The Guide to Psychophysics Programming

with JAVA

Huseyin Boyaci

September 28, 2006

Contents

1 Introduction

Today there are countless number of labs in the world where researchers investigate behavioral and neuronal responses of human observers to visual and other stimulation. In many of those labs computer controlled stimulus presentation techniques are extensively used. Numerous options lie before the computational psychophysicist, which range from easier to use and less flexible "blackbox" programs to harder to program but more flexible compiler languages. Yet most of those tools lack proper documentation, or the documentation isn't written exclusively with psychophysical programming in mind. In other words, what a young psychophysicist doesn't have as an option is abundance of resources that will guide him or her while learning psychophysical programming. This guide aims to contribute to community by providing a detailed overview of computer based stimulus presentation techniques. The guide is not a manual for a certain toolbox, it is a programming book for the psychophysicist. In this Guide, concepts and problems are introduced, and their solutions are implemented using Java platform, but the same principles should apply whatever programming language you use. Then why Java? Why not a guide on Psychtoolbox, for example?

1.1 Why Java?

On one hand there are "easy to use tools", such Presentation, which become painfully complicated once you try to implement slightly more interactive experiments. On the other hand, C or C++ with native graphics libraries let you get "closest to the metal" and virtually control everything as you like, but understanding their syntax is hard and they leave too much room for errors while coding. In that regard Psychtoolbox takes its admirable position in the center of the spectrum allowing both flexibility and ease of use. But nothing is perfect and it has its own share of shortcomings. Psychophysics programming with Java offers many serious benfits over any other option the researcher has. The biggest motivation in the beginning for me to switch to Java was "communication": I needed a platform which allowed me share my experimental designs with my collegues, I thought that I should be able to e-mail them a single file and it should run with just a mouse click whatever operating system, whatever computer architecture they are using. Moreover, I needed to be able to communicate with myself: I use a Mac in my lab and office, but Linux at home and on my laptop for fMRI experiments. I found that lack of a solution to this simple need was unacceptable. Do I have to prepare two versions of each experiment if I want to work on it both at home and in the lab? And a third one to send to my collegue, which runs on MS Windows using DirectX libraries? What about sharing it with others who visit my web page? After switching to Java, my collegues and I found solutions to problems in our experimental designs literally over night, over a few e-mails. I can now convert my experiments into Java applets by changing just a few lines in my code and place them on web page. Advantages of Java are not limited by ease of communication. Due to its object oriented design and technologies, Java provides a very effective platform of programming and development. It boosts the *productivity* in several other ways as I will explain in more detail below.

1.2 What Java offers to the Psychophysicist

First, the key technologies that concern the computational psychophysicist.

Full Screen Exclusive Mode and new Java2D API

Full Screen Exclusive Mode (FSEM) is introduced to core Java platform with version 1.4. Sun probably had the gaming industry in mind when they introduced it, nevertheless this offered a marvelous opportunity for psychophysicists. This mode and the new Java2D API, allow capturing the entire screen, provides tools for double buffering and proper vertical synchronization, all of which are essential for psychophysics programming. Depending on the OS/architecture, Java may use different strategies in FSEM. For instance on MS Windows it will most probably use the DirectX routines, on Mac OS X it uses native OpenGL libraries, on Linux/Unix it uses X11 libraries but can be instructed to use OpenGL libraries instead.

Architecture specific details are hidden while low level access is still possible

Engineers at Sun do the hard work of hiding the layer between you and the low level interaction with the hardware. This way your code behaves in the exact same way on different platforms even though the Java interpreter may do completely different jobs. On the other hand Java doens't lack ways for low level hardware access for visual displays.

3D graphics

For 3D graphics there are a few options. Among the most popular ones are Java-3D, which provides a high level scene graphics API, and JOGL, which stands for Java bindings for OpenGL. These two options offer different levels of flexibility and level of hardware access. There is a strong community support behind those open source projects.

Converting your experiments into Java Applets

It is a matter of changing a few lines to convert your experiments into Java applets, or web applications (called as WebStart technology).

Pack and send, share your work with collegues

You can package your experiment in a so called jar file and send to your collegues. Whatever platform they are on, they can run the experiment with a mouse click.

Networked experiments

Java platform has excellent support for network programming, providing all the necessary tools for networked experiments.

Multithreaded programming

In psychophysics or neuroimaging experiments, particularly with animations and simultaneusly running task trials, it is desirable to have a program which can run multiple tasks concurrently. Core Java platform provides an extensive collection of tools for multitasking programming, also known as multithreaded programming, or parallel programming.

Matlab and Mathematica interface

It is a fact that many computational psychophysicists rely on Matlab or Mathematica for programming. Those who like to continue using their preferred environment can do so, because both Matlab and Mathematica offer allow interaction with Java objects.

Next, more general advantages of Java platform for improving productivity.

Cross platform portability

Even though it is not as perfect as it was promised, Java *is* platform independent (OS and architecture) to a great extent. The behavior of my experimental programs were always identical on all three systems I used (Linux, Mac OS X, MS Windows). This improves productivity and communication:

- improved productivity at programming phase - easy team work: Researchers working on their preferred OS can still work on the same piece of code together.
- Improved productivity at application phase: Researchers can work on their preferred environments and can still run their experiments on different systems in their labs. For example, they can prepare an experimental program to measure the behavioral response to a visual stimulus in their psychophysics lab on Mac OS X and use the same code on a PC to determine the behevioral effect to the same stimulus inside the scanner before an fMRI experiment.
- Code sharability: Once written a class (objects belong to classes), you can use the same class again and again (see also Object Oriented Programming below), moreover anybody else can use the same code on their own platform. This is actually a great asset because it means that you can get advantage of many existing classes and build upon them new tools.

Object Oriented Programming advantages

- Extendibility and reusability: OOP by design targets re-using and/or extending existing classes. Mind you that this is not just re-use of your functions in a more traditional language. You will find many examples where we take full advantage of OOP in this guide.
- Hide the implementation: By hiding the implementation you can write code which does not break the client's program with every new version.
- Strict type checking: Once it compiles a Java program usually runs without any problem, you will not get memory errors as you would in C. This is because Java compiler is extremely strict in variable typess. Java won't do the conversions for you, it wants *you* to do the "type casting" to see that you know what you are doing.
- Exception handling: In languages like C, and Fortran there is no real exception handling. You invoke a method and if it fails your program crashes. Or they may return a funny result which indicates that the method failed to accomplish what you asked from it. Java provides a much more advanced approach to handling such errors through its Exception class. This way you can put the mechanisms in your code which will fix the problems during its execution. Even if it is unfixible, you can instruct your code to gracefully exit rather than crash.
- Automatic memory management: Java has an automatic "garbage collector". You usually do not have to worry about memory allocations and deallocations. In case you have special needs you can still manually manage your memory.

Easier to code

- Extensive API (collection of thousands of classes) in the core Java platform. This eliminates to re-write many classes and their methods, because as discussed above using existing classes is the nature of OOP.
- Excellent documentation of the core classes. You can easily reach the most current documentation online at java.sun.com. The documentation is superior to even older compiler languages such as C.
- High level: One line of code does a lot for you when you use existing classes as building boxes. This makes your code less complicated, easier to read, and less prone to errors.
- New generation of Integrated Development Environments (IDEs) provide very easy and effective programming environment. One commonly used IDE is Eclipse. The editor of Eclipse flags the mistakes as you are typing your code and suggest possible options to fix those mistakes. You do not need to remember which libraries to import, because Eclipse will do that for you automatically, and suggest automatic code completions.
- Apart from the IDEs, the faithful user can still use Matlab or Mathemetica as interface to Java objects.

1.3 Comparison to other tools, particularly to Psychtoolbox

Java is a programming platform, which is as complete as possible for the modern programmer, covering a whole range of areas, from parallel programming to network tools. It is not fair to compare Matlab to Java in those areas, because Matlab is not a real programming platform, it is an advanced tool for numerical analysis of small technical problems. It is quicker to learn, quicker to implement small programs in Matlab. But it does not provide the same flexibility and variety of tools as Java. Psychtoolbox was motivated by the fact that it was hard to learn the syntax of C and graphics libraries in the absance of development environments as we have today. It is also hard to code in C because it leaves too much room for errors. Therefore there emerged a need to place a layer between C and the novice programmer. Psychtoolbox has been that layer for many years. But now Java opens another door eliminating the need to that extra layer, because it offers not only a much advanced programming platform but also a programming language, which is easy to code in.

1.3.1 Advantages

- Java is platform independent, Psychtoolbox is not: Matlab runs on multiple platforms, but Psychtoolbox does not. Even on the platforms that Psychtoolbox runs you still have to make some modifications in your code if you want to port it to another platform (actually the platforms supported by the latest version are limited by two, MS Windows and Mac OS X).
- You can convert your experiments into Java applets, there is no such option with Psychtoolbox
- You can prepare a single executable file of your experiment and send to collegues, they can then run it with a mouse click. There is no such possibility with Psychtoolbox.
- Psychtoolbox doesn't provide tools for 3D rendering. Java offers multiple options, at multiple levels of ease and levels of programming.
- Java has extensive coverage in threaded programming and network programming areas, Psychtoolbox doesn't.

- Object oriented programming, and other advanced programming techniques inrease productivity. Matlab/Psychtoolbox doesn't provide any such advanced programming techniques.
- New generation IDEs offer much superior programming environments compared to Matlab's editor. One such IDE is an open source project called Eclipse.
- The underlying graphical details of Psychtoolbox are hidden away from the researcher. One can in principle download and inspect the C source code, but it is not such an easy task to fully understand what is going on. You would also need to understand the native graphics libraries on the two OSs that it runs. This is a discomfarting solution for a researcher. Sooner or later the researcher will need more independence then that.
- Java is freely available (though its source is not completly open.) Matlab needs a licence to run. You either purchase a standalone licence, or use a network licence if your institute has one. The first option is more reliable but not cheap. The second one is free for you, but frequently suffers from number of user limitations and network distruptions. A lab under tight budget could reduce the costs considerably by using a Java system (one could further reduce the costs if the system is built on Linux).
- Backward portability: It is unlikely that a program written for an older version of Java platform breaks when run with a newer version. You can upgrade your Java platform to the latest version without the fear of breaking older code. Matlab/Psychtoolbox frequently breaks older code.

1.3.2 Disadvantages (mispercieved and real)

It is easier to program in Matlab, OOP and Java are difficult

It is definetaly simpler to program small projects in Matlab as a starter. But this simplicity doesn't scale to larger projects and it does not provide any of the productivity benefits that OOP offers in the longer run. Yes, if it is a highschool intern in your lab one may then argue that learning Java is going to waste his or her time. But you can provide a framework for experiments in a specialized class and let your student use that class, even create objects of those classes from within Matlab or Mathematica if they prefer that option. I am building such a framework in this guide and it is successfully used by inexperienced students easily. As you will see starting with Chapter 2, using such a framework can be as simple as using Psychtoolbox, if not easier! Moreover, Java and OOP are taught in courses in many Computer Science programs both at undergraduate and graduate levels. New generation of students do come to our psychophysics labs with that knowledge and are usually more keen on using Java rather then Matlab.

Community support: Many Psychophysicists use Psychtoolbox

Psyctoolbox/Matlab option is extensively used by our community. This naturally leads to a strong community support. For the moment there is no such community for Java. Your source of support will probably be the Java gaming community for now.

Integration with analysis programs: I run my experiments with Psychtoolbox, analyze them with Matlab

If you are doing your analysis using Matlab, it may seem like it is a better idea to limit yourself only to that program and use Psychtoolbox. But you can use Java from within Matlab/Mathematica. As for the pure Java programmers, I think soon there will be enough numerical computation tools, if not already are, available to perform any important statistical analysis with Java. The number of such methods may exceed the number that exists in Matlab. The widely used gnu Scientific Library (by computational physicists for example) may soon be ported to Java. I will provide some numerical algorithms in the end of the Guide.

Performance

Java *is* slower than low level compiler languages such as C. However the difference is shrinking with introduction of new technologies, notably the Hot Spot Technology. Nowadays the speed of Java is about 2/3 of C. And certainly Java is not inferior to Matlab in terms of speed. I have not met a situation where my experimental code suffered from cpu or gpu speed. Only in numerical computations I found C superior to Java (less than one fold), Java superior to Matlab usually a few folds!

Hardware manufacturers provide interfaces with Psychtoolbox, not with Java

This is the only real disadvantage of using Java in my opinion. Many companies provide interfaces using Matlab/Psychtoolbox to use their devices. For example CRS provides Matlab functions for Bits++ which integrate with Psyctoolbox. But as you will see in Chapter A, it is not all that difficult to write your own interface if you know how to program independently and as the number of users increase there will be many classes available on the internet for such purposes. Since Java allows low level access it is usually possible to build your own interfaces. Besides one would expect the manufacturing companies should soon notice the advantage of building their tools on Java to reduce their costs, just like Matlab did.

1.3.3 So, which one to choose?

In summary, Psychtoolbox was a temprory solution. The C compiler language was hard to learn and hard to code with, there was a need for the community to find an easier solution. Psychtoolbox provided that solution using an unlikely ally Matlab, which is actually an advanced numerical analysis tool, not a fullscale programming environment. But the compilers constantly advance, the IDEs get better and better with large community support and the need for such temprorary solutions is eliminated.

All of the disadvantages that Java suffers from will soon change. As the number of psychophysicists using Java increases, a community will form. Besides, there is already a very strong community support behind Java2D, java3D, and JOGL. Even though they mostly have gaming, not psychophysics programming in mind I benefited a lot from them.

However, if you already have library of functions to run your experiments using Psychtoolbox and rely heavily on Matlab for analysis, it wouldn't make sense to port all those to Java now, but I would recommend implementing the new ones in Java. If you are newly creating your library or faced a situation where Psychtoolbox isn't sufficient any more, switch as soon as possible to pure Java programming.

Another option is using Java objects from within Matlab of Mathematica. Even if you like to continue using your preferred platform, Matlab or Mathematica, you can still create and use Java objects from within them. See Section 1.4.2 below. Using Java objects has one distinctive advantage: they are platform independent! Matlab and Mathematica both have multiple OS versions, for the end user they are platform independent. Whereas Psychtoolbox fails to be completely platform independent (and built only for Matlab). When you replace Psychtoolbox with Java objects you would virtually be completely platform independent even if you are using Matlab or Mathematica for development.

1.4 How to use the Guide

1.4.1 The Guide as a programming book

Throughout the Guide I introduce the tools of Java, which are useful for psychophysics programming. I start with "Hello Psychophysicist" chapter where I introduce the essentials of displaying visual stimulus on displays. In the following chapters I introduce further details for psychophysics programming, such as

multithreaded programming, event handling and accurate timing. While introducing the tools from core Java interface, I also build up custom classes, which are suitable for reuse to design real psychophysics experiments. I do not shy away from the details and go to greater lengths to explain them. I provide simple "test" programs, some of which are replicates of real experiments I actually used in my own research, some are small tests of the functionalities just introduced in the chapter. In Appendix B, I introduce some numerical analysis methods. These are simple but very useful classes. One of them provides methods for computing population statistics like mean, varience and others. Another class provides methods for function minimization.

The reader who wants to learn and program in Java platform should read the entire Guide. Chapters 2 to 9 are essential for the fundamentals. The remaining chapters can be read selectively. For instance if you have a multiple display system you should read Chapter 10, as well. Converting your experiments to Java Applets is discussed in Chapter 11. Appendix A explains how to utilize Bits++ using Java. There is no real prerequisete but some modest Java or object oriented programming knowledge would be helpful. I try to reference further readings if I use an advanced programming technique.

1.4.2 The Guide as a manual to the tools developed

In the end of the guide, in "Chapter C: Demos, installation instructions, and Manual pages", I provide a glossory of all the classes and methods I built up, and show how to use them (not how they work). I point to the chapters where the details are discussed in greater lengths. Consider this as manual pages for the classes. I present numerous demos in that Chapter. Those demos are based on the test programs from earlier Chapters. This way those who want to get more detailed information could turn to those chapters. I implement the same test programs in Matlab and Mathematica, as well.

1.4.3 List of classes and methods built in the Guide

FullScreen

Initiates FSEM window, and provides methods to present stimuli on the secreen and to collect observer response

- FullScreen() - initiates FSEM, inherits from Java core class JFrame hence all its mehods are also available
- displayImage() - diplays image
- displayText() - displays text
- blankScreen() - blanks the screen
- setNBuffers(), getNBuffers() - set/get the number of video buffers
- updateScreen() - updates the screen by moving the back buffer (if it exists) to front
- setBackground(), getBackground() - set/get the background color
- hideCursor(), showCursor() - hide/show cursor
- closeScreen() - terminates FSEM
- getKeyPressed(), getKeyTyped(), getKeyReleased() - obtain observer's key presses etc.

- getWhenKeyPressed(), getWhenKeyTyped(), getWhenKeyReleased() - obtain the time of key presses
- flushKeyPressed(), flushKeyTyped(), flushKeyReleased() - clear the que of key press events
- isFullScreenSupported() - check whether FSEM is possible on your system
- isDisplayChangeSupported() - check whether display mode change is supported on your system
- isDisplayModeAvailable() - check whether a particular display mode available
- setDisplayMode(), getDisplayMode() - set/get display mode (resolution etc.)
- getDisplayModes() - obtain all available display modes
- reportDisplayMode() - obtain a readable version of current display mode
- reportDisplayModes() - obtain a readable list of all available display modes

NormalWindow

Similar to FullScreen, except stripped of Full Screen Exclusive Mode specific methods. It opens a normal window instead of FSEM window. By replacing your FullScreen object with NormalWindow object you can get an application to put on your web page. (See Chapter 10.) It allows user to decide on various rendering strategies. (FSEM always uses active rendering, in normal window you can decide between passive and active rendering.)

- NormalWindow() - initiates a normal window, inherits from Java core class JPanel therefore it has all its methods available
- setPassiveRendering() - set passive rendering true or false (see Chapter 10)
- isPassiveRendering() - obtain the current state of passive rendering, true or false
- displayImage() - diplays image
- displayText() - displays text
- blankScreen() - blanks the screen
- updateScreen() - updates the screen by moving the back buffer (if it exists) to front
- setBackground(), getBackground() - set/get the background color
- hideCursor(), showCursor() - hide/show cursor
- getKeyPressed(), getKeyTyped(), getKeyReleased() - obtain observer's key presses etc.
- getWhenKeyPressed(), getWhenKeyTyped(), getWhenKeyReleased() - obtain the time of key presses
- flushKeyPressed(), flushKeyTyped(), flushKeyReleased() - clear the que of key press events

Clut

Provides methods for (inverse) color look up operations. See Chapter 8 and Chapter A.

- Clut() - creates a CLUT object, and sets the values in the look up table either by reading a file holding the table or from an array
- setClut() - sets a the values in the look up table
- lum2Pix() - finds the pixel that results in a luminance value closest to the desired luminance value
- pix2Lum() - returns the luminance that results from the given pixel
- getMaxLum() - get the maximum luminance value in the table

BitsPP

A class which has methods for using Bits++. It inherits from FullScreen so it has all its methods and some additional methods needed to communicate with Bits++ device. See Chapter A.

- BitsPP() - initiates a Full Screen Exclusive Mode window and initializes Bits++ look up table
- initClut() - initializes Bits++ look up table
- setClut() - set the Bits++ look up table
- displayImage() - diplays image by using Bits++ look up table
- displayText() - displays text by using Bits++ look up table
- blankScreen() - blanks the screen by using the Bits++ look up table
- closeScreen() - terminates FSEM and leaves the screen in a nice condition by setting a linear Bits++ look up table

BitsPPFake

Similar to BitsPP class, except it is fake. Designed mainly for development when away from your Bits++ device.

- BitsPPFake() - initiates a Full Screen Exclusive Mode window and initializes Bits++ look up table
- initClut() - initializes Bits++ look up table
- setClut() - set the Bits++ look up table
- displayImage() - diplays image by using Bits++ look up table but only its most significant 8 bits

Sample

This class provides methods to compute statistics of sample populations.

- More information Coming soon!.....

Distributions, FunctionMulti, FunctionSingle, Integration, Minimize, MLE_TwoAFC, SpecialFunctions etc.

Classes which provide numerical analysis methods.... More information Coming soon!....

1.5 Setting up the Java platform

The latest Java development kit (JDK) can be downloaded from `http://java.sun.com` (current latest stable version is 5.0. Note: you should get JDK not JRE!) That site hosts versions for various operating systems, including MS Windows and Linux. I recommend installing the documentation package, as well. There may also be an external link for the Mac OS X version. The examples in this guide requires at least version 5.0. Apart from the JDK, you may want to install a development environment. One such environment is Eclipse. I will provide more information on Eclipse below. Eclipse has versions for all three OSs and is an open source project, available freely.

1.5.1 MS Windows

The installation on MS Windows is usually straightforward. However, if you had an older version of JDK you may want to it after installing the newer version.

1.5.2 Mac OS X

The Mac OS X version of Java 5.0 can be found at http://www.apple.com/downloads/macosx/ `http://www.apple.com/downloads/macosx/` by searching for Java SE 5.0. Get the latest update. On Mac OS X (Tiger) the default JDK is still version 1.4 (as of September 2006, no version 5.0 for Panther is available). When the 5.0 version is installed the default Java compiler and virtual machine are not automatically updated. To fix that you should run the preferences utility under /Applications/Utilities/Java/J2SE 5.0/ directory. If you like to inspect further, Java Virtual Machine (VM) is located under /System/Library/Frameworks/JavaVM.framework.

1.5.3 Linux - Fedora Core 5 (FC5)

Linux installation on some distributions may need extra work. Please consult your distribution's documentation. Below I will discribe how to set up a Fedora Core 5 system for psychophysics experiments.

Installing Sun's JDK

Due to its closed source and its license Sun's Java Development Kit is not included in Fedora Core Distribution. Instead FC5 ships with gnu version of Java, which is completely compliant with Sun's version but currently lacks necessary graphical tools for psychophysics experiments. Therefore you first have to install Sun's JDK. There are various ways of installing Sun's JDK. One way is using the online FC5 repositories. These repositories provide pre-compiled binaries and places them in proper locations on your system and performs necessary 'alternatives' commands.

Other option is manual installation. First get the latest version from http://java.sun.com, get the beta of version 6.0, not 5.0 because 6.0 works better in FSEM on Linux systems. However, if you would like to use Matlab for development you should install version 5.0. Download the "Linux self extracting file" not the rpm package. After download, open a terminal and become root (su -), move the file to /opt directory

```
mv jdk-6-beta2-linux-i586.bin /opt
```

change directory to /opt

```
cd /opt
```

change the mod of the file to executable,

```
chmod +x jdk-6-beta2-linux-i586.bin
```

and then execute the program

```
./jdk-6-beta2-linux-i586.bin
```

this will install the platform. After install succeds, you should tell your system that the Sun's JDK/JRE is your primary java platform, not the gcc-java which is the default package in FC5. Create a symbolic link

```
ln -s jdk1.6.0 jdk1.6
```

install the new JDK as java alternatives

```
/usr/sbin/alternatives --install /usr/bin/java java /opt/jdk1.6/bin/java 2 \
--slave /usr/bin/javac javac /opt/jdk1.6/bin/javac --slave /usr/bin/javadoc \
javadoc /opt/jdk1.6/bin/javadoc --slave /usr/bin/jar jar /opt/jdk1.6/bin/jar \
--slave /usr/bin/javaws javaws /opt/jdk1.6/bin/javaws
```

configure the java alternatives

```
/usr/sbin/alternatives --config java
```

choose the newly installed JDK as the primary choice. Check the alternatives to java

```
/usr/sbin/alternatives --display java
```

Because of the symbolic link, you don't need to install new alternatives each time you install an update to JDK/JRE. For browser (firefox) plugins to work, do the following as root

```
ln -s /opt/jdk1.6/jre/plugin/i386/ns7/libjavaplugin_oji.so \
/usr/lib/mozilla/plugins/libjavaplugin_oji.so
```

Installing Eclipse

FC5 includes a natively built Eclipse package. You shouldn't download and install Eclipse from eclipse.org, instead use the "Add/Remove Software" under Applications menu. Click on Applications→Add/Remove Software. Click on Development and check the box next to Eclipse, then click apply.

1.6 Getting the source code for the Guide

The source code of the programs in this guide can be found following the links at `http://tc.umn.edu/~boyac003/Guide`. You can either download individual programs or download the compressed package. After downloading and putting them in a directory, follow the directions in the next section to compile and run the programs.

Those readers, who would like to first try the demos and install the final version of all tools in a package called PsychWithJava can find the instructions to do so in Chapter C: The Guide to Psychophysics programming with Java: Manual pages and demos.

1.7 Development environments

1.7.1 Using command-line tools

This is the harder way of development, nevertheless it is very useful to know in case your options are limited, for example on a colleague's computer who doesn't have an advanced IDE installed. As an example, I will show how to compile and run the sample code from Chapter 2. First download all the files from Chapter 2 into a directory. Open a terminal, and change directory to PsycWithJava/ch02/, where PsycWithJava is the directory where you downloaded the code, type

```
cd PsycWithJava/ch02
```

and press enter (if using MS Windows replace the slash with back slash). Then type

```
javac HelloPshychophysicist.java
```

and press enter. This will "compile" the source code. After this you will see couple of something.class files in your directory. If there were no errors during the compilation, type

```
java HelloPsychophysicist
```

and press enter. You should now be running the first example of the book detailed in Chapter 2. Note that, you also need FullScreen1.java file in the same directory. Even though you didn't compile FullScreen1.java, the compiler still compiled and produced FullScreen1.class file. Because HelloPsychophysicist uses FullScreen1 and the compiler automatically compiles other necessary files in your directory.

To edit your code you can use any of your favorite editors, for example vi under Linux/UNIX systems.

1.7.2 Using Eclipse, an Integrated Development Environment (IDE)

Integrated development environments (IDE) certainly make the development phase easier and boost productivity. One of many such environments is Eclipse, which is an open source project and available at `http://eclipse.org`. I will demonstrate how to use Eclipse here. First install the latest version (3.1 currently) for your OS (Fedora Core 5 users, see section 1.5.3 above for installing Eclipse). If you haven't yet downloaded the source code of The Guide, download it as explained above in Section 1.6.

Start Eclipse. (The first time it is invoked Eclipse offers a Welcome screen, it is possible to use tutorials linked from that welcome screen.)

Click on the File Menu. Click New →Project. A "New Project" window will appear. Choose "Java Project" and click "Next" (Fig. 1.1(a)). "New Java Project" dialog should appear (Figure 1.1(b))

In the "New Java Project" window, inspect the radio buttons under JDK Compliance (see Figure **??**). If the default compiler compliance isn't set to 5.0, click on "Configure default". "Properties" window should appear, using it set the compliance level. In case 5.0 is not listed among the available compilers, you may need to update your configuration as follows: Close the Properties window and New Java project window. Go to Window →Preferences, navigate to Java →Installed JREs using the list on the left column (Figure 1.2). If JRE 5.0 (or 1.5, they actually are the same thing, i.e. version 5.0 = version 1.5) is not among the installed JREs, click on Add button and add the directory where JRE 5.0 is installed. Make sure that the box next to JRE 5.0 is checked (FC5 users: choose JRE 1.6 as default, not 1.5). Finally go to Compiler item in the Preferences dialog and set the default compiler to 5.0 (Figure 1.3.)

If you had closed the "New Java Project" window open it as described above. In the "New Java Project" window type "Chapter 2" for Project name, click on the radio button next to "Create project from existing

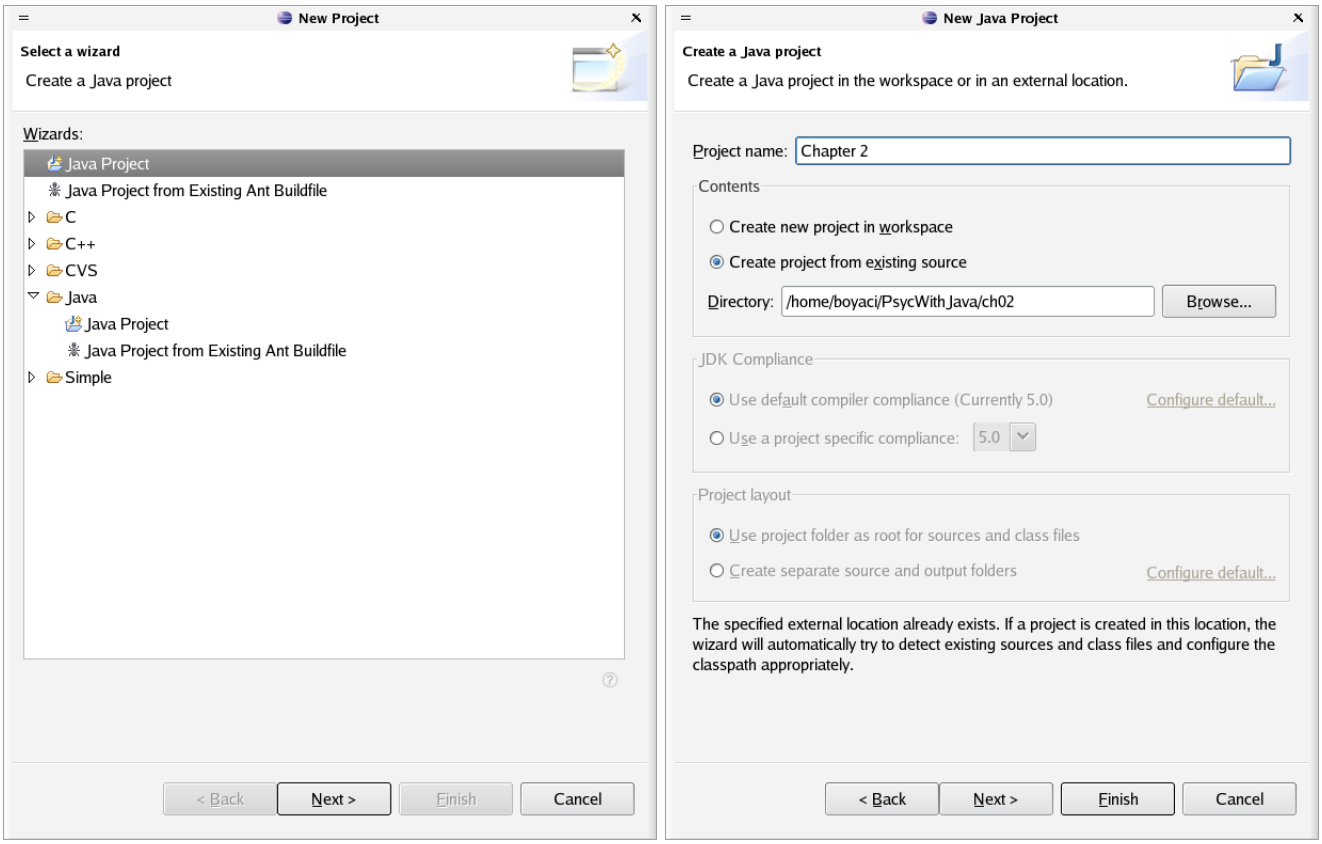

Figure 1.1: Left: New Project window. Right: New Java Project window

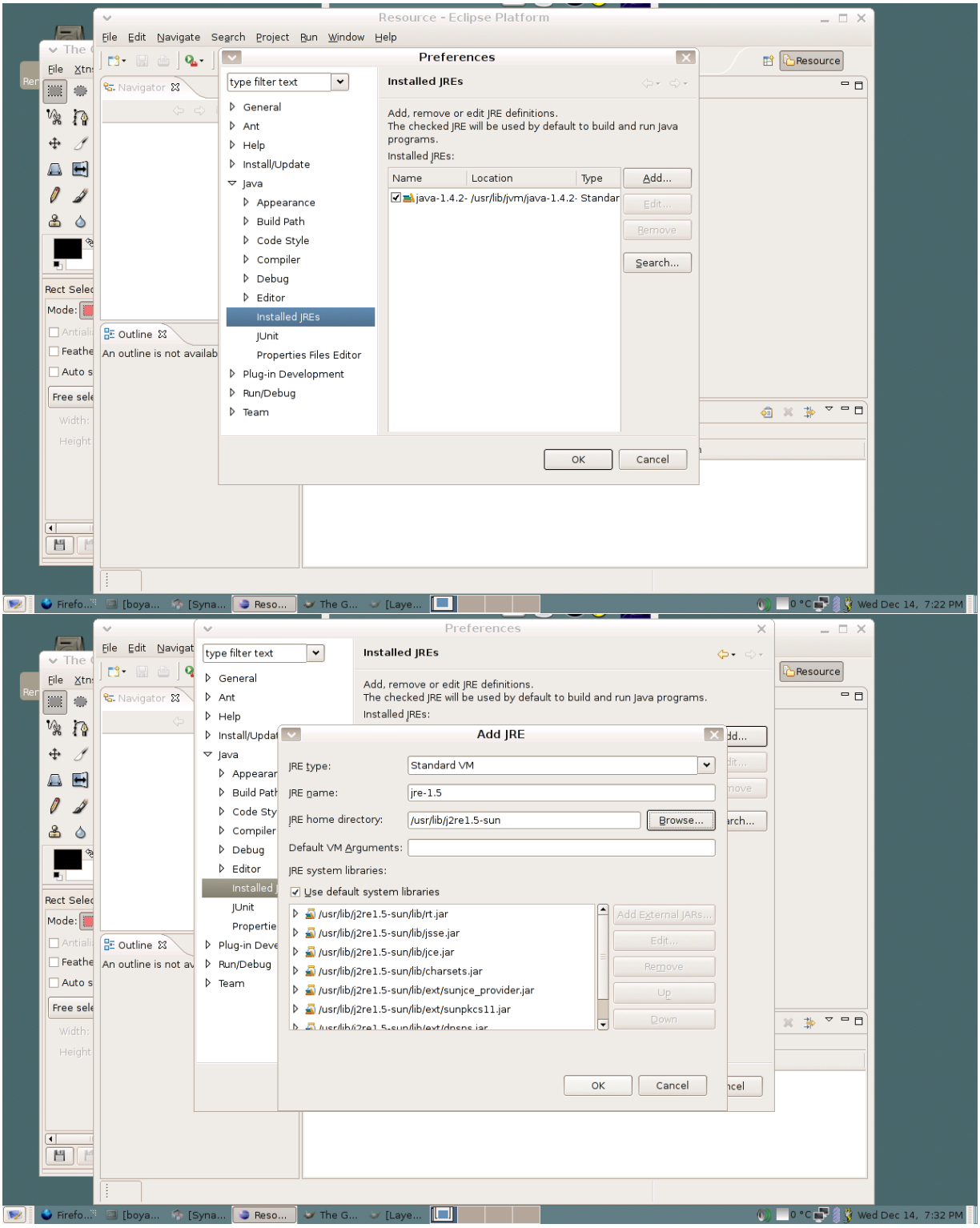

Figure 1.2: Setting up the Installed JREs.

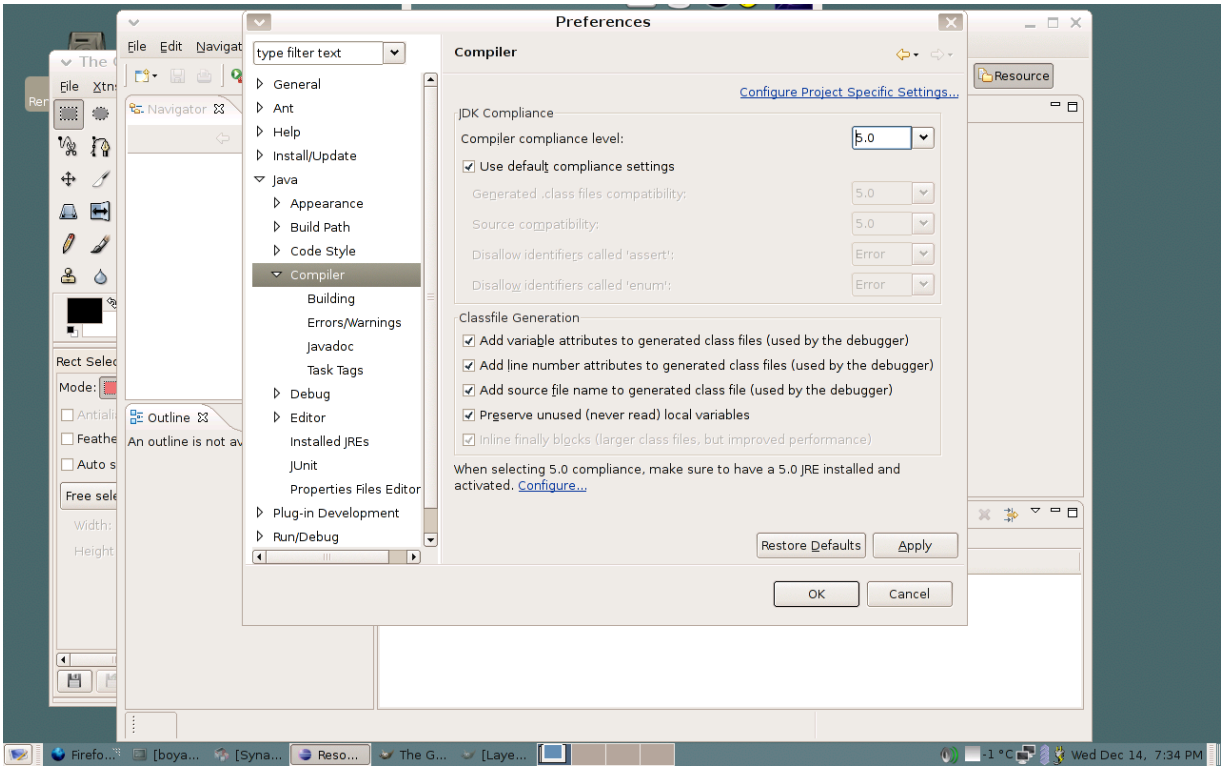

Figure 1.3: Setting the compiler compliance level.

source" and type in the full path to Guide/ch02 (this should be directory where you downloaded the source code), or alternatively click on "Browse" button to choose the directory. Click "Finish" (Fig. **??**).

Now the project "Chapter 2" is created. Click on the triangle next to Chapter 2 inside the "Package Explorer Pane", and then on the triangle next to "(default package)". Next double click on "HelloPsychophysicist.java". This will open the listing of the program in the editor pane. You can edit the code and save it by clicking on the "Save" button (the conventional floppy disk icon) or by choosing File→Save. To run the program, Right-Click (CTRL-Click on Mac OS X) on "HelloPsychophysicist.java" at the "Package Explorer Pane" and choose "run as"→"java application". You should now be running the first example of this guide. In case there are errors in the code, Eclipse flags them with "red" cards on the right side of the editor. Carefully inspect and fix them.

1.7.3 Using with Matlab

It is possible to create Java objects from within Matlab. You can also invoke methods of Java objects, which means that you can use the tools I will create in this guide from Matlab. I provide examples of doing this in Chapter C.

1.7.4 Using with Mathematica

Same as Matlab, it is possible to create Java objects and invoke their methods from Mathematica. Examples can be found in Chapter C.

1.8 Further readings

For Java platform in general I recommend the following books:

- Core Java, Volume 1 Fundamentals and Volume 2 Advanced Features, 7th edition (J2SE 5.0) by Horstmann and Cornell.
- Thinking in Java, by Bruce Eckel.

For Java2D and other graphics tools

- Killer Game Programming in Java, by Davison.

Other useful community sources

- Java2D forum: http://forums.java.net/jive/forum.jspa?forumID=69