CHAPTER 7

■ ■ ■

Java Native Interface

QUESTIONS AND EXERCISES

1. What is JNI?

   Answer:

   The Java Native Interface (JNI) is a programming interface that facilitates interaction between Java code and code written in native languages such as C, C++, FORTRAN, etc. The JNI supports calling C and C++ functions directly from Java. If you need to use native code written in any other language such as FORTRAN, you can use a C/C++ wrapper function to call it from Java. Interaction can take place both ways. Java code can call native code and vice versa.

2. What is the difference in using the `load()` and `loadLibrary()` method of the `System` class to load a shared library?

   Answer:

   The `load(String filename)` static method of the `System` class loads the native library specified by the `filename` argument. The filename argument must be an absolute path name.

   The `loadLibrary(String libname)` static method of teh `System` class loads the native library specified by the `libname` argument. The `libname` argument must not contain any platform specific prefix, file extension, or path. The method will add the prefix and file extension to `libname` depending on teh platform.

3. What are the JNI equivalent data types for Java primitive types?

Answer:

Java Primitive Types	Native Primitive Type	Description
boolean	jboolean	Unsigned 8 bits
byte	jbyte	Signed 8 bits
char	jchar	Unsigned 16 bits
double	jdouble	64 bits
float	jfloat	32 bits
int	jint	Signed 32 bits
long	jlong	Signed 64 bits
short	jshort	Signed 16 bits
void	void	N/A

4. What are the JNI equivalent data types used to represent objects of the `java.lang.String`, `java.lang.Class`, `java.lang.Throwable` classes, and any Java classes?

Answer:

The JNI equivalent data types to represent object of the `java.lang.String`, `java.lang.Class`, `java.lang.Throwable` classes and any Java classes are `jstring`, `jclass`, `jthrowable`, and `jobject`, respectively.

5. Suppose you have the reference of an object in a variable named `myObject` in the C++ native code. How do you get the reference of the class of this object using a JNI function?

Answer:

```
jclass cls = env->GetObjectClass(myObject);
```

6. What JNI function do you use to get the reference of a Java class using the class name?

Answer:

You use the `FindClass(JNIEnv *env, const char *className)` JNI function to get the reference of a class object using the class name.

7. What JNI function do you use to get the reference of the `Module` object of a Java class?

Answer:

From JDK9, a class is a member of a module. You can use the `GetModule()` JNI function to get the reference of the `java.lang.Module` object of a class. The function signature is as follows:

```
object GetModule(JNIEnv *env, jclass cls);
```

The function returns the `java.lang.Module` object for the module that the class is a member of. If the class is not in a named module, the unnamed module of the class loader for the class is returned. If the class represents an array type, this function returns the `Module` object for the element type. If the class represents a primitive type or `void`, the `Module` object for the `java.base` module is returned.

8. What is the difference in using the `AllocObject()` and `NewObject()` JNI functions to create an object of a Java class?

Answer:

The `AllocObject()` JNI function allocates memory for a Java object without invoking any of its constructors. Note that all instance fields will have their default values according to their data types. Instance fields will not be initialized when you use `AllocObject()` JNI function and no instance initializer will be invoked either.

The `NewObject()` JNI function creates an object of a Java class by invoking one of its constructors.

9. What is the Invocation API.

Answer:

The part of the JNI API that lets you create and load a JVM in native code is known as the *Invocation API*.

10. Describe exception handling in JNI.

Commented [KS1]: Check font

Answer:

The JNI lets you handle exceptions in native code. Native code can detect and handle exceptions that are thrown in the JVM as a result of calling a JNI function. Native code can also throw an exception that can be propagated to Java code. The exception handling mechanism in the native code differs from that of the Java code. When an exception is thrown in Java code, the control is transferred immediately to the nearest `catch` block that can handle the exception. When an exception is thrown during native code execution, the native code keeps executing and the exception remains pending until the control returns to the Java code. Once an exception is pending, you should not execute any other JNI functions except the ones that free native resources.

If an exception occurs in the native code, you can clear the exception and handle the exceptional condition in the native code. Use the `ExceptionClear()` JNI function to clear a pending exception. Once you clear the exception, that exception is not pending anymore.

If an exception occurs in the native code, you can return the control to the caller by using a `return` statement and let the caller handle the exception as shown in Java code.

You can also throw a new exception in the native code. Note that throwing an exception from the native code does not transfer the control back to the Java code. You must write code such as a `return` statement to transfer the control back to the Java code, so the exception you throw is handled in Java. You can throw an exception in the native code using either of the following two JNI functions. `jint Throw(jthrowable obj)` or `jint ThrowNew(jclass clazz, const char *message)`.

11. List the JNI functions that support thread synchronization in native code.

Answer:

- `jint MonitorEnter(jobject obj)`
- `jint MonitorExit(jobject obj)`