized (sa) {
         done = false;
         whichCall = hashCode_CALL;
         sa.notify();
      }
      synchronized (sb) { while (!done) sb.wait(); }
   }
   catch (InterruptedException ie) {
      ie.printStackTrace();
   }
   return hashCode_RET;
}

private final int equals_CALL = 6;
private java.lang.Object equals_ARG1;
private boolean equals_RET;
public  synchronized boolean equals(java.lang.Object arg1)
{
   try {
      synchronized (sa) {
         done = false;
         equals_ARG1 = arg1;
         whichCall = equals_CALL;
         sa.notify();
      }
      synchronized (sb) { while (!done) sb.wait(); }
   }
```

```
   catch (InterruptedException ie) {
      ie.printStackTrace();
   }
   return equals_RET;
}

private final int toString_CALL = 7;
private java.lang.String toString_RET;
public  synchronized java.lang.String toString()
{
   try {
      synchronized (sa) {
         done = false;
         whichCall = toString_CALL;
         sa.notify();
      }
      synchronized (sb) { while (!done) sb.wait(); }
   }
   catch (InterruptedException ie) {
      ie.printStackTrace();
   }
   return toString_RET;
}

public void run()
{
   try {
      synchronized (sa) {
         while (true) {
            sa.wait();
            switch(whichCall) {
               case foo2_CALL:
                  o.foo2(foo2_ARG1, foo2_ARG2);
                  break;

               case foo1_CALL:
                  try {
                     foo1_RET = o.foo1(foo1_ARG1, foo1_ARG2);
                  }
                  catch (java.io.IOException e1) {
                     foo1java_io_IOException = e1;
                  }
                  catch (java.lang.ClassNotFoundException e2) {
                     foo1java_lang_ClassNotFoundException = e2;
                  }
                  break;

               case hashCode_CALL:
                  hashCode_RET = o.hashCode();
                  break;

               case equals_CALL:
                  equals_RET = o.equals(equals_ARG1);
                  break;

               case toString_CALL:
                  toString_RET = o.toString();
                  break;

               default:
                  System.err.println("Wrong method code in Cl2TSWrapper: " + whic
```

```
                    break;
                }
                done = true;
                synchronized (sb) { sb.notify(); }
            }
         }
      }
      catch (InterruptedException ie) {
         ie.printStackTrace();
      }
   }
}
```