

GNU Linear Programming Kit Java Binding

Reference Manual

Version 1.0.15

September 2010

Copyright © 2009, 2010 Heinrich Schuchardt, xypron.glpk@gmx.de

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided also that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions.

Windows is a registered trademark of Microsoft Corporation. Java is a trademark when it identifies a software product of Sun Microsystems, Inc.

Contents

1	Introduction	4
2	Architecture	5
2.1	GLPK library	5
2.1.1	Source	5
2.1.2	Linux	5
2.1.3	Windows	5
2.2	GLPK for Java JNI library	6
2.2.1	Source	6
2.2.2	Linux	6
2.2.3	Windows	6
2.3	GLPK for Java class library	6
2.3.1	Linux	6
2.3.2	Windows	7
2.3.3	Classpath	7
3	Classes	8
4	Usage	10
4.1	Loading the JNI library	10
4.2	Exceptions	10
4.2.1	Implementation details	11
4.3	Callbacks	11
4.4	Output listener	13
4.5	Aborting a GLPK library call	13
4.6	Threads	13
5	Examples	14
5.1	Lp.java	14
5.1.1	Description	14
5.1.2	Coding	14
5.2	Gmpl.java	16
5.2.1	Description	16
5.2.2	Coding	17
6	License	19

Chapter 1

Introduction

The GNU Linear Programming Kit (GLPK)[2] package supplies a solver for large scale linear programming (LP) and mixed integer programming (MIP). The GLPK project is hosted at <http://www.gnu.org/software/glpk>.

It has two mailing lists:

- help-glpk@gnu.org and
- bug-glpk@gnu.org.

To subscribe to one of these lists, please, send an empty mail with a Subject: header line of just "subscribe" to the list.

GLPK provides a library written in C and a standalone solver.

The source code provided at <ftp://gnu.ftp.org/gnu/glpk/> contains the documentation of the library in file `doc/glpk.pdf`.

The Java platform provides the Java Native Interface (JNI)[3] to integrate non-Java language libraries into Java applications.

Project GLPK for Java delivers a Java Binding for GLPK. It is hosted at <http://glpk-java.sourceforge.net/>.

To report problems and suggestions concerning GLPK for Java, please, send an email to the author at xypron.glpk@gmx.de.

Chapter 2

Architecture

A GLPK for Java application will consist of the following

- the GLPK library
- the GLPK for Java JNI library
- the GLPK for Java class library
- the application code.

2.1 GLPK library

2.1.1 Source

The source code to compile the GLPK library is provided at <ftp://gnu.ftp.org/gnu/glpk/>.

2.1.2 Linux

The GLPK library can be compiled from source code. Follow the instructions in file INSTALL provided in the source distribution. Precompiled packages are available in many Linux distributions.

The usual installation path for the library is `/usr/local/lib/libglpk.so`.

2.1.3 Windows

The GLPK library can be compiled from source code. The build and make files are in directory w32 for 32 bit Windows and in w64 for 64 bit Windows. The name of the created library is `glpk_4_45.dll` for revision 4.45.

A precompiled version of GLPK is provided at <http://winglpk.sourceforge.net>.

The library has to be in the search path for binaries. Either copy the library to a directory that is already in the path (e.g. `C:\windows\system32`) or update the path in the system settings of Windows.

2.2 GLPK for Java JNI library

2.2.1 Source

The source code to compile the GLPK for Java JNI library is provided at <http://glpk-java.sourceforge.net>.

2.2.2 Linux

The GLPK for Java JNI library can be compiled from source code. Follow the instructions in file INSTALL provided in the source distribution.

The usual installation path for the library is `/usr/local/lib/libglpk-java.so`.

2.2.3 Windows

The GLPK for Java JNI library can be compiled from source code. The build and make files are in directory `w32` for 32 bit Windows and in `w64` for 64 bit Windows. The name of the created library is `glpk_4_45_java.dll` for revision 4.45.

A precompiled version of GLPK for Java is provided at <http://winglpk.sourceforge.net>.

The library has to be in the search path for binaries. Either copy the library to a directory that is already in the path (e.g. `C:\windows\system32`) or update the path in the system settings of Windows.

2.3 GLPK for Java class library

The source code to compile the GLPK for Java class library is provided at <http://glpk-java.sourceforge.net>.

2.3.1 Linux

The GLPK for Java class library can be compiled from source code. Follow the instructions in file INSTALL provided in the source distribution.

The usual installation path for the library is `/usr/local/share/java/glpk-java.jar`.

For Debian and Ubuntu the following packages are needed for compilation:

- `libtool`
- `swig`
- `java-gcj-compat-dev`

2.3.2 Windows

The GLPK for Java class library can be compiled from source code. The build and make files are in directory w32 for 32 bit Windows and in w64 for 64 bit Windows. The name of the created library is glpk-java.jar.

A precompiled version of GLPK including GLPK-Java is provided at <http://winglpk.sourceforge.net>.

2.3.3 Classpath

The library has to be in the CLASSPATH. Update the classpath in the system settings of Windows or specify the classpath upon invocation of the application, e.g.

```
java -classpath ./glpk-java.jar;. MyApplication
```

Chapter 3

Classes

GLPK for Java uses the Simplified Wrapper and Interface Generator (SWIG)[4] to create the JNI interface to GLPK. Classes are created in path `org.gnu.glpk`.

Class `GlpkCallback` is called by the MIP solver callback routine.

Interface `GlpkCallbackListener` can be implemented to register a listener for class `GlpkCallback`.

Class `GlpkTerminal` is called by the MIP solver terminal output routine.

Interface `GlpkTerminalListener` can be implemented to register a listener for class `GlpkTerminal`.

Class `GlpkException` is thrown if an error occurs.

Class `GLPK` maps the functions from `include/glpk.h`.

Class `GLPKConstants` maps the constants from `include/glpk.h` to methods.

Class `GLPKJNI` contains the definitions of the native functions.

The following classes map structures from `include/glpk.h`:

- `glp_attr`
- `glp_bfcp`
- `glp_cpxcpx`
- `glp_data`
- `glp_iocp`
- `glp_iptcp`
- `glp_long`
- `glp_mpscp`
- `glp_prob`
- `glp_smcp`
- `glp_tran`
- `glp_tree`

- LPXKKT
- _glp_arc
- _glp_graph
- _glp_vertex

The following classes are used to map pointers:

- SWIGTYPE_p_double
- SWIGTYPE_p_f_p_glp_tree_p_void__void
- SWIGTYPE_p_f_p_q_const__char_v____void
- SWIGTYPE_p_f_p_void__void
- SWIGTYPE_p_f_p_void_p_q_const__char__int
- SWIGTYPE_p_int
- SWIGTYPE_p_p__glp_vertex
- SWIGTYPE_p_va_list
- SWIGTYPE_p_void

Chapter 4

Usage

Please, refer to file `doc/glpk.pdf` of the GLPK source distribution for a detailed description of the methods and constants.

4.1 Loading the JNI library

To be able to use the JNI library in a Java program it has to be loaded. The path to dynamic link libraries can be specified on the command line when calling the Java runtime, e.g.

```
java -Djava.library.path=/usr/local/lib/jni/libglpk_java
```

The following code is used in class GLPK to load the JNI library:

```
static {
    try {
        if (System.getProperty("os.name").toLowerCase().contains("windows")) {
            // try to load Windows library
            System.loadLibrary("glpk_4_45_java");
        } else {
            // try to load Linux library
            System.loadLibrary("glpk_java");
        }
    } catch (UnsatisfiedLinkError e) {
        System.err.println(
            "The dynamic link library for GLPK for Java could not be"
            + "loaded.\nConsider using\njava -Djava.library.path=");
        throw e;
    }
}
```

If the JNI library could not be loaded, you will receive an exception `java.lang.UnsatisfiedLinkError`.

4.2 Exceptions

When illegal parameters are passed to a function of the GLPK native library an exception `GlpkException` is thrown. Due to the architecture of GLPK all GLPK objects are invalid when such an exception

has occurred.

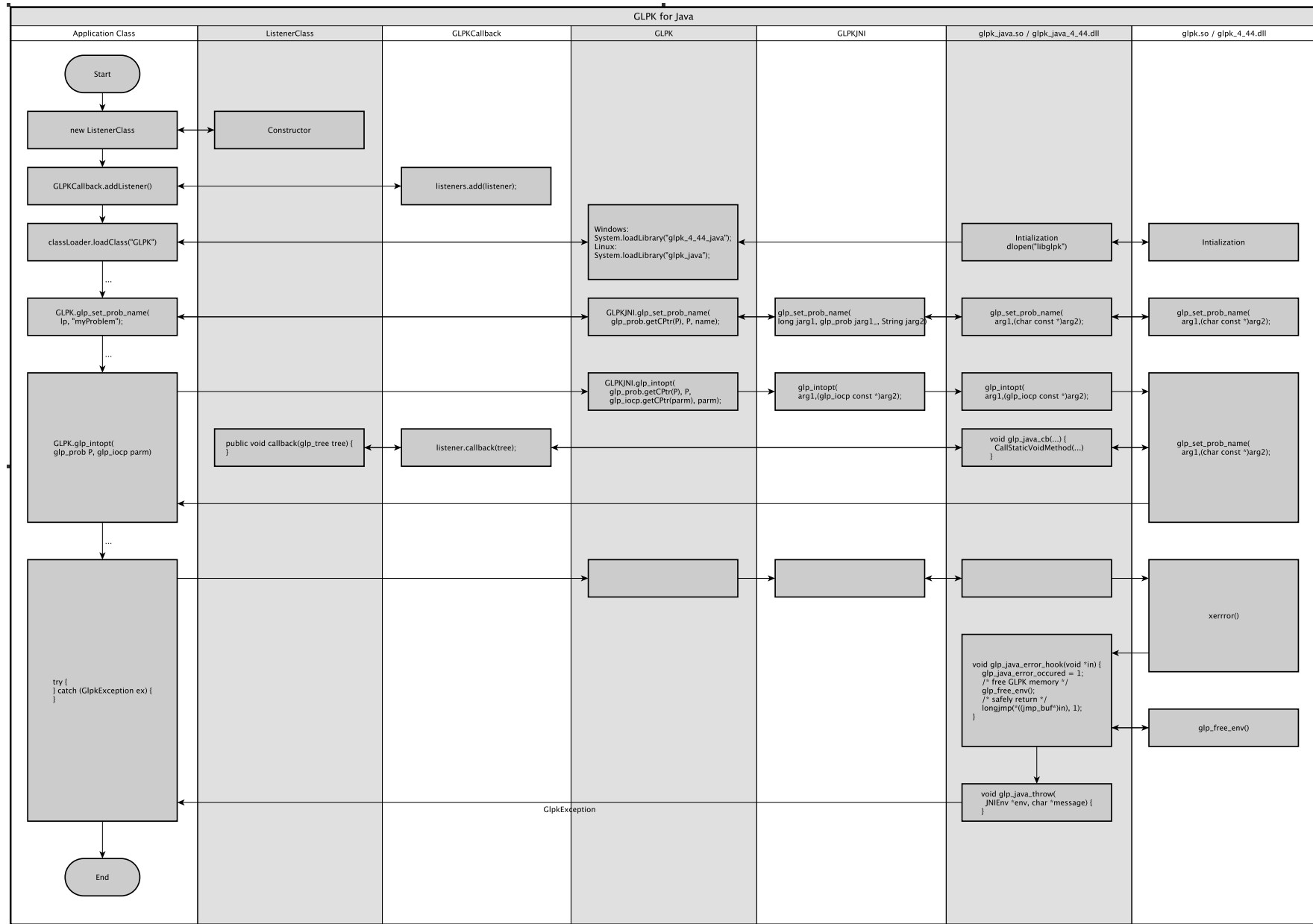
4.2.1 Implementation details

GLPK for Java registers a function `glp_java_error_hook()` to `glp_error_hook()` before calling an GLPK API function. If an error occurs function `glp_free_env` is called and a long jump is used to return to the calling environment. Then function `glp_java_throw()` is called which throws `GlpkException`.

4.3 Callbacks

The MIP solver provides a callback functionality. This is used to call method `callback` of class `GlpkCallback`. A Java program can listen to the callbacks by instantiating a class implementing interface `GlpkCallbackListener` and registering the object with method `addListener()` of class `GlpkCallback`. The listener can be deregistered with method `removeListener()`. The listener can use method `GLPK.glp_ios_reason()` to find out why it is called. For details see the GLPK library documentation.

Figure 4.1: Callbacks and Error Handling



4.4 Output listener

GLPK provides a hook for terminal output. A Java program can listen to the callbacks by instantiating a class implementing interface `GlpkTerminalListener` and registering the object with method `addListener()` of class `GlpkTerminal`. The listener can be deregistered with method `removeListener()`. After a call to `glp_free_env()` the `GlpkTerminal` has to be registered again by calling `GLPK.glp_term_hook(null, null)`. `glp_free_env()` is called if an exception `GlpkException` occurs.

4.5 Aborting a GLPK library call

Method `void GLPK.glp_java_error(String message)` can be used to abort any call to the GLPK library. An exception `GlpkException` will occur. As GLPK is not threadsafe the call must be placed in the same thread as the initial call that is to be aborted. The output method of a `GlpkTerminalListener` can be used for this purpose.

4.6 Threads

The GLPK library is not thread safe. Never two threads should be running that access the GLPK library at the same time. When a new thread accesses the library it should call `GLPK.glp_free_env()`. When using an `GlpkTerminalListener` it is necessary to register `GlpkTerminal` again by calling `GLPK.glp_term_hook(null, null)`.

When writing a GUI application it is advisable to use a separate thread for the calls to GLPK. Otherwise the GUI cannot react to events during the call to the GLPK library.

Chapter 5

Examples

Examples are provided in directory `examples/java` of the source distribution of GLPK for Java.

To compile the examples the classpath must point to `glpk-java.jar`, e.g.

```
javac -classpath /usr/local/shared/java/glpk-java.jar Example.java
```

To run the examples the classpath must point to `glpk-java.jar`. The `java.library.path` must point to the directory with the dynamic link libraries, eg.

```
java -Djava.library.path=/usr/local/lib/jni \  
-classpath /usr/local/shared/java/glpk-java.jar:. \  
Example
```

5.1 Lp.java

5.1.1 Description

This example solves a small linear problem and outputs the solution.

5.1.2 Coding

```
import org.gnu.glpk.GLPK;  
import org.gnu.glpk.GLPKConstants;  
import org.gnu.glpk.GlpkException;  
import org.gnu.glpk.SWIGTYPE_p_double;  
import org.gnu.glpk.SWIGTYPE_p_int;  
import org.gnu.glpk.glp_prob;  
import org.gnu.glpk.glp_smcp;  
  
public class Lp {  
    // Minimize  $z = (x_1 - x_2) / 2 + (1 - (x_1 - x_2)) = -.5 * x_1 + .5 * x_2 + 1$   
    //  
    // subject to  
    //  $0.0 \leq x_1 - x_2 \leq 0.2$   
    // where,  
    //  $0.0 \leq x_1 \leq 0.5$ 
```

```

// 0.0 <= x2 <= 0.5

public static void main(String[] arg) {
    glp_prob lp;
    glp_smcp parm;
    SWIGTYPE_p_int ind;
    SWIGTYPE_p_double val;
    int ret;

    try {
        // Create problem
        lp = GLPK.glp_create_prob();
        System.out.println("Problem created");
        GLPK.glp_set_prob_name(lp, "myProblem");

        // Define columns
        GLPK.glp_add_cols(lp, 2);
        GLPK.glp_set_col_name(lp, 1, "x1");
        GLPK.glp_set_col_kind(lp, 1, GLPKConstants.GLP_CV);
        GLPK.glp_set_col_bnds(lp, 1, GLPKConstants.GLP_DB, 0, .5);
        GLPK.glp_set_col_name(lp, 2, "x2");
        GLPK.glp_set_col_kind(lp, 2, GLPKConstants.GLP_CV);
        GLPK.glp_set_col_bnds(lp, 2, GLPKConstants.GLP_DB, 0, .5);

        // Create constraints
        GLPK.glp_add_rows(lp, 1);

        GLPK.glp_set_row_name(lp, 1, "c1");
        GLPK.glp_set_row_bnds(lp, 1, GLPKConstants.GLP_DB, 0, 0.2);
        ind = GLPK.new_intArray(3);
        GLPK.intArray_setitem(ind, 1, 1);
        GLPK.intArray_setitem(ind, 2, 2);
        val = GLPK.new_doubleArray(3);
        GLPK.doubleArray_setitem(val, 1, 1.);
        GLPK.doubleArray_setitem(val, 2, -1.);
        GLPK.glp_set_mat_row(lp, 1, 2, ind, val);

        // Define objective
        GLPK.glp_set_obj_name(lp, "z");
        GLPK.glp_set_obj_dir(lp, GLPKConstants.GLP_MIN);
        GLPK.glp_set_obj_coef(lp, 0, 1.);
        GLPK.glp_set_obj_coef(lp, 1, -.5);
        GLPK.glp_set_obj_coef(lp, 2, .5);

        // Solve model
        parm = new glp_smcp();
        GLPK.glp_init_smcp(parm);
        ret = GLPK.glp_simplex(lp, parm);

        // Retrieve solution
        if (ret == 0) {
            write_lp_solution(lp);
        } else {

```

```

        System.out.println("The problem could not be solved");
    }

    // Free memory
    GLPK.glp_delete_prob(lp);
} catch (GlpkException ex) {
    ex.printStackTrace();
}
}

/**
 * write simplex solution
 * @param lp problem
 */
static void write_lp_solution(glp_prob lp) {
    int i;
    int n;
    String name;
    double val;

    name = GLPK.glp_get_obj_name(lp);
    val = GLPK.glp_get_obj_val(lp);
    System.out.print(name);
    System.out.print(" = ");
    System.out.println(val);
    n = GLPK.glp_get_num_cols(lp);
    for (i = 1; i <= n; i++) {
        name = GLPK.glp_get_col_name(lp, i);
        val = GLPK.glp_get_col_prim(lp, i);
        System.out.print(name);
        System.out.print(" = ");
        System.out.println(val);
    }
}
}

```

5.2 Gmpl.java

5.2.1 Description

This example reads a GMPL file and executes it. The callback function is used to write an output line when a better MIP solution has been found.

Run the program with the model file as parameter.

```

java -Djava.library.path=/usr/local/lib \
-classpath /usr/local/shared/java/glpk-java.jar:. \
GLPKSwig marbles.mod

```


5.2.2 Coding

```
import org.gnu.glpk.GLPK;
import org.gnu.glpk.GLPKConstants;
import org.gnu.glpk.GlpkCallback;
import org.gnu.glpk.GlpkCallbackListener;
import org.gnu.glpk.glp_iocp;
import org.gnu.glpk.glp_prob;
import org.gnu.glpk.glp_tran;
import org.gnu.glpk.glp_tree;

public class Gmpl implements GlpkCallbackListener {

    public static void main(String[] arg) {
        if (1 != arg.length) {
            System.out.println("Usage: java Gmpl model.mod");
            return;
        }
        new Gmpl().solve(arg);
    }

    public void solve(String[] arg) {
        glp_prob lp = null;
        glp_tran tran;
        glp_iocp iocp;

        String fname;
        int skip = 0;
        int ret;

        GlpkCallback.addListener(this);

        fname = new String(arg[0]);

        lp = GLPK.glp_create_prob();
        System.out.println("Problem created");

        tran = GLPK.glp_mpl_alloc_wksp();
        ret = GLPK.glp_mpl_read_model(tran, fname, skip);
        if (ret != 0) {
            GLPK.glp_mpl_free_wksp(tran);
            GLPK.glp_delete_prob(lp);
            throw new RuntimeException("Model file not found: " + fname);
        }

        // generate model
        GLPK.glp_mpl_generate(tran, null);
        // build model
        GLPK.glp_mpl_build_prob(tran, lp);
        // set solver parameters
        iocp = new glp_iocp();
        GLPK.glp_init_iocp(iocp);
        iocp.setPresolve(GLPKConstants.GLP_ON);
    }
}
```

```

        // solve model
        ret = GLPK.glp_intopt(lp, iocp);
        // postsolve model
        if (ret == 0) {
            GLPK.glp_mpl_postsolve(tran, lp, GLPKConstants.GLP_MIP);
        }
        // free memory
        GLPK.glp_mpl_free_wksp(tran);
        GLPK.glp_delete_prob(lp);
    }

    public void callback(glp_tree tree) {
        int reason = GLPK.glp_ios_reason(tree);
        if (reason == GLPKConstants.GLP_IBINGO) {
            System.out.println("Better solution found");
        }
    }
}

```

Chapter 6

License

GLPK for Java is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License[1] as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

GLPK for Java is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with GLPK for Java. If not, see <http://www.gnu.org/licenses/>.

Bibliography

- [1] Free Software Foundation, Inc. *GNU General Public License*, 2007.
- [2] Andrew Makhorin. *GNU Linear Programming Kit*. GNU Software Foundation, 2010.
- [3] Sun Microsystems, Inc. *Java Native Interface Specification v1.5*, 2004.
- [4] SWIG.org. *Simplified Wrapper and Interface Generator*, 2010.

Index

abort, 13

callbacks, 11

class path, 8

classes, 8

classpath, 7

examples, 14

exceptions, 10

glp_java_error, 13

GlpkCallback, 11

GlpkCallbackListener, 11

GlpkException, 10, 13

GlpkTerminal, 13

GlpkTerminalListener, 13

JNI library, 6, 10

license, 19

Linux, 5, 6

output listener, 13

support, 4

SWIG, 8

threads, 13

Windows, 5–7