The Definitive Guide to Building Java Robots

Scott Preston

Apress®

ISBN: 1-59059-556-4

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editor: Steve Anglin
Technical Reviewer: Simon Ritter
Editorial Board: Steve Anglin, Dan Appleman, Ewan Buckingham, Gary Cornell, Tony Davis, Jason Gilmore, Jonathan Hassell, Chris Mills, Dominic Shakeshaft, Jim Sumser
Project Manager: Sofia Marchant
Copy Edit Manager: Nicole LeClerc
Copy Editor: Mike McGee
Assistant Production Director: Kari Brooks-Copony
Production Editor: Katie Stence
Compositor: Susan Glinert
Proofreader: April Eddy
Indexer: Carol Burbo
Artist: April Milne
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail orders-ny@springer-sbm.com, or visit http://www.springeronline.com.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail info@apress.com, or visit http://www.apress.com.

The source code for this book is available to readers at http://www.apress.com in the Source Code section.

CHAPTER 5

Speech

It usually takes more than three weeks to prepare a good impromptu speech.

— Mark Twain

5.0 Introduction

It won't take you that long to get your PC to speak. Outside of download times, it should only be about ten minutes. I'm also not going to use any microcontrollers in this chapter. This is going to run 100 percent off a PC, so as long as you have a soundcard, speakers, and a microphone, you'll be in good shape.

To get started, follow these steps:

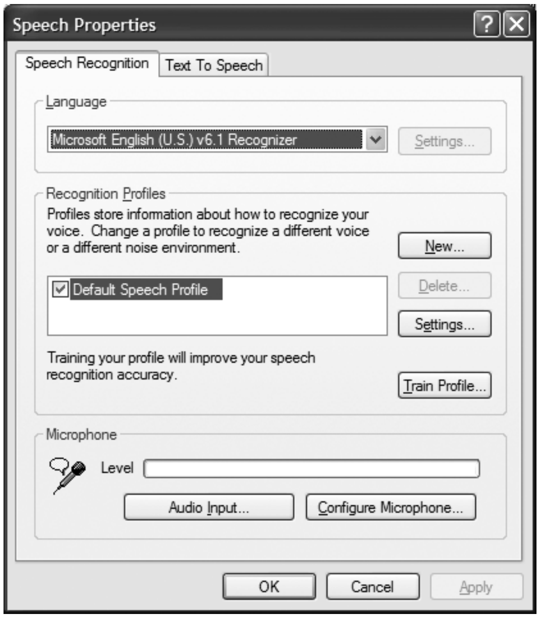
1. Make sure your soundcard is working.
2. Download and install FreeTTS from SourceForge from http://freetts.sourceforge.net. This will install the Java Speech API (JSAPI) and the FreeTTS JARs.
3. Add the following JARs to your class path:
   - jsapi.jar and freetts.jar: Both are available for download from http://freetts.sourceforge.net.
   - sphinx4.jar: Can be downloaded from http://cmusphinx.sourceforge.net.
   - WSJ_8gau_13dCep_16k_40mel_130Hz_6800Hz.jar: Available for download from http://cmusphinx.sourceforge.net/sphinx4/.
4. Download and install the Quadmore DLLs from the Source Code area of the Apress web site (www.apress.com) or from www.quadmore.com.

Note The files from the Apress web site follow the examples in this book. If you download from www.quadmore.com, you will need to modify your examples so they don't include any package structure. Both the QuadmoreTTS.class and QuadmoreSR.class will also need to be in your class path.

5. Install the Microsoft Speech SDK for Windows. This will install the other voices and set up your machine for text to speech, as well as speech recognition using the Microsoft API. You can download these at www.microsoft.com/speech.

6. Once you've downloaded all the files, make sure you spend 15 minutes or so training your system to recognize your voice. You'll also need to buy a headset microphone (about $20), which picks up less noise than external microphones.

To configure Microsoft Speech, see Figures 5-1 and 5-2, shown next.


Speech Properties

Language: Microsoft English (U.S.) v6.1 Recognizer

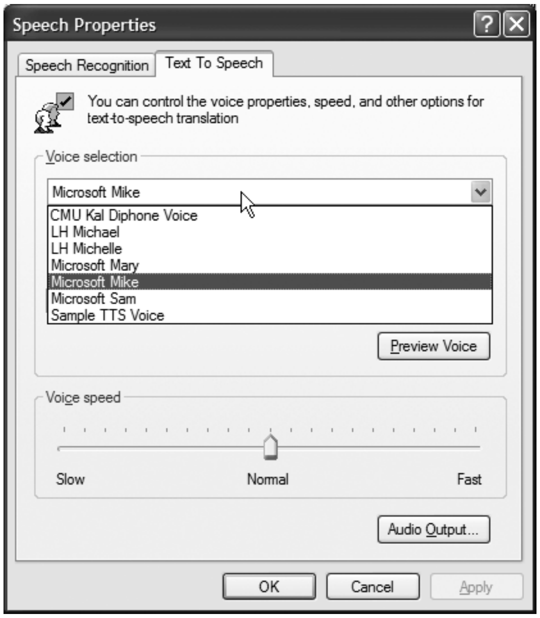
Recognition Profiles

Profiles store information about how to recognize your voice. Change a profile to recognize a different voice or a different noise environment.

☑ Default Speech Profile

Training your profile will improve your speech recognition accuracy.

Figure 5-1. *The Microsoft Speech Recognition tab*


Speech Properties

Speech Recognition | Text To Speech

You can control the voice properties, speed, and other options for text-to-speech translation

Voice selection

Microsoft Mike

CMU Kal Diphone Voice
LH Michael
LH Michelle
Microsoft Mary
Microsoft Mike
Microsoft Sam
Sample TTS Voice

Preview Voice

Voice speed

Slow Normal Fast

Audio Output...

OK Cancel Apply

Figure 5-2. *The Microsoft Text To Speech tab*

What Is Speech Technology?

Speech technology consists of speech synthesis and speech recognition. The speech recognition engines are responsible for converting acoustical signals to digital signals, and then to text. Two modes of speech recognition are available:

- Dictation: Users read data directly into a microphone. The range of words the engine can recognize is limited to the recognizers, grammar, or dictionary of recognizable words.
- Command and control: Users speak commands or ask questions. The range of words the engine can recognize in this case is usually defined by a limited grammar. This mode often eliminates the need to “train” the recognizers.

The speech synthesizer engines are responsible for converting text to a spoken language. This process first breaks the words into phonemes, which are then transformed to a digital audio signal for playback.

In this chapter, I’ll introduce two types of speech recognition engines: one for continuous dictation using JNI (see the following section), and one using command and control. I’ll also introduce three different speech synthesizers: two in Java and one using JNI.

Before I start with speech recognition or synthesis, the following is a quick-start reference to the Java Native Interface or JNI.

The Java Native Interface (JNI)

JNI allows programs that run within the Java Virtual Machine (JVM) to operate with applications and libraries written in other languages. To illustrate, let's create a simple native code example using Visual C++.

1. Open Visual Studio.
2. Select Managed C++ Library. If it's named SimpleJNI, the tool will create the following files:
   - AssemblyInfo.cpp: C++ source; contains custom attributes
   - SimpleJNI.cpp: C++ source; the main file in .dll source
   - SimpleJNI.h: Header file
   - SimpleJNI.ncb: IntelliSense database
   - SimpleJNI.sln: Solution file
   - SimpleJNI.suo: Solution Options file
   - SimpleJNI.vcproj: Project file
   - ReadMe.txt: ReadMe file
   - Stdafx.cpp: C++ source; contains standard system includes
   - Stdafx.h: C++ header; contains standard system includes
3. Create your native Java class. In Example 5-1, I'll call this TempConvert.

Example 5-1. *TempConvert.java*

```
package com.scottpreston.javarobot.chapter5;

public class TempConvert {

    //  this is a DLL in system path SimpleJNI.dll
    static {
        System.loadLibrary("SimpleJNI");
    }

    // native method
    public native float CtoF(float c);

    // native method
    public native float FtoC(float f);
    // sample program
    public static void main(String args[]) {
        TempConvert tc = new TempConvert();
        for (int c = 0; c < 101; c++) {
            System.out.println("c=" + c + ",f=" + tc.CtoF(c));
        }
    }
}
```

4. Run javah on the compiled .class file. Javah produces C and C++ header files from your native Java class.

```
javah - jni TempConvert
```

■ **Note** From your IDE, you can either go to the compiled class path (for example, /bin/com/scottpreston/javarobot/chapter5) or you can specify the class name from path root or a compiled JAR file.

The output of the file is shown in Example 5-2.

Example 5-2. *TempConvert.h*

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

#ifndef _Included_TempConvert
#define _Included_TempConvert
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     TempConvert
 * Method:    CtoF
 * Signature: (F)F
 */
JNIEXPORT jfloat JNICALL Java_com_scottpreston_javarobot_➥
 chapter5_TempConvert_CtoF
  (JNIEnv *, jobject, jfloat);
/*
 * Class:     TempConvert
 * Method:    FtoC
 * Signature: (F)F
 */
JNIEXPORT jfloat JNICALL Java_com_scottpreston_javarobot_➥
chapter5_TempConvert_FtoC
  (JNIEnv *, jobject, jfloat);

#ifdef __cplusplus
}
#endif
#endif
```

5. I had to make two modifications to this file because I compiled it from the /bin/com/scottpreston/javarobot/chapter5 directory.
   - I had to change the fully qualified class name from “Java_TempConvert” to “Java_com_scottpreston_javarobot_chapter5_TempConvert”.
   - I had to replace the “<jni.h>” with “jni.h”.
6. Copy jawt.h, jvmpi.h, jvmdi.h, jni_md.j, jni.h, and jawt_md.h to the project directory. These are required for jni.h and for compiling.
7. Add your native code.

■ **Note** In Example 5-3, I performed the calculation using the JNIEXPORT method. Later when discussing Microsoft speech and voice recognition, calls to other classes and methods are inserted here, rather than performing 100 percent of the native action within these methods.

Example 5-3. *SimpleJNI.h*

```
// SimpleJNI.h

#include "TempConvert.h"

JNIEXPORT jfloat JNICALL➥
Java_com_scottpreston_javarobot_chapter5_TempConvert_CtoF
  (JNIEnv *, jobject, jfloat f)
{        // native code
        float out = f *1.8 + 32;
        return out;
}
```

```
JNIEXPORT jfloat JNICALL➥
Java_com_scottpreston_javarobot_chapter5_TempConvert_FtoC
  (JNIEnv *, jobject, jfloat f)
{
        // native code
return (f -= 32) /= 1.8;
}
```

8. Build the .dll and place it in system32.

9. Run the program.

By using the JNI, you can take advantage of some native voices for speech synthesis and recognition. This way, if you already have some software or have configured your machine to recognize your voice with native software, you won't have to retrain your system. I'll discuss how to use the JNI for speech synthesis in section 5.2, but first let's talk about Java Speech Synthesis.

5.1 Speech Synthesis

Java Speech API (JSAPI) and Free Text To Speech (FreeTTS) are speech synthesis programs written entirely in Java. It's important to note that FreeTTS supports only a subset of JSAPI 1.0 and has some restrictions. Please reference the FreeTTS web site (http://freetts.sourceforge.net) for more information on the software and its usage.

I'll have three implementations for speech synthesis: two Java and one native. So, to standardize behavior, I'll create an interface called JVoice and then have two implementation classes: one using JSAPI, called JavaVoice; and one using FreeTTS, called FreeTTSVoice. Figure 5-3 shows a class diagram of the setup.

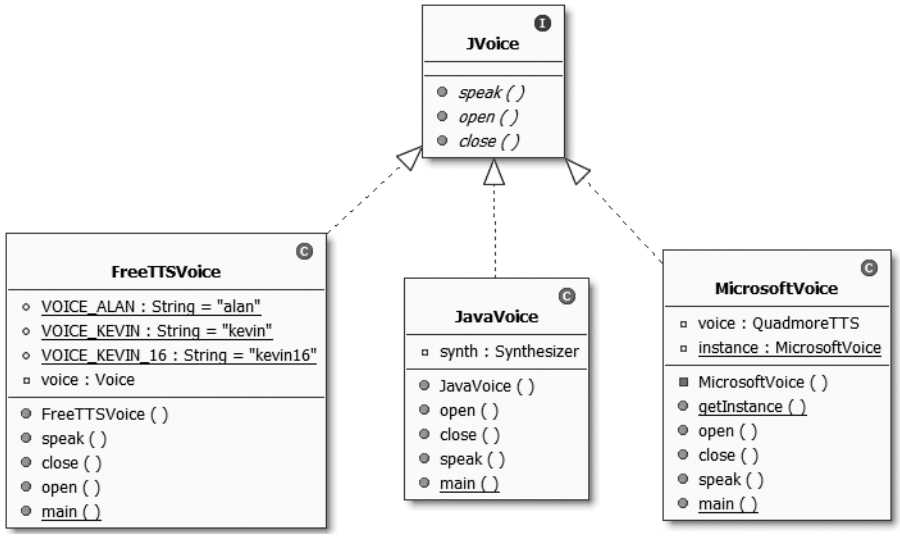


Figure 5-3. *Java speech synthesis classes*

Code Objective

Create an interface that standardizes behavior between three implementing speech synthesis classes.

Code Discussion

Our interface has only three methods. The first, open, gets the system ready for voice processing. The second, speak, takes a string input and contains the implementation for speaking. It throws an exception if there's a problem. Finally, there's close, which shuts down the voice processing. See Example 5-4.

Example 5-4. *Voice.java*

```
package com.scottpreston.javarobot.chapter5;

public interface JVoice {

    // opens or allocates voice engine
    public void open();
    // speaks
    public void speak(String words) throws Exception;
    // closes or deallocates voice engine
    public void close();

}
```

Code Objective

The code objective here is to create a speech synthesis implementation using JSAPI 1.0.

Code Discussion

The class that does all of our work for the Java Speech API is the java.speech.synthesis. Synthesizer class. To use this class, we need to create a new Synthesizer via the Central. createSynthesizer() method. This allows us to create any type of synthesizer we like with the constructor being a SynthesizerModeDesc class. After construction, the other methods follow our interface defined in Example 5-4.

The method open() calls the allocate() method on the Synthesizer. The close() method calls deallocate() on the Synthesizer.

The speak() method does three things. First, it calls resume() on the Synthesizer because it's recently allocated and needs to change its state to RESUMED so it can begin processing text to speech. Second, we call speakPlainText because we want to ignore Java Speech Markup Language (JSML). Third, we call waitEngineState() because we want to wait until the engine has placed itself in the QUEUE_EMPTY state. It does this when it's done talking. See Example 5-5.

Example 5-5. *JavaVoice.java*

```
package com.scottpreston.javarobot.chapter5;

import java.util.Locale;

import javax.speech.Central;
import javax.speech.synthesis.Synthesizer;
import javax.speech.synthesis.SynthesizerModeDesc;

public class JavaVoice implements JVoice {

    private Synthesizer synth;

    public JavaVoice() throws Exception {
        // constructs synthesizer for US English

        synth = Central.createSynthesizer(new SynthesizerModeDesc(
                null, // engine name
                "general", // mode name
                Locale.US, // local
                null, // Boolean, running
                null)); // Voices[]
    }
    // allocates synthesizer resources, puts engine in state ALLOCATED.
    public void open() {
        try {
            synth.allocate();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    // deallocates synthesizer resources, puts engine in state DEALLOCATED.
    public void close() {
        try {
            synth.deallocate();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    // speaks
    public void speak(String words) throws Exception {
        // removes from paused state as set by allocate
        synth.resume();
        // speaks plain text and text is not interpreted by Java Speech Markup
        // Language JSML
        synth.speakPlainText(words, null);
        // waits until queue is empty
        synth.waitEngineState(Synthesizer.QUEUE_EMPTY);
    }

    // sample program
    public static void main(String[] args) {
        try {

            JavaVoice voice = new JavaVoice();
            voice.open();
            voice.speak("Java Robots Are Cool!");
            voice.close();

        } catch (Exception e) {
            e.printStackTrace();
        }
        System.out.println("done");
    }
}
```

Code Objective

The code objective here is to create a speech synthesis implementation using FreeTTS.

Code Discussion

First, I've created three static fields with the names alan, kevin, and kevin16. Alan sounds the best, but his domain (things he can speak) is limited to date and time sounds. Kevin is an 8-bit voice of unlimited domain (any word) and sounds very close to the JavaVoice class created in Example 5-5. Kevin16 is a 16-bit voice, is medium quality, and has unlimited domain. The remaining field is com.sun.speech.freetts.Voice called voice.

I construct voice in the class constructor via the getInstance().getVoice()method from the VoiceManager. The remaining methods, open(), speak(), and close(), are self-explanatory. See Example 5-6.

Example 5-6. *FreeTTSVoice.java*

```
package com.scottpreston.javarobot.chapter5;

import com.sun.speech.freetts.Voice;
import com.sun.speech.freetts.VoiceManager;

public class FreeTTSVoice implements JVoice {

    // create these for use in constructor
    public static final String VOICE_ALAN = "alan";
    public static final String VOICE_KEVIN = "kevin";
    public static final String VOICE_KEVIN_16 = "kevin16";
    private Voice voice;

    // creates with name
    public FreeTTSVoice(String voiceName) {
        voice = VoiceManager.getInstance().getVoice(voiceName);
    }

    // speaks
    public void speak(String msg) {
        voice.speak(msg);
    }
    // deallocates and frees resources
    public void close() {
        voice.deallocate();
    }
    // allocates and opens resources
    public void open() {
        voice.allocate();
    }
    // sample program
    public static void main(String[] args) {
        FreeTTSVoice me = new FreeTTSVoice(FreeTTSVoice.VOICE_KEVIN_16);
        me.open();
        me.speak("Java Robots Are Cool.");
        me.close();
    }
}
```

Speech Synthesis Using JNI

Sometimes, whether I like it or not, I need to use some native code to get functionality for my robots. In the next two examples I'll create a native text-to-speech class and a native speech recognition class. The C++ project can be downloaded from www.quadmore.com.

If you recall from the introduction, I must make sure the method name for the JNIEXPORT matches the fully qualified class name. See Example 5-7.

■ **Note** If you get a java.lang.UnsatisfiedLinkError, check to make sure the method names match.

Example 5-7. *QuadmoreTTS.h and QuadTTS.h*

```
JNIEXPORT jboolean JNICALL ➥
Java_com_scottpreston_javarobot_chapter5_QuadmoreTTS_SpeakDarling
```

Code Objective

The objective here is to use the JNI native class to synthesize speech.

Code Discussion

This class has a static block that calls the QuadTTS.dll. This DLL must be in the path; I put it in the c:\windows\system32 directory. The constructor is just a default QuadmoreTTS() and is excluded from the source. I have the three methods from the native code at my disposal—SpeakDarling(), setVoice(), and getVoiceToken()—but currently, I'm only using SpeakDarling(). See Example 5-8.

Example 5-8. *QuadmoreTTS.java*

```
package com.scottpreston.javarobot.chapter5;

public class QuadmoreTTS {

    // this is a DLL in system path QuadTTS.dll
    static {
        System.loadLibrary("QuadTTS");
    }

    // native method
    public native boolean SpeakDarling(String strInput);
    // native method
    public native boolean setVoiceToken(String s);
    // native method
    public native String getVoiceToken();

    // sample program
    public static void main(String args[]) {
        QuadmoreTTS v = new QuadmoreTTS();
        boolean result = v.SpeakDarling("Java Robots Are Cool!");
        System.out.println("done!");
    }
}
```

I could use the QuadmoreTTS class in my programs, but I decided to create a wrapper class that implements the JVoice interface of my other text-to-speech classes.

There are two fields—one is the QuadmoreTTS class I created in the previous example, and the other is a static instance of the MicrosoftVoice—because I only want one program at a time accessing the voice synthesis engine.

Next, I implement the following methods from JVoice: open(), close(), and speak(). While open() and close() do nothing, speak() calls the native class SpeakDarling() method. If there's an error from this class, I've thrown an exception.

The sample program, main(), says the same phrase done earlier in the last two speech implementation classes. See Example 5-9.

Example 5-9. *MicrosoftVoice.java*

```
package com.scottpreston.javarobot.chapter5;

public class MicrosoftVoice implements JVoice {

    // worker class for voice
    private QuadmoreTTS voice;
    // private instance to ensure only one is active
    private static MicrosoftVoice instance;

    // private constructor prevents initialization
    // called by getInstance
    private MicrosoftVoice() {
        voice = new QuadmoreTTS();
    }

    // static methods ensure one instance per class
    public static MicrosoftVoice getInstance() throws Exception {
        if (instance == null) {
            // returns self
            instance = new MicrosoftVoice();
        }
        return instance;
    }

    public void open() {
        // do nothing
    }

    public void close() {
        // do nothing
    }

    //speak, otherwise throw exception
    public void speak(String s) throws Exception {
        if (!voice.SpeakDarling(s)) {
            throw new Exception("Unable to speak");
        }
    }

    // sample usage
    public static void main(String args[]) {
        try {
            MicrosoftVoice v = MicrosoftVoice.getInstance();
            v.speak("Java Robots Are Cool!");
        } catch (Exception e) {
            e.printStackTrace();
            System.exit(1);
        }
        System.out.println("done!");
    }

}
```

Now that I've created three separate implementations of the voice, it's time to see what they all sound like. Try it and see which one you like.

In TTSCompare, I'm showing a technique called factory method in getVoice(). In a nutshell, this means that I can get whatever voice I want by sending a parameter—in this case, an int enumerating one of the voices. Because all of the voices share the same interface, I can send this back from the method and use them the same way I do the sample program main(). See Example 5-10.

Example 5-10. *TTSCompare.java*

```
package com.scottpreston.javarobot.chapter5;

public class TTSCompare {

    public static final int JAVA_VOICE = 0;
    public static final int FREETTS_VOICE = 1;
    public static final int MICROSOFT_VOICE = 2;

    public JVoice getVoice(int voiceID) throws Exception {

        JVoice voice;

        if (voiceID == FREETTS_VOICE) {
            voice = new FreeTTSVoice(FreeTTSVoice.VOICE_KEVIN_16);
        } else if (voiceID == MICROSOFT_VOICE) {
            voice = MicrosoftVoice.getInstance();
        } else {
            voice = new JavaVoice();
        }
        return voice;
    }

    // simple program to test all classes and compare quality
    public static void main(String[] args) {
        try {
            TTSCompare tts = new TTSCompare();
            // java voice
            JVoice voice1 = tts.getVoice(TTSCompare.JAVA_VOICE);
            // free tts voice
            JVoice voice2 = tts.getVoice(TTSCompare.FREETTS_VOICE);
            // microsoft voice
            JVoice voice3 = tts.getVoice(TTSCompare.MICROSOFT_VOICE);
            // open all of these
            voice1.open();
            voice2.open();
            voice3.open();
            // speak some text
            voice1.speak("Java Voice... Hello World!");
            voice2.speak("Free TTS Voice... Hello World!");
            voice3.speak("Microsoft Voice... Hello World!");
            // close them
            voice1.close();
            voice2.close();
            voice3.close();
        } catch (Exception e) {
            e.printStackTrace();
            System.exit(1);
        }

    }
}
```

If you'd like to improve the quality of the FreeTTS voice, there are other extensions available at http://freetts.sourceforge.net.

Section Summary

In this section, I programmed my PC to talk using three different implementations of the JVoice interface. Those classes are

- JavaVoice: This uses the JSAPI speech synthesis engine.
- FreeTTsVoice.java: This uses the FreeTTS speech synthesis engine.
- MicrosoftVoice.java: This uses JNI to connect the Microsoft Sound API and text-to-speech engine.

Make sure you look at that site for more details. In the next section, I'm going to talk about the second part of speech: recognition.

5.2 Speech Recognition

In this section, I'll show you two ways to create a speech recognition engine: the first using Java and the second using JNI. In these two examples, I'll show the two ways of speech recognition: one using continuous dictation, and one using command and control. Also, because I'm going to use two different implementations, I'll create an interface that both will implement.

A class diagram of the classes I'll create in this section is shown in Figure 5-4.

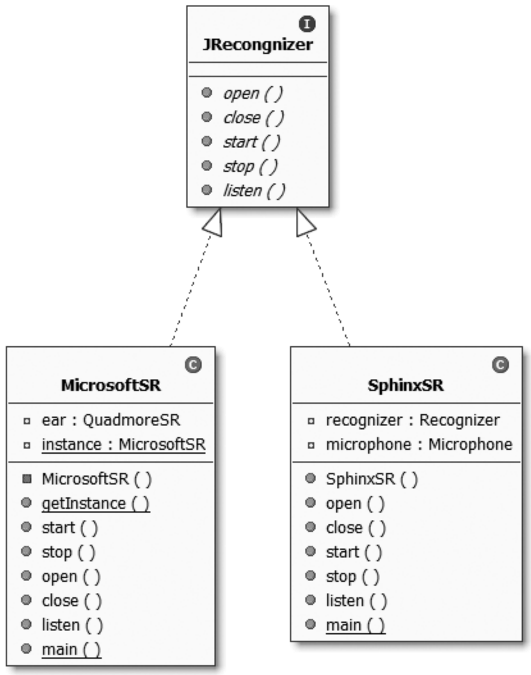


Figure 5-4. *Recognition classes*

Code Objective

The objective of this section is to create an interface that standardizes behavior between the two implementing speech recognition classes.

Code Discussion

This interface includes five methods. The open() and close() methods allocate and deallocate the same way that the JVoice interfaces implementation classes do. The start() method starts the recording device(), while the stop() method stops the recording device. Finally, the listen() method returns a String of the spoken phrase or word. See Example 5-11.

Example 5-11. *JRecognizer.java*

```
package com.scottpreston.javarobot.chapter5;

public interface JRecognizer {

    // opens device or allocates it
    public void open();
    // closes device or deallocates it
    public void close();
    // starts recognizer engine
    public void start();
    // stops recognizer engine
    public void stop();
    // starts listening
    public String listen();

}
```

First, speech recognition implementation uses the Sphinx-4 project at SourceForge. You can find more information and other detailed examples at the project home page at http://cmusphinx.sourceforge.net/sphinx4/. The Sphinx-4 is a speech recognition system written entirely in the Java programming language. It was created as a joint project between the Sphinx group at Carnegie Mellon University, Sun Microsystems, Mitsubishi Electric Research Labs (MERL), and Hewlett Packard (HP), with contributions from the University of California at Santa Cruz (USCS) and the Massachusetts Institute of Technology (MIT).

This program utilizes the "Command & Control" aspect of speech recognition. This means I must specify a grammar file. The format is called Java Speech Grammar Format.

The first part of the grammar file is the grammar header. The header format is

```
#JSGF version char-encoding local;
```

The second part is called the grammar name declaration. It's just a name; you can use either a package or a name.

```
grammar packageName.someName;
grammar someName;
```

The third part is optional and gives you the ability to import other grammars.

```
import <fullyQualifiedRuleName>;
import <fullGrammarName>;
```

The fourth part is the grammar body. This is where your rule definitions are located. You can use either pattern for the rule definition.

```
<ruleName> = rule expression;
public <ruleName> = rule expression;
```

Example 5-12 is a grammar file I'll use to open notepad. It has two rules, one for notepad and another for exiting.

Example 5-12. *notepad.gram*

```
#JSGF V1.0;

/**
 * JSGF Grammar for notepad example
 */

grammar notepad;

public <notepad> = (note pad);
public <exit> = (exit | good bye | quit);
```

Next, I need to create a class that will use this grammar file to actually launch a program.

Code Objective

The objective here is to perform basic speech recognition to open notepad, and then exit.

Code Discussion

After the grammar file, I need to create a configuration file for the recognizer. I used a provided configuration file from one of the examples, but had to make a few modifications to the grammar configuration for my class and my new grammar file. The configuration contains the other following sections:

- word recognizer
- decoder
- linguist
- dictionary
- acoustic model
- unit manager
- frontend
- monitors
- miscellaneous

In the following configuration, the dictionary is defined later in the configuration file. This contains all the words the recognizer can find.

The second line is the grammar location, which will be the class path of the project name: com.scottpreston.javarobot.chapter5.

The third line is the name of the grammar file at the location specified by the previous line.

The fourth line is required because all scores and probabilities are maintained in this class. See Example 5-13.

Example 5-13. *Grammar Configuration of notepad.config.xml*

```
<component name="jsgfGrammar" type="edu.cmu.sphinx.jsapi.JSGFGrammar">
    <property name="dictionary" value="dictionary"/>
    <property name="grammarLocation"      ➥
value="resource:/com.scottpreston.javarobot.chapter5.SphinxSR!/➥
com/scottpreston/javarobot/chapter5/"/>
    <property name="grammarName" value="notepad"/>
    <property name="logMath" value="logMath"/>
</component>
```

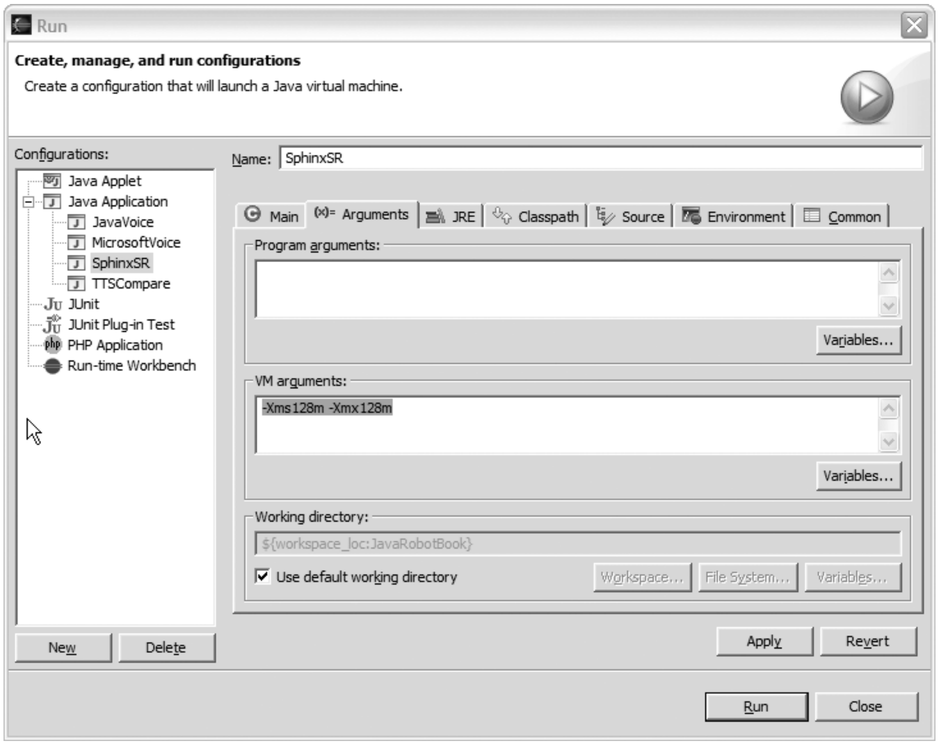
Now that I have both text files—notepad.gram and notepad.config.xml—I can create my speech recognition class, SphinxSR. It has two fields: recognizer and microphone. Both have parameters defined in the configuration file. I also get a copy of the grammar file. Later, I'll use the RuleGrammar to test what command I want to execute, because the rules are structured per command. Once I get the GSGFGrammar from the ConfigurationManager, I allocate() the GSFFGrammar object to create it. Then I set the private field ruleGrammar. At the end of the constructor, I loop through all the rule names and rules for informational purposes so I know what to speak.

The next methods—open() and close()—allocate() and deallocate() resources for speech recognition. The method, start(), begins recording from the microphone, while stop() halts recording from the microphone.

The listen() method calls recognize() on the recognizer. This method returns the recognition results. They can be either full or partial. As long as the result is not null, I can call a few methods on it, but the only one I care about is the one that returns the best and final result with no filler: getBestFinalResultNoFiller(). This returns a string of the spoken words. Next, to test the ruleName, I get the ruleNames from the ruleGrammar object. Then I create a RuleParse object to parse the resultText against the ruleName. If there's a match, I return the ruleName.

In main(), after constructing SphinxSR with the URL path to the configuration file, I open(), start(), and then listen() until I hear the commands/ruleNames I'm looking for. When I hear the rule "notepad", I execute notepad. When I hear the rule "exit", I exit the program. See Example 5-14.

■ **Note** To get this to work and avoid an out-of-memory exception, you may be required to increase the memory size for your JVM by adding the arguments –Xms 128m and –Xmx 128m. To do this in Eclipse, click the Run menu, followed by Run, and then modify the VM arguments as shown in Figure 5-5.


Run

Create, manage, and run configurations

Create a configuration that will launch a Java virtual machine.

Configurations:
- Java Applet
- Java Application
  - JavaVoice
  - MicrosoftVoice
  - SphinxSR
  - TTSCompare
- JUnit
- JUnit Plug-in Test
- PHP Application
- Run-time Workbench

Name: SphinxSR

Main | Arguments | JRE | Classpath | Source | Environment | Common

Program arguments:

VM arguments: -Xms128m -Xmx128m

Working directory: ${workspace_loc:JavaRobotBook}

☑ Use default working directory

Figure 5-5. *Increasing memory in Eclipse for SphinxSR*

Example 5-14. *SphinxSR.java*

```
package com.scottpreston.javarobot.chapter5;

import java.net.URL;

import javax.speech.recognition.RuleGrammar;
import javax.speech.recognition.RuleParse;

import edu.cmu.sphinx.frontend.util.Microphone;
import edu.cmu.sphinx.jsapi.JSGFGrammar;
import edu.cmu.sphinx.recognizer.Recognizer;
import edu.cmu.sphinx.result.Result;
import edu.cmu.sphinx.util.props.ConfigurationManager;

public class SphinxSR implements JRecognizer {

    private Recognizer recognizer;
    private Microphone microphone;
    private RuleGrammar ruleGrammar;
    public SphinxSR(URL url) throws Exception {
        //loads configuration data from XML-based configuration file
        ConfigurationManager cm = new ConfigurationManager(url);
        // gets component by name
        recognizer = (Recognizer) cm.lookup("recognizer");
        microphone = (Microphone) cm.lookup("microphone");
        // get grammar file
        JSGFGrammar gram = (JSGFGrammar) cm.lookup("jsgfGrammar");
        // create the grammar
        gram.allocate();
        // get rules
        ruleGrammar = gram.getRuleGrammar();
        // get rule names
        String[] rules = ruleGrammar.listRuleNames();
        // display to console so you know what to speak.
        for (int i=0; i < rules.length;i++) {
            System.out.println("rule name = " + rules[i]);
            System.out.println("rule = " +ruleGrammar.getRule(rules[i]).➥
toString());;
        }
        // separator
        System.out.println("----");
    }

    // allocates resources
    public void open() {
        try {
            recognizer.allocate();
        } catch (Exception e) {
            e.printStackTrace();
            System.exit(1);
        }
    }

    // deallocates resources
    public void close() {
        recognizer.deallocate();
    }

    // start recording
    public void start() {
        // begins capturing audio data
        if (microphone.startRecording() == false) {
            recognizer.deallocate();
            System.exit(1);
        }
    }
    // stop capturing audio data
    public void stop() {
        microphone.stopRecording();
    }

    public String listen(){
        // gets recognition results from recognizer
        Result result = recognizer.recognize();
        String ruleName = "";
        if (result != null) {
            // gets best and final with no filler words
            String resultText = result.getBestFinalResultNoFiller();
            // display text
            System.out.println("I heard --> " + resultText);
            RuleParse rParse = null;
              String [] rules = ruleGrammar.listRuleNames();
            for (int i=0; i < rules.length;i++) {
                try {
                   // test rule name and execute
                rParse = ruleGrammar.parse(resultText,rules[i]);
                // set rulename
                ruleName = rParse.getRuleName().getRuleName();
                } catch (Exception e) {
                    // do nothing
                }

            }
        }
        // return rulename
        return ruleName;
    }

    // test class
    public static void main(String[] args) throws Exception {
        // this is the configuration file
        URL url = SphinxSR.class.getResource("notepad.config.xml");
        SphinxSR sr = new SphinxSR(url);
        System.out.println("Loading...");
        sr.open();
        sr.start();
        String rule = "";
        System.out.println("Listening...");
        while (true) {
            rule = sr.listen();
            if (rule.equals("notepad")) {
                Runtime.getRuntime().exec("cmd /c notepad.exe");
            }
            if (rule.equals("exit")) {
                break;
            }
        }
        sr.stop();
        sr.close();
        System.out.println("done!");
    }

}
```

Next, I want to use JNI and implement a continuous dictation example. This will not use a grammar file, but will require you to train the recognizer as to how you dictate words.

Code Objective

The objective here is to perform basic speech recognition to open notepad, and then exit via JNI using continuous dictation.

Code Discussion

Just like the modification in Example 5-3, I'll only modify the methods to match our new package signature. See Example 5-15.

Example 5-15. *QuadmoreSR.h and QuadSR.h*

```
JNIEXPORT jstring JNICALL➥
Java_com_scottpreston_javarobot_chapter5_QuadmoreSR_TakeDictation
```

This class has a static block that calls QuadSR.dll. This DLL must be in the path, so I've put it in the c:\windows\system32 directory. After the static block, I define a single native method from the C++ project called TakeDictation. This method returns a string that I output to System.out. See Example 5-16.

Example 5-16. *QuadmoreSR.java*

```
package com.scottpreston.javarobot.chapter5;

public class QuadmoreSR {

    // this is a DLL in system path QuadSR.dll
    static {
        System.loadLibrary("QuadSR");
    }
    // from native class
    public native String TakeDictation();
    // sample program
    public static void main(String args[]) {
        int i;
        i = 0;
        String strRecognizedText;
        System.out.println("Beginning speech recognition...");
        // create speech recognition class
        QuadmoreSR sr = new QuadmoreSR();
        // wait until four words are heard
        while (i < 4) {
            strRecognizedText = sr.TakeDictation();
            System.out.println("\n");
            System.out.println(strRecognizedText);
            i++;
        }
        System.out.println("Done.");
    }

}
```

Just like Example 5-9, I could use the QuadmoreSR class in my programs, but I decided to create a wrapper class that implements the SpeechRecognizer interface. There is no need to implement the methods start(), stop(), open(), and close(), but they are required for the interface so I will just create stubs. See Example 5-17.

Example 5-17. *MicrosoftSR.java*

```
package com.scottpreston.javarobot.chapter5;

public class MicrosoftSR implements JRecognizer {

    // class used for recognizer
    private QuadmoreSR ear;

    // holds single instance of recognizer
    private static MicrosoftSR instance;

    // private constructor prevents initialization
    // called by getInstance()
    private MicrosoftSR() {

        ear = new QuadmoreSR();
    }
    // gets single instance of speech recognizer.
    public static MicrosoftSR getInstance() throws Exception {
        if (instance == null) {
            instance = new MicrosoftSR();
        }
        return instance;
    }

    public void start() {
    } // do nothing

    public void stop() {
    } // do nothing

    public void open() {
    } // do nothing

    public void close() {
    } // do nothing

    // starts listening and returning strings of spoken text
    public String listen() {
        return ear.TakeDictation();
    }
    // sample usage
    public static void main(String[] args) {
        try {
            // gets instance
            MicrosoftSR sr = MicrosoftSR.getInstance();
            String words = "";
            System.out.println("Listening...");
            // loops until hears exit
            while (words.equalsIgnoreCase("exit") == false) {
                words = sr.listen();
                System.out.println("I heard --> " + words);
                // if it hears note, then it opens notepad
                if (words.equalsIgnoreCase("note")) {
                    Runtime.getRuntime().exec("cmd /c notepad.exe");
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
            System.exit(1);
        }
        System.out.println("done");
    }
}
```

Example 5-18 is useful when you want to hear what the Microsoft speech recognition engine “hears.” You can use whatever voice you want. Here I’m using the MicrosoftVoice to repeat what it hears.

Example 5-18. *EchoTalk.java*

```
package com.scottpreston.javarobot.chapter5;

public class EchoTalk {

    private MicrosoftVoice voice;
    private MicrosoftSR ear;

    public EchoTalk() throws Exception {
        // generic constructor gets instance of two worker classes
        voice = MicrosoftVoice.getInstance();
        ear = MicrosoftSR.getInstance();
        // give user instructions
        voice.speak("I will repeat what you say. Say exit, to end program.");
    }

    public void start() throws Exception {
        String words = "";
        // tell user to begin talking.
        voice.speak("listening");
        // this will loop until it hears 'exit'
        while (words.equalsIgnoreCase("exit") == false) {
            // gets words heard.
            words = ear.listen();
            //prints this to system out (good for debugging)
            System.out.println("I heard --> " + words);
            // say the words
            voice.speak(words);
        }
        // last words spoken.
        voice.speak("goodbye");
    }

    public static void main(String[] args) {

        try {
            EchoTalk echo = new EchoTalk();
            echo.start();
        } catch (Exception e) {
            //print error and exit.
            e.printStackTrace();
            System.exit(1);
        }

    }
}
```

Section Summary

In this section, I showed two examples of using speech recognition, both of which implemented the JRecognizer interface. The first used Sphinx-4 for Java to demonstrate command and control. The second used JNI and the Microsoft speech recognition engine to give an example of continuous dictation.

Before introducing the first example, I gave a brief overview of the Java Speech Grammar Format (JSGF). This is the format of what words the recognizer needs to understand.

The classes discussed were

- SphinxSR: This is the Java recognizer using command and control.
- MicrosoftSR: This is the continuous recognizer using JNI and Microsoft speech recognition engine. (You must train this engine prior to use.)

5.3 Chapter Summary

In this chapter, I talked about how to get your robot to talk and recognize what you're saying. I also showed you how to use speech technology written in other languages, and I gave a simple introduction to the Java Native Interface (JNI).

The JNI example, TempConvert, does a simple temperature conversion using a C++ project and class to perform the calculation. You should be able to follow the step-by-step example to create your own JNI classes.

In section 5.1, I introduced the JVoice interface to standardize implementation of the three text-to-speech classes. The first, JavaVoice, uses the Java Speech API (JSAPI) to produce speech from text. The second, FreeTTSVoice, uses the FreeTTS speech synthesis libraries to produce speech from text. The third, MicrosoftVoice, uses JNI to connect the speech synthesizer engine built in to my system after installing the Microsoft Speech API (MSAPI). Also in section 5.1, I showed a sample class that tests the voice quality of all three speech synthesis engines.

In section 5.2, I introduced the JRecognizer interface, used for standardizing implementations of the following two speech recognition classes. The first recognizer, SphinxSR, uses a grammar file, a configuration file, and the synthesizer from the Sphinx-4 project. The second recognizer uses JNI to connect to the Microsoft speech recognition engine. Also in section 5.2, I wrote a sample class that echoes what's heard in continuous dictation.

Personally, I like MicrosoftVoice for sound quality and SphinxSR for recognition. I'll use these in Chapter 9.

Now that our PC can speak and listen, it's time to get it to see, which is the topic of the next chapter.