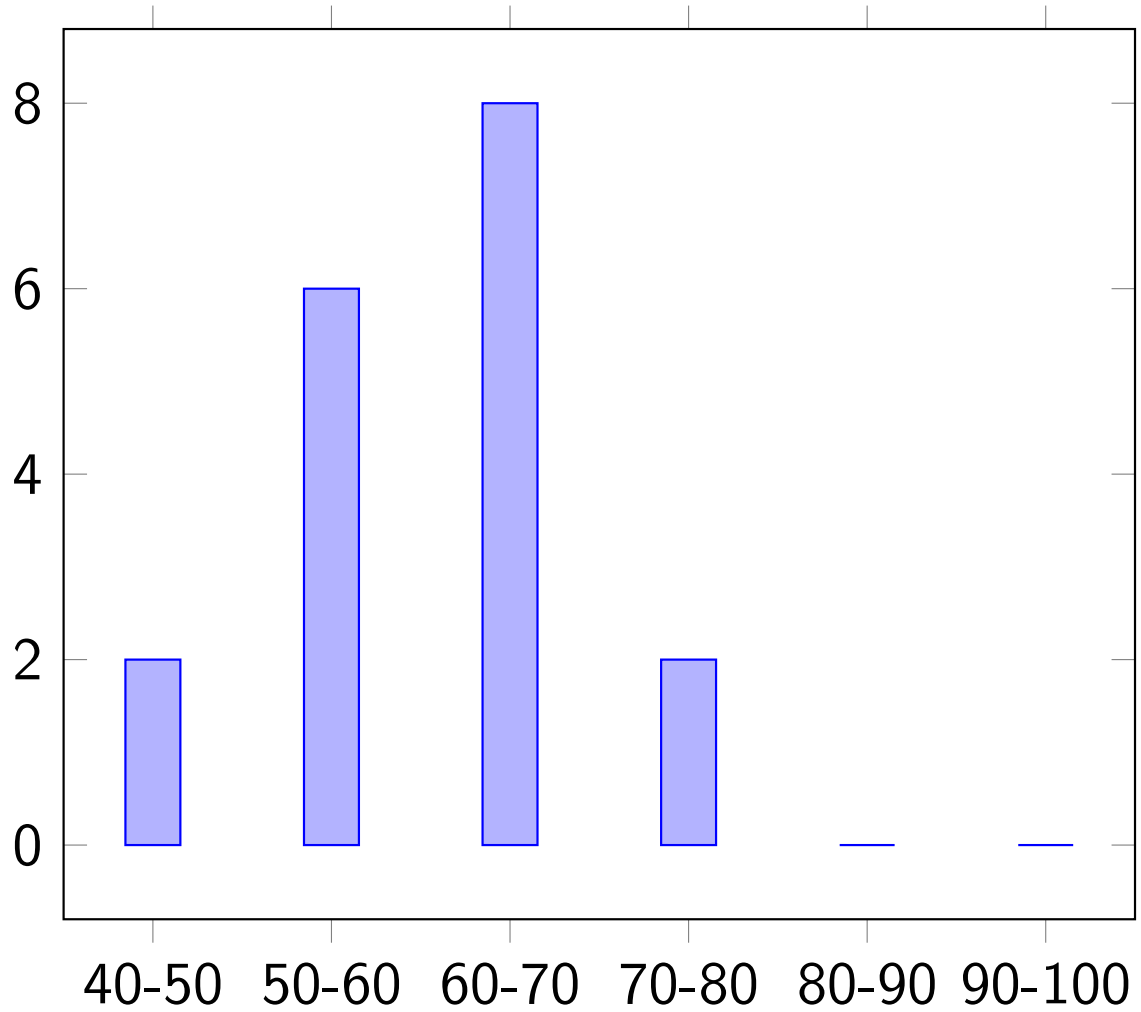
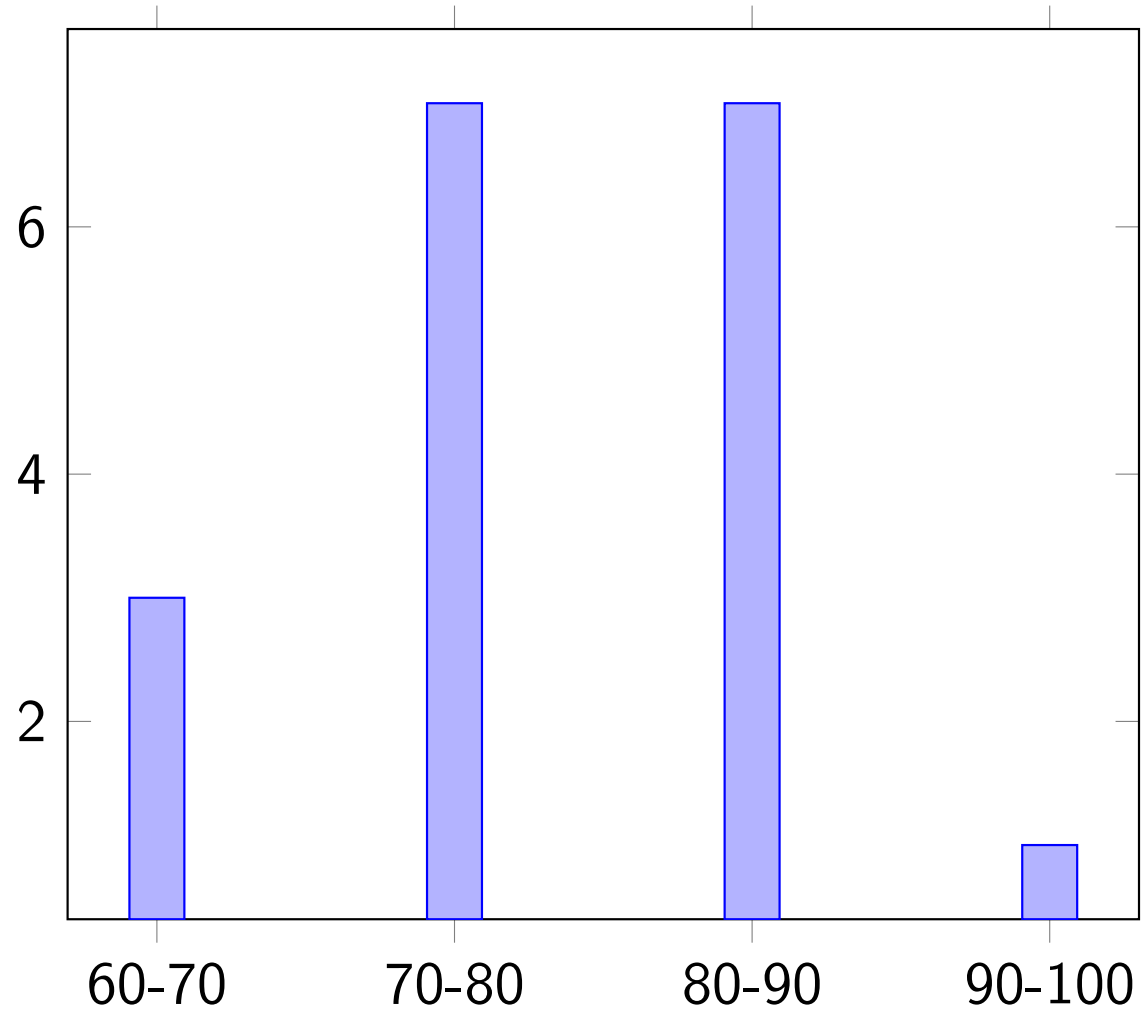

Rhymes With Orange
THE PROGRAM
She's a pocketful
Facebook.com/RhymesComic
HOW'S THE EXCHANGE STUDENT?
©2018 RWO STUDIOS DIST. BY KING FEATURES
Piccolo & HILARY B. PRICE 2018 RhymesWithOrange.com

Midterm: grade distribution



Average: 61%

Overall: grade distribution



Average: 79%

Peers and Native Peers

EECS 4315

www.eecs.yorku.ca/course/4315/

Java

All about Java
(we will get back to model checking and JPF later)

TeleTYpewriter

Linux command:

```
tty - print the file name of the terminal connected to
      standard input
```

Output:

```
% tty
/dev/pts/8
```

"Early user terminals connected to computers were electromechanical teleprinters or teletypewriters (TeleTYpewriter, TTY), and since then TTY has continued to be used as the name for the text-only console although now this text-only console is a virtual console not a physical console."[1]

[1] askubuntu.com/questions/481906/what-does-tty-stand-for

TeleTYpewriter

Problem

Write a Java app that checks if the Java virtual machine is connected to a terminal device. If it is, print "TTY: " followed by the name of the device, otherwise print "Not a TTY." Use the `my.util.TeleTYpewriter` class whose API can be found here.

TeleTYpewriter

```
import my.util.TeleTYpewriter;

public class Main {
  public static void main(String[] args) {
    if (TeleTYpewriter.isTTY()) {
      System.out.println("TTY: " +
                         TeleTYpewriter.getTTYName());
    } else {
      System.out.println("Not a TTY");
    }
  }
}
```

Step 1: write Java class

```
package my.util;

public class TeleTYpewriter {
  private TeleTYpewriter() {}

  static {
    System.loadLibrary("ttyutil");
  }

  public static native boolean isTTY();
  public static native String getTTYName();
}
```

Question

Why does the class contain a private constructor?

Question

Why does the class contain a private constructor?

Answer

So that we cannot create an instance of the class. Also, no default (public) constructor is added and, hence, no constructor appears in the API.

```
public static void loadLibrary(String libname)
```

Loads the system library specified by the libname argument.

```
public static void loadLibrary(String libname)
```

Loads the system library specified by the libname argument.

The method call `System.loadLibrary("ttyutil");`

- In Linux: loads the shared object file libttyutil.so.
- In Windows: loads the dynamic link library file ttyutil.dll.

```
public static native boolean isTTY();
```

Question

What is a native method?

```
public static native boolean isTTY();
```

Question

What is a native method?

Answer

A method that is implemented in a language other than Java but that is invoked from a Java app.

Native methods

Question

Why are there native methods?

Native methods

Question

Why are there native methods?

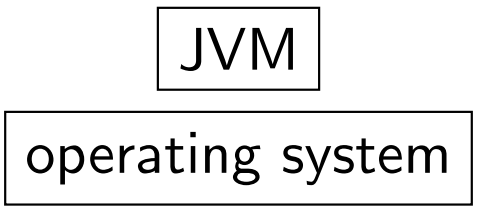
Answer

- Allows programmers to use code that has already been implemented in other languages.
- May increase the performance.
- May support certain platform-dependent features.

Many of the classes of the Java standard library include native methods.

Java native interface (JNI)

JNI provides the infrastructure for Java code to use libraries written in other languages such as C, C++ and assembly.



Invoking a native method can be viewed as transferring the execution from the JVM to the operating system, since the native code will be executed outside the JVM and will run on the operating system.

Sheng Liang. *Java Native Interface: Programmer's Guide and Specification*. Prentice Hall. 1999.

Step 2: compile Java class and generate header file

```
javac -h <folder of header file> <java class>
```

compiles the class and generates a header file.

```
javac -h . TeleTYpewriter.java
```

compiles the `TeleTYpewriter` class and generates the following header file.

Header file

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class my_util_TeleTYpewriter */

#ifndef _Included_my_util_TeleTYpewriter
#define _Included_my_util_TeleTYpewriter
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     my_util_TeleTYpewriter
 * Method:    isTTY
 * Signature: ()Z
 */
JNIEXPORT jboolean JNICALL
  Java_my_util_TeleTYpewriter_isTTY(JNIEnv *, jclass);
```

Header file (continued)

```
/*
 * Class:     my_util_TeleTYpewriter
 * Method:    getTTYName
 * Signature: ()Ljava/lang/String;
 */
JNIEXPORT jstring JNICALL
  Java_my_util_TeleTYpewriter_getTTYName(JNIEnv *, jclass);

#ifdef __cplusplus
}
#endif
#endif
```

jni.h

The header file `jni.h` is distributed with the JDK and defines `JNIEXPORT`, `JNICALL`, `jboolean`, `jstring`, `JNIEnv`, and `jclass`.

- `JNIEXPORT`: macro
- `JNICALL`: macro
- `JNIEnv`: JNI environment, provides access to all the JNI functions

Types

Java type	native type
int	jint
double	jdouble
boolean	jboolean
String	jstring
Object	jobject
Class	jclass

Function declaration

```
JNIEXPORT jboolean JNICALL
  macro
  Java_my_util_TeleTYpewriter_isTTY(JNIEnv *, jclass);
```

Function declaration

```
JNIEXPORT   jboolean   JNICALL
            ^^^^^^^
            return type
  Java_my_util_TeleTYpewriter_isTTY(JNIEnv *, jclass);
```

Function declaration

```
JNIEXPORT jboolean JNICALL
                   ^^^^^^^
                   macro
  Java_my_util_TeleTYpewriter_isTTY(JNIEnv *, jclass);
```

Function declaration

```
JNIEXPORT jboolean JNICALL
Java_   my_util   _TeleTYpewriter_isTTY(JNIEnv *, jclass);
```

package name

Function declaration

```
JNIEXPORT jboolean JNICALL
  Java_my_util_TeleTYpewriter_isTTY(JNIEnv *, jclass);
```

class name

Function declaration

```
JNIEXPORT jboolean JNICALL
  Java_my_util_TeleTYpewriter_   isTTY   (JNIEnv *, jclass)
```

method name

Function declaration

```
JNIEXPORT jboolean JNICALL
  Java_my_util_TeleTYpewriter_isTTY(    JNIEnv *    , jclass
```

JNI environment

Function declaration

```
JNIEXPORT jboolean JNICALL
  Java_my_util_TeleTYpewriter_isTTY(JNIEnv *, jclass);
                                              ^^^^^^
                                              class
```

The class on which the static method is invoked.

Step 3: write C code

```
/* header file generated from my.util.TeleTYpewriter.java *
#include "my_util_TeleTYpewriter.h"
/* header file that provides access to the POSIX operating
#include <unistd.h>

JNIEXPORT jboolean JNICALL
  Java_my_util_TeleTYpewriter_isTTY(JNIEnv *env, jclass cls
    return isatty(STDOUT_FILENO)? JNI_TRUE: JNI_FALSE;
}

JNIEXPORT jstring JNICALL
  Java_my_util_TeleTYpewriter_getTTYName(JNIEnv *env, jclas
    char *name = ttyname(STDOUT_FILENO);
    return (*env)->NewStringUTF(env, name);
}
```

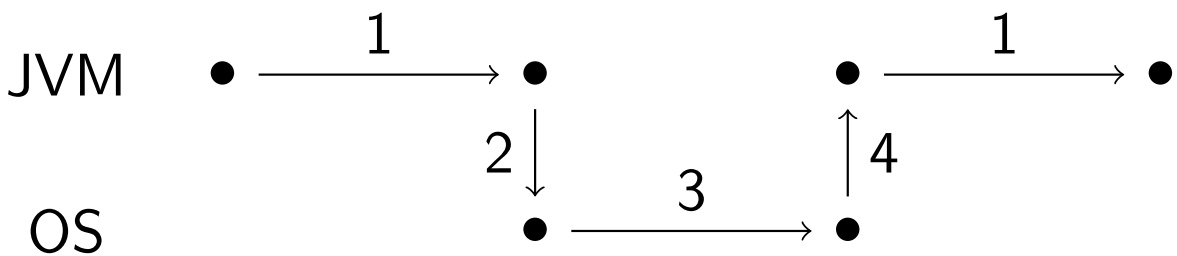
Executing a native method

Transferring the execution from the JVM to the operating system.

1. JVM: convert Java values (parameters) to native values.
2. Operating system: execute the C function.
3. JVM: convert native value (return) to Java value.

The Java_my_util_TeleTYpewriter_isTTY function

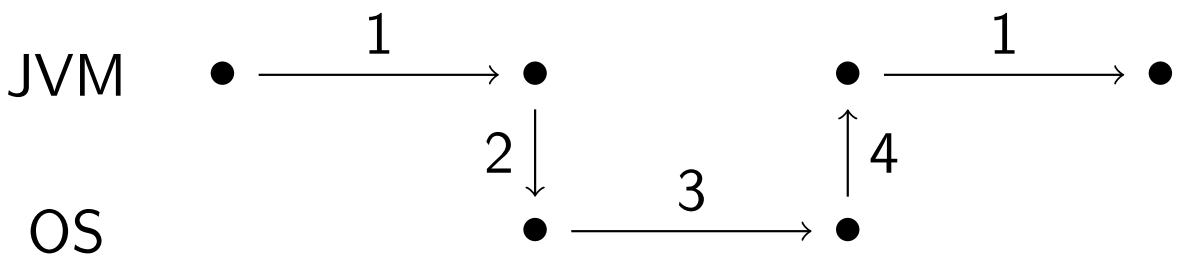
```
JNIEXPORT jboolean JNICALL
  Java_my_util_TeleTYpewriter_isTTY(JNIEnv *env, jclass cls
    return isatty(STDOUT_FILENO)? JNI_TRUE : JNI_FALSE;
}
```



1. Execution of `Main.main`.
2. Conversion of `TeleTYpewriter.class` to a `jclass` value.
3. Execution of `Java_my_util_TeleTYpewriter_isTTY`.
4. Conversion of `jboolean` value to a `boolean` value.

The Java_my_util_TeleTYpewriter_getTTYName function

```
JNIEXPORT jstring JNICALL
  Java_my_util_TeleTYpewriter_getTTYName(JNIEnv *env, jclas
    char *name = ttyname(STDOUT_FILENO);
    return (*env)->NewStringUTF(env, name);
}
```



1. Execution of `Main.main`.
2. Conversion of `TeleTYpewriter.class` to a `jclass` value.
3. Execution of `Java_my_util_TeleTYpewriter_getTTYName`.
4. Conversion of `jstring` value to a `String` value.

Step 4: compile C code and generate native library file

- Linux:

```
gcc -fPIC -I <folder containing jni.h> \
  -I <folder containing jni_md.h> -shared \
  -o libttyutil.so my_util_TeleTYpewriter.c
```

  On the EECS system, `jni.h` resides in `/cs/local/pkg/jdk-1.8.0_202/include` and `jni_md.h` can be found in `/cs/local/pkg/jdk-1.8.0_202/include/linux`.

- Windows:

```
gcc -I <folder containing jni.h> \
  -I <folder containing jni_md.h> -shared \
  -o ttyutil.dll my_util_TeleTYpewriter.c
```

Step 5: run Java app

To run the Java app, we need to add the native library to Java's library path. That is, we have to set the Java property `java.library.path`, which captures the folders with native libraries. This can be accomplished as follows.

```
java -Djava.library.path=<folder of file> <name of app>
```

To run the `Main` app, use

```
java -Djava.library.path=<folder of libttyutil.so> Main
```

Back to model checking

Applying JPF

To run JPF on the `Main` app, we create the following application properties file.

```
target=Main
classpath=.
```

Applying JPF

```
========================================================== erro
gov.nasa.jpf.vm.NoUncaughtExceptionsProperty
java.lang.UnsatisfiedLinkError: cannot find native my.util.
  at my.util.TeleTYpewriter.isTTY(no peer)
  at Main.main(Main.java:5)
```

Applying JPF

Question

Why does JPF report the following error?

```
gov.nasa.jpf.vm.NoUncaughtExceptionsProperty
java.lang.UnsatisfiedLinkError: cannot find native my.util.
  at my.util.TeleTYpewriter.isTTY(no peer)
  at Main.main(Main.java:5)
```

Applying JPF

Question

Why does JPF report the following error?

```
gov.nasa.jpf.vm.NoUncaughtExceptionsProperty
java.lang.UnsatisfiedLinkError: cannot find native my.util.
  at my.util.TeleTYpewriter.isTTY(no peer)
  at Main.main(Main.java:5)
```

Answer

Because the isTTY method is native.

```
public static native boolean isTTY();
```

Handling native methods

JPF provides several ways to handle native methods.

- Using peers (also known as model classes).
- Using native peers.
- Using the extension jpf-nhandler.

Handling native methods

First approach

Peers

A peer class captures the behaviour of a native method in pure Java.

Question

How can we capture the behaviour of the `isTTY` method?

Peers

A peer class captures the behaviour of a native method in pure Java.

Question

How can we capture the behaviour of the `isTTY` method?

Answer

For example, randomly returning `true` or `false`.

Peer class

```
package my.util;

import java.util.Random;

public class TeleTYpewriter {
 public static boolean isTTY() {
   System.out.println("Modelling isTTY");
   Random random = new Random();
   return random.nextBoolean();
  }
}
```

Peers

A peer class captures the behaviour of a native method in pure Java.

Question

How can we capture the behaviour of the `getTTYName` method?

Peers

A peer class captures the behaviour of a native method in pure Java.

Question

How can we capture the behaviour of the `getTTYName` method?

Answer

For example, returning the string "name".

Peer class

```
package my.util;

public class TeleTYpewriter {
  ...

  public static String getTTYName() {
    System.out.println("Modelling getTTYName");
    return "name";
  }
}
```

Peer class

- The peer class `TeleTYpewriter` is part of the package `my.util`.
- The peer class need not contain all methods of the original `TeleTYpewriter` class.

Peer classes

To ensure that JPF verifies the peer class, rather than the original class, we need to add the peer class to JPF's classpath.

```
target=Main
classpath=<folder containing the folder my of the peer clas
cg.enumerate_random=true
```

The file `TeleTYpewriter.class` is contained in a folder named `util` which is contained in a folder named `my`.

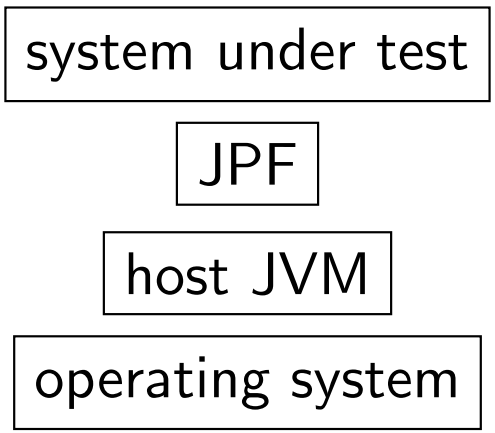
```
...
========================================================== sea
Modelling isTTY
Not a TTY
Modelling getTTYName
TTY: name

========================================================== resu
no errors detected
...
```

Handling native methods

Second approach

Native peers

Native peers are the JPF analogue of native code.



Native peers are executed by the host JVM (recall that peers are executed by JPF).

Model Java interface (MJI)

MJI is the JPF analogue of JNI. It provides the infrastructure for interaction between JPF and the host JVM when executing a native peer.

Naming

MJI uses a specific name pattern to establish the correspondence between the original class (containing the native method) and its native peer, similar to JNI.

```
package my.util;

public class TeleTYpewriter
```

corresponds to

```
public class JPF_my_util_TeleTYpewriter
  extends NativePeer
```

Naming

Question

What is the MJI counterpart of

```
package java.lang;

public class Boolean
```

Naming

Question

What is the MJI counterpart of

```
package java.lang;

public class Boolean
```

Answer

```
public class JPF_java_lang_Boolean
  extends NativePeer
```

Naming

MJI uses a specific name pattern to establish the correspondence between the original class (containing the native method) and its native peer, similar to JNI.

`https://docs.oracle.com/javase/8/docs/technotes/guides/jni/spec/design.html`

```
public static native boolean isTTY();
```

corresponds to

```
@MJI
public boolean isTTY____Z(MJIEnv env, int clsObjRef)
```

Naming

boolean	Z
byte	B
char	C
short	S
int	I
long	L
float	F
double	D

Naming

Question

What is the MJI counterpart of

```
public static Boolean valueOf(boolean b)
```

Naming

Question

What is the MJI counterpart of

```
public static Boolean valueOf(boolean b)
```

Answer

```
@MJI
public int valueOf__Z__Ljava_lang_Boolean_2
  (MJIEnv env, int clsObjRef, boolean b)
```

Naming

Question

What is the MJI counterpart of

```
public abstract boolean compareAndSet(T obj,
  int expect, int update)
```

Naming

Question

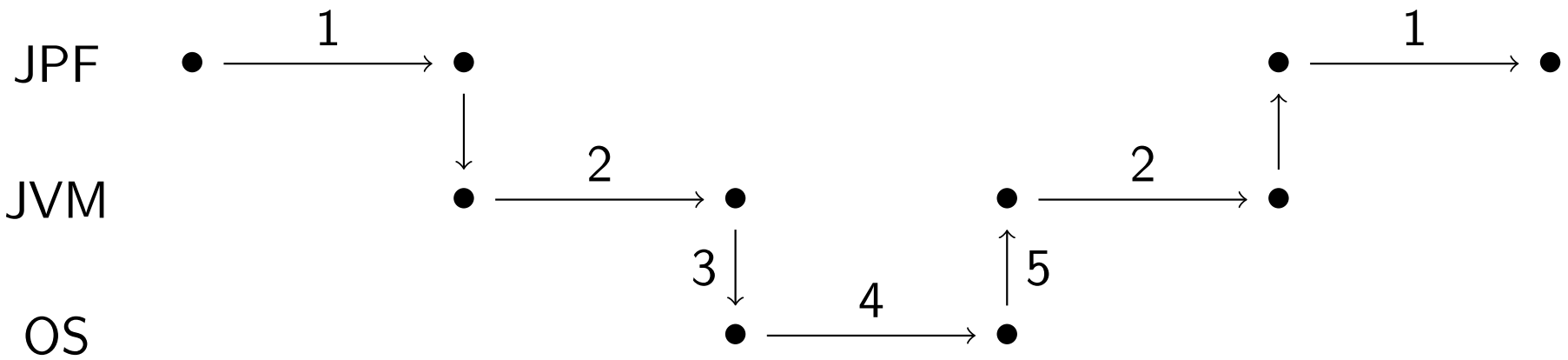
What is the MJI counterpart of

```
public abstract boolean compareAndSet(T obj,
  int expect, int update)
```

Answer

```
@MJI
public boolean compareAndSet__Ljava_lang_Object_2II__Z
  (MJIEnv env, int objRef, int obj, int expect, int update)
```

The Java_my_util_TeleTYpewriter_isTTY method



1. Model checking of `Main.main`.
2. Execution of `isTTY____Z`.
3. Conversion of `TeleTYpewriter.class` to a `jclass` value.
4. Execution of `Java_my_util_TeleTYpewriter_isTTY`.
5. Conversion of `jboolean` value to a `boolean` value.

GenPeer

The app `GenPeer`, which is part of the package `gov.nasa.jpf.tool`, generates the framework of a native peer MJI class from a class.

The command

```
java -cp /cs/fac/packages/jpf/jpf-core/build/jpf.jar \
gov.nasa.jpf.tool.GenPeer
```

generates the output

```
usage:  'GenPeer [<option>..] <className> [<method>..]'
options: -s  : system peer class (gov.nasa.jpf.vm)
         -ci : create <clinit> MJI method
         -m  : create mangled method names
         -a  : create MJI methods for all target class meth
```

GenPeer

The command

```
java -Djava.library.path=<folder containing libttyutil.so>
  -cp /cs/fac/packages/jpf/jpf-core/build/jpf.jar \
  gov.nasa.jpf.tool.GenPeer -m -a my.util.TeleTYpewriter
```

generates the output

```
import gov.nasa.jpf.vm.MJIEnv;
import gov.nasa.jpf.vm.NativePeer;
import gov.nasa.jpf.annotation.MJI;

public class JPF_my_util_TeleTYpewriter extends NativePeer
  @MJI
  public boolean isTTY____Z(MJIEnv env, int clsObjRef) {
    boolean v = (boolean)0;
    return v;
  }
  ...
}
```

Native peers

```
@MJI
public boolean isTTY____Z(MJIEnv env, int clsObjRef) {
  System.out.println("Invoking isTTY");
  return TeleTYpewriter.isTTY();
}

@MJI
public int getTTYName____Ljava_lang_String_2(MJIEnv env, in
  System.out.println("Invoking getTTYName");
  String name = TeleTYpewriter.getTTYName();
  int ref = env.newString(name);
  return ref;
}
```

Native peers

To ensure that JPF uses the native peer, rather than the original class, we need to add the native peer to the host JVM's classpath.

```
target=Main
classpath=<folder containing Main.class>, \
  <folder containing the folder my of orginal class>
native_classpath= \
  <folder containing JPF_my_util_TeleTYpewriter.class>, \
  <folder containing the folder my of orginal class>
```

Before running JPF, we have to set the native library path. This can be done as follows.

```
setenv LD_LIBRARY_PATH <folder that contains libttyutil.so>
```

```
...
========================================================== sea
Invoking isTTY
Invoking getTTYName
TTY: /dev/pts/15

========================================================== resu
no errors detected
...
```

Handling native methods

Third approach

jpf-nhandler

jpf-nhandler is an extension of JPF. It automatically delegates the execution of methods from JPF to the host JVM.

`https://github.com/javapathfinder/jpf-nhandler`

jpf-nhandler has been installed on the EECS system.

jpf-nhandler

To use jpf-nhandler add

```
jpf-nhandler=/cs/fac/packages/jpf/jpf-nhandler
```

to the `site.properties` file in the `.jpf` folder.

jpf-nhandler

To use jpf-nhandler add

```
jpf-nhandler=/cs/fac/packages/jpf/jpf-nhandler
```

to the `site.properties` file in the `.jpf` folder.
jpf-nhandler can be applied to the `Main` app with the following properties file.

```
@using jpf-nhandler
target=Main
classpath=<folder containing Main.class>, \
native_classpath= \
  <folder containing the folder my of orginal class>
nhandler.delegateUnhandledNative=true
```

jpf-nhandler

To use jpf-nhandler add

```
jpf-nhandler=/cs/fac/packages/jpf/jpf-nhandler
```

to the `site.properties` file in the `.jpf` folder.
jpf-nhandler can be applied to the `Main` app with the following properties file.

```
@using jpf-nhandler
target=Main
classpath=<folder containing Main.class>, \
native_classpath= \
  <folder containing the folder my of orginal class>
nhandler.delegateUnhandledNative=true
```

Before running JPF, we have to set the native library path. This can be done as follows.

```
setenv LD_LIBRARY_PATH <folder that contains libttyutil.so>
```

jpf-nhandler

```
JavaPathfinder core system v8.0 (rev 26e11d1de726c19ba8ae1(
========================================================== syst
Main.main()
========================================================== sear
* DELEGATING Unhandled Native -> my.util.TeleTYpewriter.isT
* DELEGATING Unhandled Native -> my.util.TeleTYpewriter.get
TTY: /dev/pts/7
========================================================== resu
no errors detected
========================================================== stat
elapsed time:       00:00:01
states:             new=1,visited=0,backtracked=1,end=1
search:             maxDepth=1,constraints=0
choice generators:  thread=1 (signal=0,lock=1,sharedRef=0,th
heap:               new=571,released=19,maxLive=0,gcCycles=1
instructions:       9970
max memory:         240MB
```