Getting Stanford's "Karel the Robot" to Run in Debian's Eclipse

BY TERRY HANCOCK IN HACKING 10/10/2009 PERMALINK
TAGS: ECLIPSE JAVA OPEN-COURSEWARE KAREL ACM STANFORD

I'm taking Stanford's Open Courseware "Programming Methodology" this semester, but I got stumped early on by the problem of setting up the special Stanford class libraries in my Debian-standard Eclipse installation. The instructions and files available from the website are only available for Windows and Macintosh platforms. The process is not that hard, but if you're new to Java and Eclipse (and especially if you are new to programming, as the class assumes), you'll likely be thrown by this. I couldn't find any documentation on how to do this after extensive searching, so here it is.

About Stanford's CS106A "Programming Methodology" class

After years of learning programming by doing it, with hardly any formal training, I decided it might be good to go back to the basics and take some of the theory classes I missed. Stanford and MIT both have excellent computer science classes available completely free through the Open Courseware initiative, so this is a great (and very economical) way to catch up on what you might have missed. After studying both curricula, I decided that I would start with Stanford's "more pragmatic" sequence, simultaneously taking CS106A "Programming Methodology" (taught in Java) and CS106B "Programming Abstractions" (taught in C++).

> With hardly any formal training, I decided it might be good to go back to the basics and take some of the theory classes I missed

I can get away with this because I really already know how to do programming, so CS106A is strictly review for me in terms of theory. However, Java is pretty much completely new territory for me, as is the Eclipse Integrated Development Environment (IDE) used in the classroom demonstrations.

For a long time, I avoided Java, both because I didn't really want to develop for a proprietary platform and because, with the reference implementation being proprietary and none of the free alternatives fully compatible, it was absurdly complex getting it to work as expected on my Debian GNU/Linux system.

With Sun's recent release of Java under the GPL as "Open JDK", however, Debian now has a fully functional, fully compatible Java environment (I know there's been a lot of work on alternatives over the years, but they still aren't 100% compatible and a lot of things don't run on them. The problem is Java's *huge* collection of class libraries, of course). Eclipse, of course, is also available.

> With Sun's recent release of Java under the GPL as "Open JDK", Debian now has a fully functional, fully compatible Java environment

Eclipse is very popular in the commercial development world, and I wanted to find out why. I also wanted to see if it could help me make more productive use of my programming time (and as I've written previously, it certainly does look like it will).

I'm not sure how much use I will actually get from programming in Java. Most of what Java can do, can be done with Python -- possibly with less work and greater efficiency. On the other hand, there are huge numbers of libraries for Java (although a much lower fraction of them are free software).

Karel the Robot

Stanford's Programming Methodology course starts out with a micro-language called "Karel the Robot", adapted for Java. The original Karel was a minimalist teaching language based on Pascal, but with extremely reduced syntax. Karel also operated in a highly concrete graphical microworld. As such, it was a great way to show abstract programming concepts without the language getting in the way.

> The original Karel was a minimalist teaching language based on Pascal, but with extremely reduced syntax

Today, there are a number of different "Karel the Robot" implementations, including the Java-modified "Karel" used in the Stanford class and the Python-modified "Guido van Robot" which is available as a free-standing Debian package (`gvrng` in Lenny).

The concept is surprisingly deep. I also encountered a very similar abstraction in Russell and Norvig's "Artificial Intelligence: A Modern Approach" in which a microworld called the "Vacuum World" is used to explain AI searching and behavior problems. Minus the labels, though, the Vacuum World is exactly a Karel the Robot! I'm fairly sure the vacuum world problems in the AI book can be implemented as "Karel the Robot" (KtR) or "Guido van Robot" (GvR) problems, which would save some coding!

By the way, if all you want is a Karel-the-Robot-like environment in Debian, I do recommend using "Guido van Robot". It has major advantages over Stanford's Karel:

- It runs stand-alone
- No boilerplate code is needed
- Editing of code and world is available in the GvR window (the graphical world-editing interface is considerably better than the one in Stanford's Karel)
- The "pythonic" syntax of GvR is more minimal, and thus closer to the original spirit of the Karel concept

If it were just me taking this class, and if the Karel library were the only thing missing, I would probably have just substituted GvR problems for the KtR problems, which would've been simpler than following the directions included here.

> The instructor, Mehran Sahami, is so engaging that my 11-year-old son expressed an interest in taking the class with me

My main motivation, though, is that the class is at such a basic level and the instructor, Mehran Sahami, is so engaging that my 11-year-old son expressed an interest in taking it with me (feasible, I think, since the Open Courseware structure means he can follow at his own pace). But for him, I *really* wanted the examples to work and look *exactly* as they did in the class. Mentally juggling Python and Java syntax would just be too much for him.

Finally, it isn't just Karel. In addition to the Karel the Robot introductory assignments, the Stanford course also makes extensive use of a class library from the Association of Computing Machinery, called simply `acm`.

These are even more essential to the class than Karel, and are included in the `karel.jar` file in the archives.

So I really wanted this to work. Unfortunately, the Stanford website only provides pre-built customized Eclipse IDEs for Windows and Macintosh computers! No GNU/Linux build is available. And, since the class is intended for extreme beginners, they apparently didn't think it was worth explaining how to set up the problems in a standard Eclipse installation! I searched extensively, and I couldn't find any instructions on how to do this.

Hence the documentation that follows!

These instructions should work to install Java, Eclipse, and the Stanford assignment packages, including the Karel the Robot and ACM class libraries.

> Unfortunately, the Stanford website only provides pre-built customized Eclipse IDEs for Windows and Macintosh computers... Hence the documentation that follows!

Adapting it to other GNU/Linux platforms should be fairly trivial. The only hiccup I anticipate for other platforms is that Eclipse's menus may be arranged a little differently in later versions. If you are using a different Eclipse (than 3.2.2) then some of the menu options I refer to may be in different menus or might be named differently (I have run across instructions for later versions of Eclipse that don't quite agree with how mine is set up). If so, don't panic. The options will be there, you may just have to look around a bit to find them.

Getting to Square One

Install Open JDK and Eclipse for Debian/Ubuntu systems

Nothing could be simpler! Just use the `apt-get` system to install the packages:

```
# apt-get install openjdk-6-jre eclipse eclipse-jdt
```

This will install version 6 of Sun's recently GPL-licensed Java and class libraries (which will be by far the most compatible with Stanford's examples, although you could certainly try any one of the several free software Java implementations that have been created). Debian currently has Eclipse 3.2.2, which is what the following instructions are written for. This is a bit behind the upstream Eclipse development, so if you want a more current Eclipse, you may need to install the package from source. If so, be aware that some of the following instructions will probably require subtle changes (some of the menu options have been moved around in later versions of Eclipse, but they are of course, still there).

Download the assignment skeleton from Stanford

The CS106A website has zip archives for each assignment. These are skeleton file trees containing everything you need, except for the code you are expected to write. The source files have a comment simply saying "`// You fill in this part`", which is where you write your code.

All of the examples shown here are from the `Assignment1` skeleton, but presumably the others are much the same.

Expand the archive into your choice of an appropriate working directory on your computer.

Start the project in Eclipse

There are several possible false starts at this point! I discovered a number of different ways that you can start a project in Eclipse, but they don't all do what you want for CS106A.

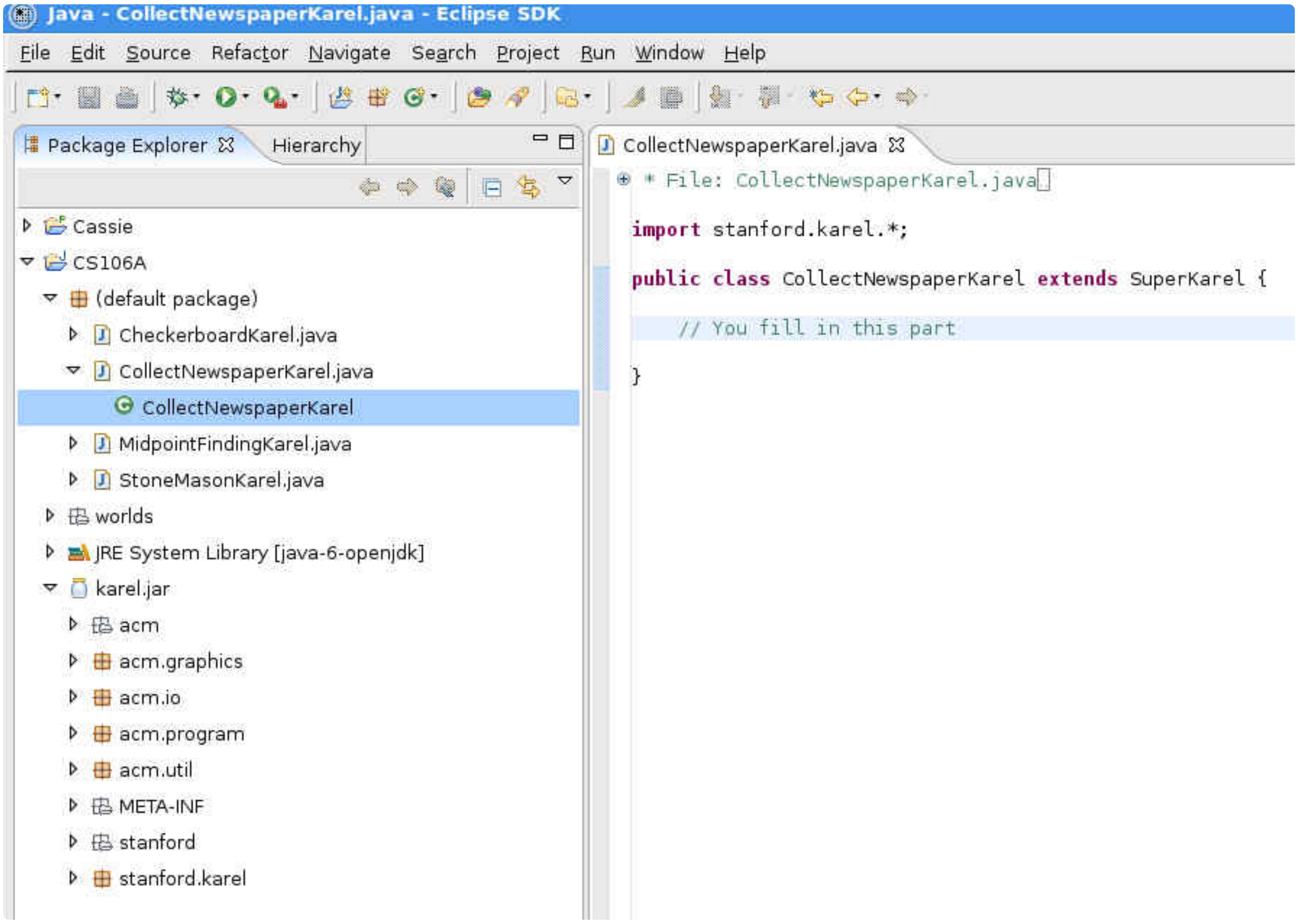
Here's what worked for me:

Start with "File"→"New Project". Select "Java Project", then click the "Next" button.

Fill in a project name. I used "CS106A" for this, although I'm not 100% sure of whether I'm going to need a separate project for each assignment or one for the whole class. The name isn't really that important, though.

Select the radio button for "Create project from existing source". Then browse to the directory you expanded the Assignment archive into. Select the "Assignment" directory itself (e.g. `Assignment1`). Click the "Finish" button to create the project.

At this point, you can look in the new project folder, and you should see something like figure 1.


Java - CollectNewspaperKarel.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
Hierarchy
Cassie
CS106A
(default package)
CheckerboardKarel.java
CollectNewspaperKarel.java
CollectNewspaperKarel
MidpointFindingKarel.java
StoneMasonKarel.java
worlds
JRE System Library [java-6-openjdk]
karel.jar
acm
acm.graphics
acm.io
acm.program
acm.util
META-INF
stanford
stanford.karel
CollectNewspaperKarel.java

```
* File: CollectNewspaperKarel.java

import stanford.karel.*;

public class CollectNewspaperKarel extends SuperKarel {

    // You fill in this part

}
```

Figure 1: Assignment 1 imported into Debian's eclipse. Note that the assignment is under "(default package)" and the karel.jar file can be expanded to show the acm and stanford.karel packages. This is evidence that you've got it loaded correctly. I've opened the CollectNewspaperKarel.java skeleton which you would use to complete the first assignment

Create a run option for it

Next we need to tell Eclipse how to run the file. As far as I can tell from watching the videos, this is what Stanford hacked the most in their "special version" of Eclipse, because this is not quite as simple as clicking "Run Karel". However, it's not that hard either.

Click on the pull-down arrow beside the "Run" icon (this is a green button with a white arrow on it in Figure 1. Eclipse is skinnable, though, so this may look different if you are using a different version). Note that just clicking on the "Run" icon itself will run the top menu entry from the pull-down, which may not be what you want. From the pull-down menu, select "Run..." to get the "Run" dialog window.

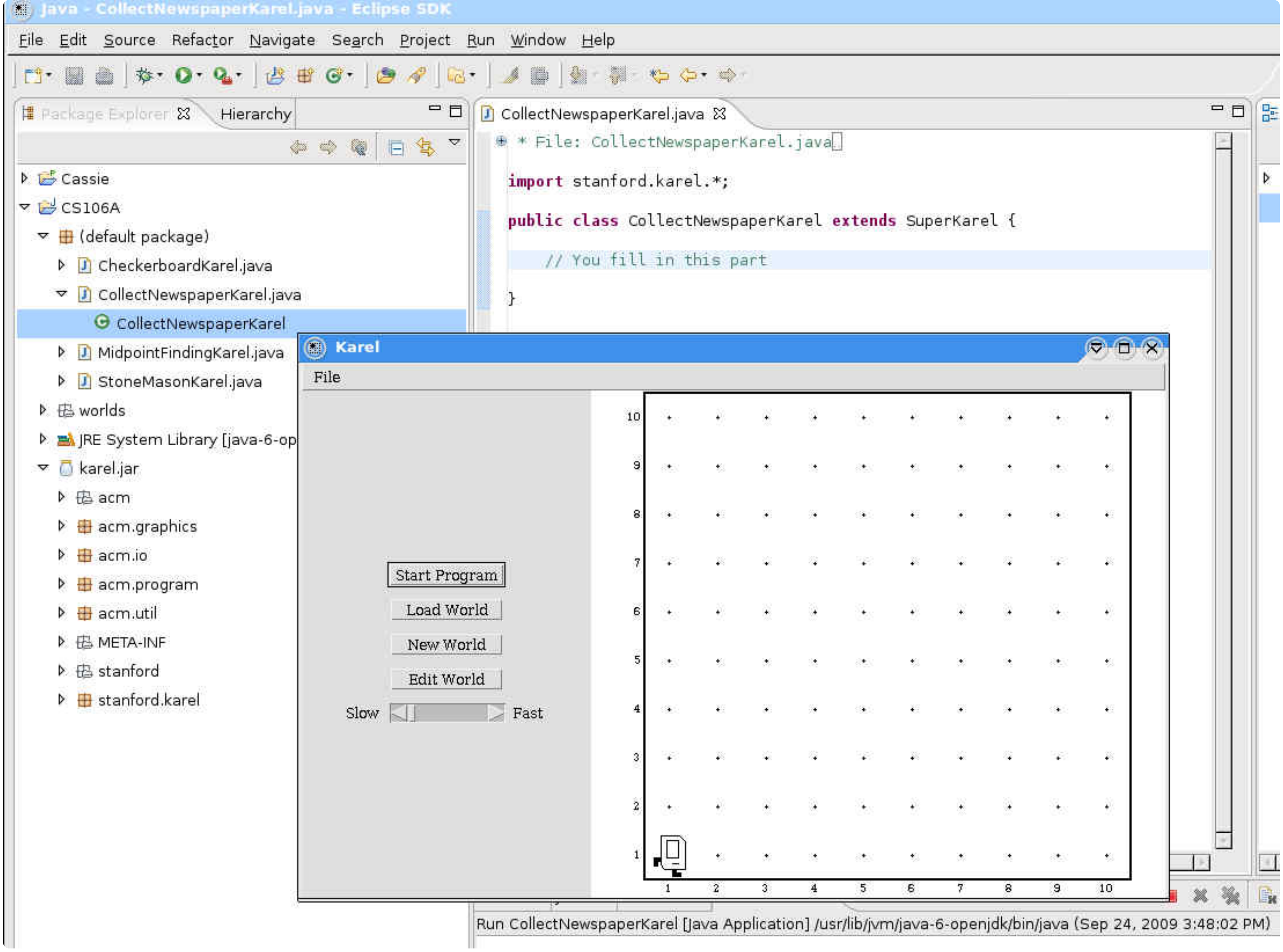
You'll see that the project is already set to "CS106A". Now add a name for the run profile. I used "Run CollectNewspaperKarel" (there may be a way to run the currently-opened file, but if so, I haven't found it -- so I'm just planning to create a separate profile for each program).

Click the check boxes for "Include libraries when searching for a main class" and "Include inherited mains when searching for a main class". The latter is probably the one that matters, because the Karel class appears to contain the dummy "main()" method. On the other hand, if you follow the class assignment, you'll create your own "main()", so you could skip this (but the testing steps below won't run properly).

Now click "Search...". Eclipse should find the main class and fill in "stanford.karel.Karel". This shows that Eclipse is able to import the Jar file and read the library, so if you see this, you can be pretty confident that you're on the right track.

Now click "Apply", and then "Run" (or you can close the window and select "Run CollectNewspaperKarel" from the "Run" pull-down, as you will do later on).

There will be a delay while Java compiles the bytecode and runs the program, and then the Karel window will pop up (see figure 2), just as it does in the class videos!

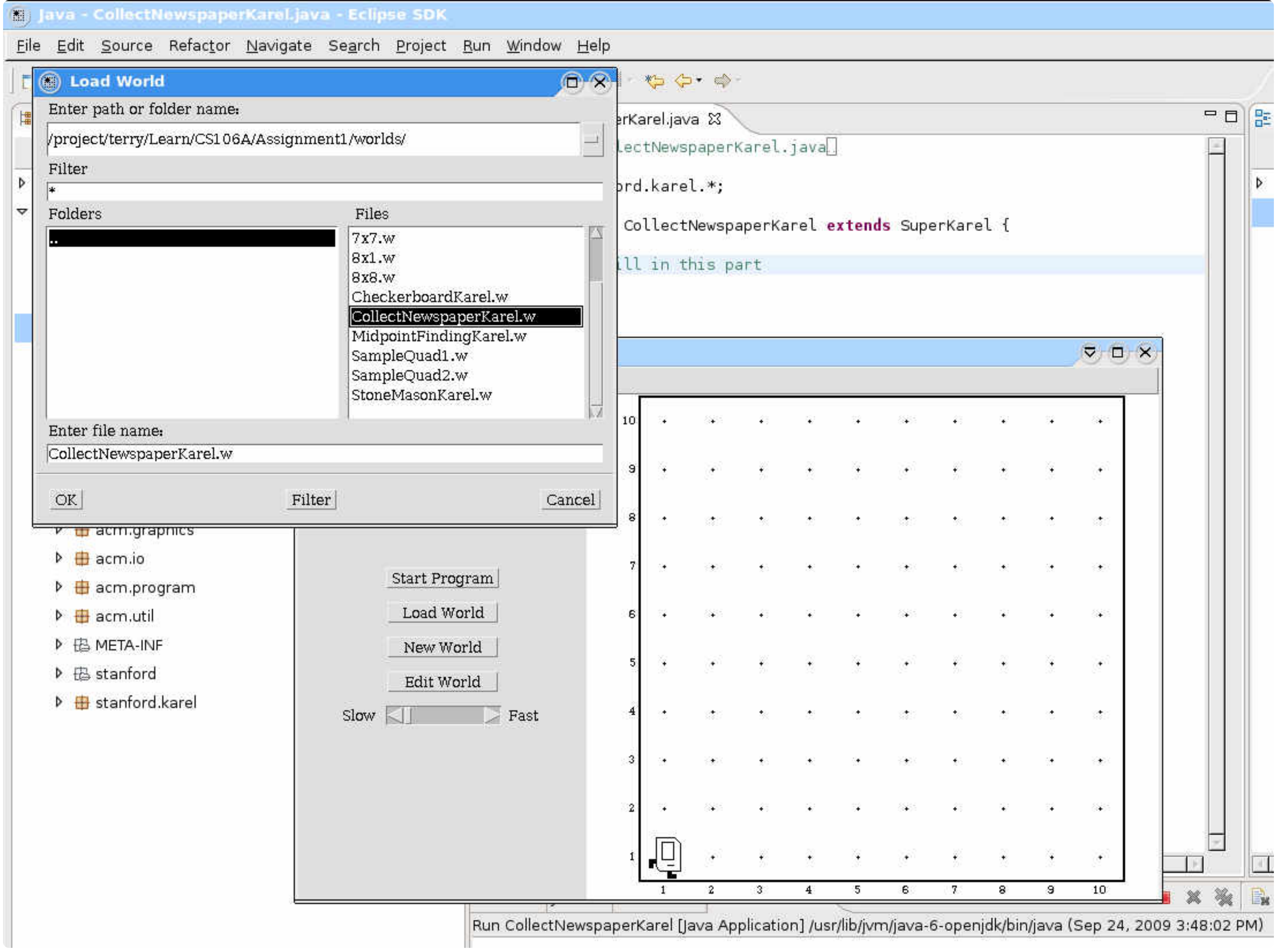


```
* File: CollectNewspaperKarel.java

import stanford.karel.*;

public class CollectNewspaperKarel extends SuperKarel {

    // You fill in this part

}
```

```
Run CollectNewspaperKarel [Java Application] /usr/lib/jvm/java-6-openjdk/bin/java (Sep 24, 2009 3:48:02 PM)
```

Figure 2: There's Karel!

Now this is not the correct scenario for the assignment, so you'll need to click "Load World" and browse to the correct world file (in this case, it's `CollectNewspaperKarel.w`). Click "Ok" to load it. See Figure 3 for the menu, and Figure 4 for the resulting world.



```
rKarel.java
lectNewspaperKarel.java
ord.karel.*;
 CollectNewspaperKarel extends SuperKarel {
ill in this part
```

Run CollectNewspaperKarel [Java Application] /usr/lib/jvm/java-6-openjdk/bin/java (Sep 24, 2009 3:48:02 PM)

Figure 3: Loading a world for Karel

Finally, you actually get Karel to go by pushing the "Start" button. At this point, if you haven't added anything to the program file, *absolutely nothing will happen!* (That's how you know it's working, right?) Seriously, though, that's what should happen, because you haven't given Karel any instructions, and the dummy `main()` method doesn't do anything.

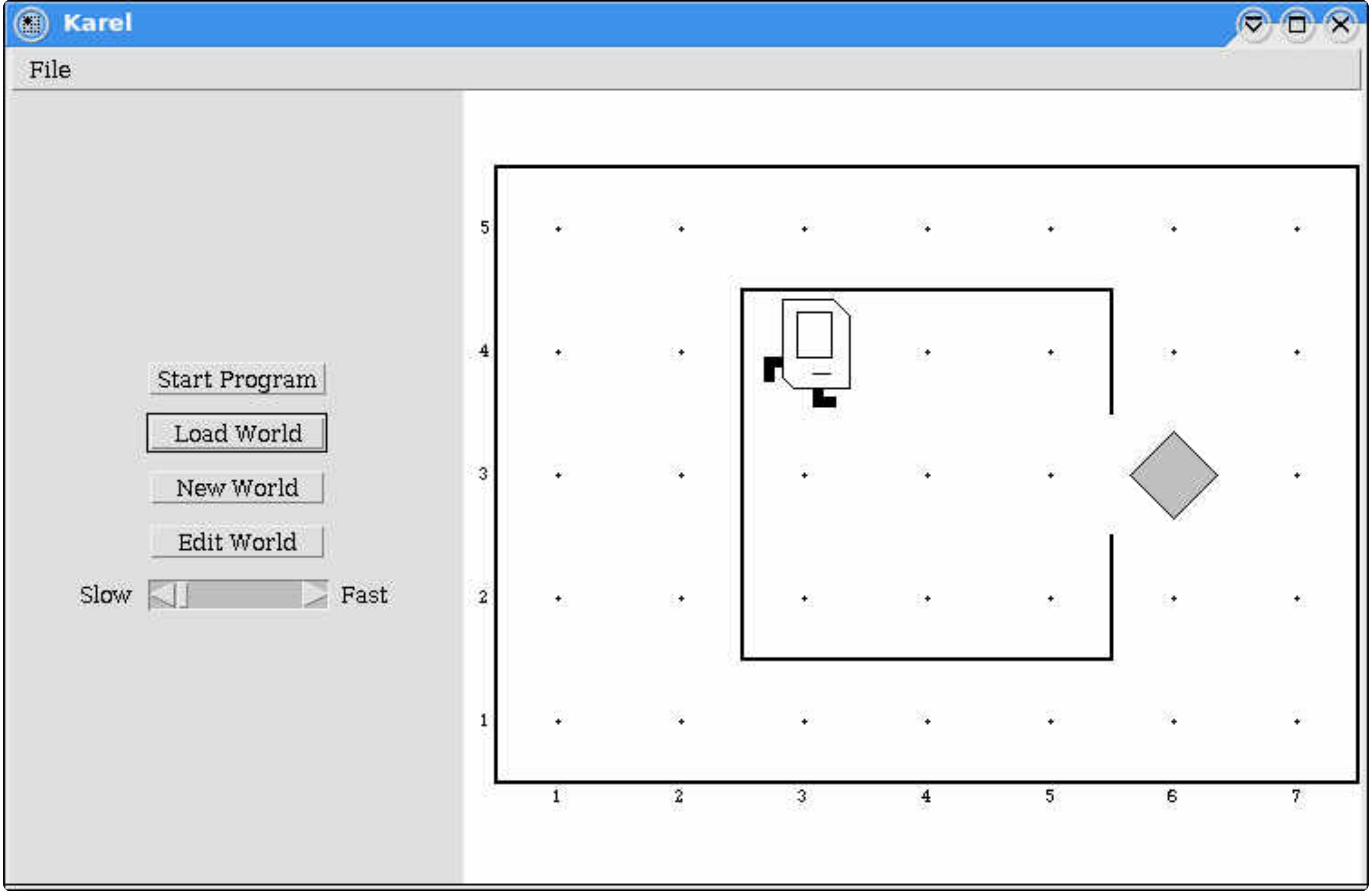


Figure 4: The CollectNewspaperKarel world. This is what you will see after loading the correct world (and also after running the program if you haven't added any code to it)

After you've added some code to Karel (see the class for what you should add), then you'll see some action when you push the "Start" button. Of course, after each run, you'll need to re-load the world scenario to reset the program for each run (this is covered in the lecture).

Karel learns Debian

That's it. Everything else should work as it does in the class demonstrations, so I will leave you to follow those for what to do with Karel. I haven't actually tried it, but it appears that the ACM examples will also work in this way, since the ACM libraries are clearly there, along with the Karel library.

Licensing Notice