# INTRODUCTION TO JAVA PROGRAMMING

 Java is a programming language that is designed to allow software developers to quickly and easily create applications. In this session, you're going to learn how to program using some custom software that allows you to simulate robots using Java.

**Step 1: Download The Starter File**

 You can find the web page for this course by searching for “Panitz BIT 115”, following the link to my home page, and following the link near the top of the page that reads “CCC Open House”.

 On that page, find the link that reads **Starter File**. Use your mouse to RIGHT click on that link, meaning that you should move the mouse over the link, then press the RIGHT mouse button (instead of the left like you normally do). You should see a *Context Menu* spring up, immediately next to the link. You should select the option **“Save Target As”** (in some browsers this is listed as **“Save Link As”**), and then save the file someplace where you can work on it. If you simply click on it like normal, the web browser may open the file and show it you, but when you tell it to save, it won't save the file in the correct format.

 For now I recommend creating a folder on the desktop use your name as the folder’s name, and then save the file into that folder.

## Step : Running JGrasp

 From the *Start* menu in Windows, select *All Programs > jGrasp > jGrasp*. This will start jGrasp, which is the program that we will use to edit the textual description of our program, which is called ***source code***. Much in the same way that Microsoft Word will let you edit your word processing documents, we'll use jGrasp to edit our Java programs' source code. Once jGrasp has finished starting, you should see a window that looks like the one in Figure 1.

 JGrasp’s main window is divided into three sections: the editor panel, the workspace panel and the output panel (see Figure 1). The editor panel is where you will spend most of your time editing code and the output panel will display system messages. The workspace panel is not that useful to us, and you may close it if you want, by clicking the arrows on the divider bar (circled in red in the diagram).

 

Workspace

Panel

Output Panel

Editor Panel

Figure

Next, you should open the **Starter\_File.java** file. You'll do this in jGrasp just like you would in Word: click on the *File* menu, then click on the *Open* menu option. You'll see a dialog box on your screen asking you to find the file. The box might look a little odd, but it's functionally the same as the one that you're used to seeing. Once you've opened the file, your jGrasp window will look like the one in figure 2, below.



Figure 2

While you're not required to do so, your instructor would recommend clicking the 'Maximize' button (circled in red, in Figure 2) so that the **Starter\_File.java** file occupies the entire window. Since we'll typically be working with only one file at a time, this will allow you to see more of the file, without any other drawbacks. After that, your jGrasp window will look like the one in figure 3, below:



Figure 3

## Step : Configure JGrasp for use (do this only once per computer)

 In addition to the functionality that standard Java gives us, we're going to use some custom software that will simulate robots. This software is found in the named **becker.jar**, which you can download from the same website as the **Starter\_File.java**.

 However, before you can write Java programs that use this, you need to tell JGrasp where to find the file. You do this by *seting the classpath* to make JGrasp locate becker.jar NONE OF YOUR PROGRAMS WILL WORK UNTIL YOU DO THIS.

However, *you will only need to do this* ***ONCE*** – just the first time that you start JGrasp on a given computer.

To set the classpath, open the *Settings* menu and select the *PATH / CLASSPATH*option, and then finally the *Workspace* menu item (see Figure 4.) You will see a dialog window that resembles Figure 5. First click on the **CLASSPATHS** tab (not merely the PATHS tab), and then click the **New** button, and you'll see another dialog window that resembles Figure 6. Click on the button labeled **Browse**, and find the Becker.jar file that you've downloaded. Becker.jar contains all of the custom software that we'll need to simulate our robots. You may also add the documentation here, if you'd like, but you're not required to. When you click OK, you should an entry in the window, which is circled in blue in the Figure 5, below. Click OK button (circled in pink) to get back to the main editor window.

|  |  |
| --- | --- |
| Figure 4 | Figure 5 |
| Figure 6 |

## Step : Compiling and running the program

You are now ready to compile and run the Starter\_File.java program. The key buttons in JGrasp are “Compile File”  and “Run Application For Current File”  (this icon looks like a running person.) The "Green Plus Sign" button compiles programs. Compiling a program converts it from a textual **.java** file that we can understand into a binary **.class** file that the computer can understand. If you don't do this, then you won't be able to run your program, since the computer can't understand textual Java directly. The running person button executes (or runs) the binary version of your Java programs. All Java programs must be compiled before you run them, and you must re-compile the program *every single time you change the program!*

Look at the contents of the output panel after you compile the program. If all goes well, the last line should say “operation complete” (see Figure 7) If something goes wrong, the compiler will generate errors and print them in this output panel. Since there weren't any error messages, we know that the program was successfully compiled.



Figure 7

After compiling, then run the program, using either the runner icon button or the *Build*menu's *Run* menu option. After a second, the Robot window will open; the red arrow is the robot and the yellow circle is a Thing, as you can see on the next page.

 

Move The Thing Here

Thing

Jo

Figure 8

## Step : Making The Program Do Something

Jo can do a variety of things. To start, we'll restrict ourselves to four simple actions: move (this moves the robot forwards one intersection), turnLeft (this turns the robot left 90°), pickThing (if there's a Thing in the same intersection as the robot, the robot will pick the Thing up), and putThing (if the robot is holding at least one Thing in it's backpack, it will put down the Thing that it most recently picked up).

If you look at the Starter\_File.java file, on line 23 you'll see:

 Robot **Jo** = new Robot(toronto, 0, 3, Direction.EAST, 0);

We would examine this line in more detail in Cascadia’s BIT 115 class (“Introduction To Programming”), but for now, the important thing to learn from this line is that a new Robot is created, and *that Robot's name is* ***Jo****.* If we want to give a command to a robot, we first have to indicate *which* Robot (since we could create more than one, although we don't do that here). Then, we have to indicate which command we want to give to the Robot. The Java language requires that we separate the name of the Robot and the name of the command using a single period. Further, every command must end in a pair of parentheses: one open paren, and one close paren. Lastly, almost every line of Java must end in a semicolon. Thus, if we wanted to tell Jo to move, we'd add the Jo.move(); command to the existing program, as shown here:

(If you want jGrasp to show you the line numbers in the Editor Panel, go to *View🡪Line Numbers*, or push Control+L as the shortcut)

20 public static void main(String[] args)
21 {
22 City toronto = new City();
23 Robot Jo = new Robot(toronto, 3, 0, Direction.EAST, 0);
24 new Thing(toronto, 3, 2);
25
... // there’s a bunch of lines I left out that all start with “new Wall”
53 **Jo.move();**

Code Snippet 1

However, there's a number of subtle details here that aren't obvious. Java, the language, is incredibly detail-oriented. Get the smallest detail wrong, and the whole program will refuse to run. Notice that we named the Robot **Jo**. If we used the name **JO**, Java will think the name is different because the capitalization is different. Likewise, Java thinks that **jo** and **jO** are also different names. And of course, a misspelling, like **Joe** is even more wrong. So we have to type Jo exactly the same as we did on line 23. Likewise with the move command – we have to type **move()** exactly that way. Also, we need to put these commands in the exact right spot.

|  |
| --- |
| Now comes the fun part – modifying the program so it does something new.Start by reading through the provided Starter\_File.java file, and see if you can figure out what each command does.Next, try to get the robot to move the Thing over to the intersection at row 3, column 5 (as pictured in [Figure 8](#Figure_8), above). Make sure that the robot drops the Thing at the intersection and then moves over (so that you can clearly see the Thing).At this point I recommend modifying your program by copying an existing line and then pasting it into the program further down (but above the line which reads “put your commands ABOVE this line”)Once you've got the above code pasted (or typed) into your Starter\_File.java, you should compile it (using the Green Plus button). If you missed something, you'll get a message in the Output Panel. If you get an error message, compare your program to the original starter program file, and try and figure out what the error is. Finding these errors will take a little while to get used to fixing, so grin and bear it for the time being. Once you've fixed the errors, run the program. At that point, a new window should appear; begin the action by pressing the Start button |