Programming Your IntelliBrain-Bot

In the previous section you became familiar with the hardware features of your IntelliBrai-Bot educational robot. In this section you will begin to learn about the software features of your robot, as well as robotics programming concepts. You will use the RoboJDE Java-enabled robotics software development environment to create, build, load and run your first program. You will also learn debugging techniques which will help you quickly resolve problems with your program.

Note: The RoboJDE development environment should be installed on the computer you will be using prior to proceeding with the hands-on activities in this chapter. Your lab instructor has most likely already taken care of this; however, if you are working on your own, follow the instructions in the RoboJDE User Guide to install the RoboJDE software (http://www.ridgesoft.com/robojde/2.0/docs/RoboJDEGuide.pdf).

Creating a New Project

To begin a new project you must create a new RoboJDE project file to store your project's properties. The RoboJDE development environment uses project files to make it easy for you to switch between different projects.

Use the following procedure to create a project:

1. Start the RoboJDE development environment from the Windows start menu. The default location on the start menu is Start->All Programs->RoboJDE->RoboJDE.

   The RoboJDE Graphical User Interface (GUI) will appear.

2. Select File->New Project menu item in the RoboJDE GUI.

   The Project Properties dialog will appear.

3. Click the browse button to the right of the "Project folder" field.

   The Choose File dialog will appear.

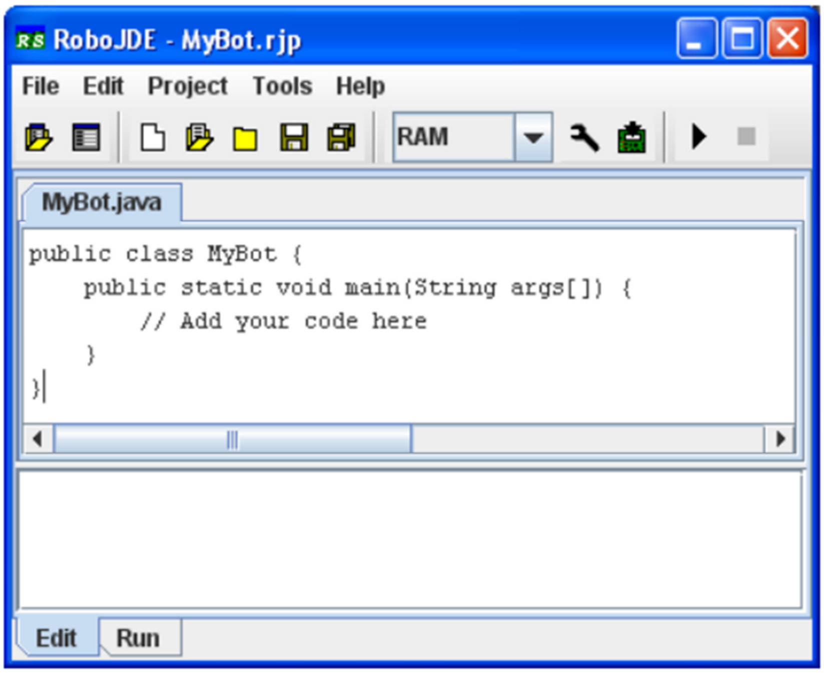
4. Browse to and select the folder in which you want to create your project.

   You can create a new folder by browsing to the location where you want to create a new folder and then clicking on the create folder button. A folder titled "New Folder" will appear. Click on the new folder's name and change it to a name you choose. Then click on the folder icon to the left of the name to select it. Click OK.

5. Enter the name "MyBot" in the "Main class" field.

6. Click OK.

   Your MyBot project and Java class file will be created, as shown in Figure 2-1.



```
public class MyBot {
    public static void main(String args[]) {
        // Add your code here
    }
}
```

Fig. 2-1 – New MyBot Project Viewed in the RoboJDE GUI.

7. Using the mouse, select the text: "// Add your code here" and replace it with "System.out.println("Your Name");" inserting your name, so your program looks similar to the following:

```
public class MyBot {
   public static void main(String args[]) {
      System.out.println("Mr. Roboto");
   }
}
```

   Java is very particular about details such as upper and lower case letters and punctuation. Paying careful attention to these details will save you a lot of time and frustration debugging subtle errors in your programs!

8. Click the "Save all" button (see Figure 2-2) or select the File->Save All menu item.

   The Save dialog will appear with "MyBot" as the proposed file name.

9. Click the "Save" button to save your project.

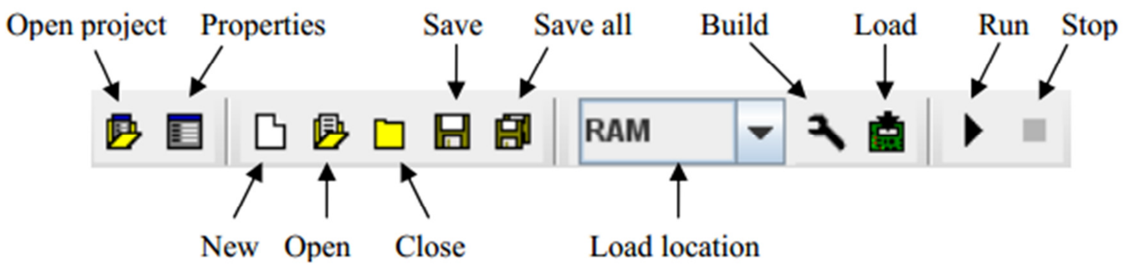


Fig. 2-2 – RoboJDE Tool Bar.

Your first program is now ready to try out. First you must connect your robot to your computer. This will enable you to download your program into the IntelliBrain 2 robotics controller's memory.

Connecting Your Robot to Your Computer

The RoboJDE development environment communicates with your IntelliBrain-Bot educational robot via a serial port on your computer. All you need to do to establish a connection is connect a cable between the port labeled "COM1" on the IntelliBrain 2 robotics controller and the serial port on your computer.

Note: Your lab instructor should have already attached the cable to the serial port on your computer and configured appropriate settings in the RoboJDE development environment. However, if this is not the case, please consult the IntelliBrain 2 User Guide and RoboJDE User Guide for further instructions.

Use the following procedure to connect your robot to your computer:

1. Locate the load button on the RoboJDE tool bar (see Figure 2-2) and note its state.

   Then the RoboJDE GUI is unable to communicate with your robot the load button will appear gray. Since your robot is not currently connected to your computer the button is gray.

2. Gently attach the free end of the serial cable to the port labeled "COM1" on the IntelliBrain 2 robotics controller.

   The cable will slide on to the connector more easily if you gently rock the cable left and right as you press it on to the connector. Although the connector on your robot is quite sturdy, be careful not to apply excessive force because you may
   damage your robot.

3. Switch the power on.

   If the cable is connected properly and the RoboJDE development environment communication settings are correct, the Load button will switch from gray to green.

If the Load button did not turn green, request help from your lab instructor to ensure the settings (Tools->Settings) in RoboJDE development environment and the baud rate setting in the IntelliBrain 2 robotics controller are correct. In most cases the baud rate should be set to 115.2K both in the RoboJDE GUI and on the IntelliBrain 2 robotics controller. Also check to be sure the Serial Port setting in the RoboJDE GUI is the port on your computer you are using. Finally, be sure the “Board type” setting in the RoboJDE GUI is set to “IntelliBrain.”

4. Switch the power off

Running Your First Program

Everything is now set to give your program a try. You will need to build it, download it and run it. Fortunately, this is much easier than it sounds – only two mouse clicks!

Do the following to give your program a try:

1. Switch the power on.

2. Click the Load button in the RoboJDE GUI.

   This will compile, link and load your program. You will see messages from the compiler and linker in the output window at the bottom of the RoboJDE GUI window. If you typed everything correctly, there will be no errors and the load progress window will display briefly. If you made a mistake, you will see error messages in the output window.

   Once the load progress window disappears, the LCD screen on your robot will display the following message on the second line:

   Ready – RAM

   This indicates there is now a program loaded into Random Access Memory (RAM) that is ready to run.

3. Click the Run button on the RoboJDE tool bar to run your program.

   Your program will run very quickly. If you watch the LCD screen you will see your name appear momentarily. Click the Run button again if you missed it.

   Also, notice the RoboJDE GUI switched to its Run window, where your name was also output by your program. By default, any output to “System.out” goes to both the LCD screen and the RoboJDE Run window, if the serial cable is connected.

4. Press the START button.

This also runs your program, but without clearing the output in the RoboJDE Run window. Each time you press the START button another line displaying your name will appear.

5. Switch power off.

   Congratulations! You have now created and run your first robotics program!

Programming Concepts

If this is the first program you've ever created, or you are new to Java, you are probably a little vague on many of the concepts we've covered so far. If you don't fully understand your program, rest assured, as you work through the hands-on lessons in this book your understanding will become clearer.

What is a Program?

A program is a series of instructions a computer executes in steps, one after the other. The computer executes one step and then proceeds on to the next step, executing each step before proceeding to the next step, and so on, until computer reaches the end of your program.

You can think of a program similar to how you think of a recipe. A computer executes a program similar to how you execute the steps of a recipe – starting at the beginning, executing steps, one after another, until you have completed the recipe.

The Method Named "main"

In the case of your IntelliBrain-Bot educational robot, the IntelliBrain 2 robotics controller is a small computer. It executes the steps of your program. It begins executing your program in the method named "main."

Look for the word "main" on the second line of your program. This is the start of the main method. Your program begins executing on the line after this; the line that contains your name. Your program is very simple; it has only one step, which prints your name. Once the robotics controller executes this step, it reaches the end of the main method and exits your program. This explains why your name was only displayed on the LCD screen for a split second.

You can change this behavior by adding one more step to your program so it will wait for the STOP button to be pressed. This will cause your program to display your name and then wait for you to press the STOP button before it exits.

Use the following procedure to extend your program:

Add the following import statement as the first line of your program

```
import com.ridgesoft.intellibrain.IntelliBrain;
```

This tells the compiler your program will be using the IntelliBrain class from the class library.

1. Add a statement to wait for the STOP button to be pressed after your name has been printed, so your program looks similar to the following:

```
import com.ridgesoft.intellibrain.IntelliBrain;
public class MyBot {

   public static void main(String args[]) {
      System.out.println("Mr. Roboto");
      IntelliBrain.getStopButton().waitPressed();
   }
}
```

2. Switch power on.

3. Click the load button.

4. Click the run button in the RoboJDE GUI, or press the START button on your robot.

   Observe that your name does not disappear from the LCD screen. This is because your program is waiting for the STOP button to be pressed.

5. Press the STOP button.
   Observe your program has stopped running and your name is no longer displayed on the LCD screen.

6. Switch power off.

   Your program now includes two steps, one which tells the computer to display your name, and another which tells the computer to wait for you to press the STOP button.

The Programming Process

As you develop programs for your robot you will become very familiar with the programming process, which is illustrated in Figure 2-3. You will use this process over and over to program your IntelliBrain-Bot educational robot. Each time you create a new program, or make a change to an existing program, you will complete the following steps:

1. **Edit** – add, modify or delete steps in your program (use the RoboJDE edit window).
2. **Build** – compile, link and download your program to your robot (click the load button).
3. **Test** – test your program (click the run button on the RoboJDE tool bar or press the START button on your robot).

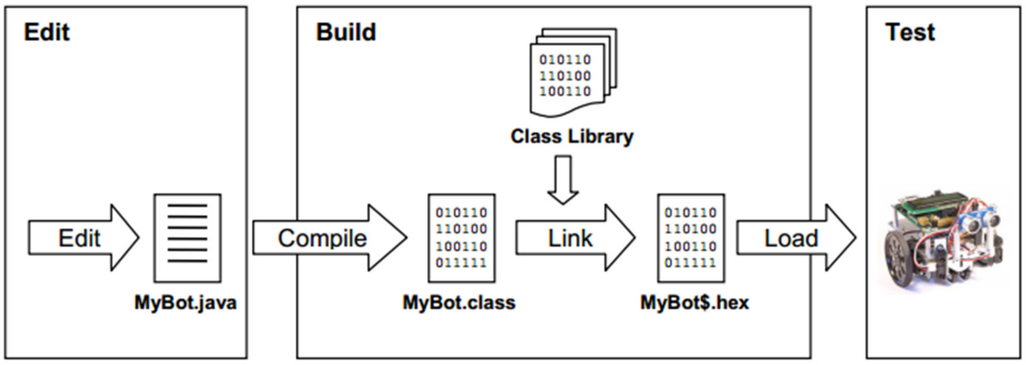


Fig. 2-3 – The IntelliBrain-Bot Educational Robot Programming Process.

You have now completed this process twice, once when you created your first program, and again when you added a step to it. You will repeat it many more times as you develop larger and more sophisticated programs.

Most seasoned programmers develop programs iteratively, making only one small change at a time, testing it, then moving on to the next small change, and so on, until his or her project is completed. Making and testing changes in small increments has an advantage over making fewer large changes; it is much easier to thoroughly test your changes, as well as find and fix problems, when you haven't made large changes. By keeping changes small, you focus your attention one very small area. If your program doesn't work after you have changed it, it will be easier to resolve the problem if you haven't made a large change. We recommend you follow this approach whenever possible, making and testing small changes, rather than attempting large changes.

Behind the Scenes

Let's take a close look at Figure 2-3 to examine what goes on behind the scenes. If you browse to the folder where you created your program you will see the following files:

1. **MyBot.java** – your Java source file
2. **MyBot.class** – the Java class file generated by the compiler
3. **MyBot$.hex** – the executable file generated by the linker
4. **MyBot.rjp** – a file containing your project's properties

The text you entered into the RoboJDE edit window to define your MyBot Java class was saved in the file named "MyBot.java." This is the Java source file for your main Java class, MyBot.

When you clicked the load button, three things took place. Your MyBot program was compiled, linked and loaded, as depicted in the block labeled "Build" in Figure 2-3. In the first step, your MyBot class was compiled from its source file, MyBot.java. This generated the Java class file, MyBot.class. In the second step, the MyBot.class file was linked with other classes MyBot references, which are included in the class library. The class named "System," is a class from the class library that is referenced by your program. The linked program was stored in the executable file, MyBot$.hex. Finally, the executable file was transferred via the serial cable and loaded into the memory of the IntelliBrain 2 robotics controller, which allowed it to be run and tested.

The compiler parses and analyzes the text in a Java source file, producing binary "byte codes," which the RoboJDE virtual machine can execute. The virtual machine resides on the IntelliBrain 2 robotics controller, enabling it to execute Java byte codes.

The linker assembles all of the classes that are needed to execute your program into a single file. While doing this, the linker also creates the necessary linkages between the classes. These linkages provide the virtual machine with the information it needs to understand how the classes interact to form your program. You only had to create one very small Java class to create your program, but your program cannot execute without including many more classes from the class library. You can see how many classes are included in the executable by reviewing the output from the linker at the bottom of the RoboJDE Edit window. Surprisingly, your simple program references approximately fifty other classes from the class library! The classes in the library provide a rich foundation on which you create your programs. This allows you to focus your effort on the algorithms that control your robot, rather than getting bogged down in low level details.

Debugging Your Programs

If you are like most programmers, your programs will rarely work on the first try. Typically, once you finish making edits, you will find you are faced with one or more compilation errors. You will need to make further edits to your program to correct your errors. Once your program compiles and links successfully, you will be able to download it to your robot and run it, but you will frequently find it doesn't do what you expect. When this happens, you will need to analyze what your program is doing and determine what changes are necessary to make it do what you intend.

The ability to debug problems is an essential programming skill. You will be able to complete your programming projects faster, and your programs will often work better if you take the time to develop and apply your debugging skills. In contrast, if you try to program without learning how to debug effectively, you will likely find programming a frustrating experience. Take the time to analyze and understand the problems you encounter. It will make you a better programmer and you will find programming more enjoyable!

Compilation Errors

The compiler converts the text you enter, which is called "source code," into Java byte codes, which the RoboJDE virtual machine can execute. Conceptually, this is similar to translating a document from a foreign language to your native language. In order to translate such a document you would apply your knowledge of the vocabulary and grammar of the foreign language to understand the source document. Once you understood it, you could then express it's meaning using the vocabulary and grammar of your native language.

Imagine if you were given the task of translating a document that was full of spelling errors, slang, poor grammar, punctuation errors and ambiguities; this would make your translation job much more difficult. It's likely you wouldn't have a lot of confidence that you could accurately communicate the thoughts and emotions of the author in your translated version.

Similarly, the Java compiler isn't able to reliably convert your Java source code to executable code if it contains misspellings, incorrect grammar, words the compiler doesn't know, poor punctuation and ambiguities. Rather than trying to guess what you intended the Java compiler outputs an error message for each problem it encounters. Each error message indicates where in your source code the compiler encountered a problem followed by a message describing the problem.

Use the following procedure to experiment and see what the compiler does when you introduce errors into your program:

1. Edit the fifth line of your program to replace the period between “out” and “println” with a comma as follows:

   ```
   System.out,println("Mr. Roboto");
   ```

2. Click the build button (wrench icon) on the RoboJDE tool bar.

   Observe the compilation error reported in the output pane at the bottom of the edit window. You will see an error message similar to the following:

   ```
    Found 1 syntax error in "MyBot.java":
         5.                 System.out,println("Mr. Roboto");

                                      ^
   *** Syntax: . expected instead of this token
   ```

   The first line of the message indicates there is a syntax error in MyBot.java. The second line shows the problem line from your program, with the number of the line displayed on the left. The third line indicates the location of the error in the problem line by using a carat (^) character. The fourth line tells you what the problem is. In this case, the compiler expected a period instead of the “token” pointed to by the carat, which is a comma.

3. Select the menu item Edit->Go to Line in the RoboJDE GUI or enter Ctrl-G using the keyboard.

   The Go to Line dialog will appear.

4. Enter the number of the line with the error: 5, and click OK.

   The RoboJDE editor will scroll to the line and highlight it. Since your program is very short, this may not seem necessary. When your programs get larger you will find this feature very useful. For example, if you had an error on line 327 of a 500 line program, you would find it convenient to jump right to the line rather than having to scroll down looking for it.

5. Correct the error and click the build button.

There are too many possible compilation errors to discuss them all here. The key to debugging them is to carefully read the messages from the compiler and understand what they are telling you. Always scroll up to the first error message and try to fix it first.

Subsequent error messages are often due to the first problem. When you fix the first problem, it is often best to re-compile immediately because the fix may eliminate subsequent errors. Re-compiling is quick and easy, so don’t hesitate to do it often. Just click the build button or load button on the RoboJDE tool bar.

Exceptions and Stack Traces

In addition to encountering errors when you compile your programs, the virtual machine, is able to catch many errors that can only be detected while your program is running. For example, if your program attempts to use more memory than is available, the virtual machine will detect the problem and “throw” an exception. There are many other types of exceptions, such as attempting to divide by zero, or attempting to use a reference to an object when the reference is “null” (not referring to any object).

Without going into all of the details of exceptions, let’s take a quick look at what you will see when an exception gets “thrown.” Use the following procedure to make a small change in your program that will introduce a bug:

Insert the following line into your program as the second line:

```
private static String myName;
```

This line creates a field to keep track of your name.

1. Modify the print statement in your program, replacing the quoted string containing your name with “myName,” the new field you just declared.

```
import com.ridgesoft.intellibrain.IntelliBrain;
public class MyBot {
   private static String myName;
   public static void main(String args[]) {
      System.out.println(myName);
      IntelliBrain.getStopButton().waitPressed();
   }
}
```

2. Switch the power on.

3. Click the load button to build and download your program.

Click the run button.

You will see the following in the run window:

```
NullPointerException
     at java.io.PrintStream.print(PrintStream.java:44)
     at java.io.PrintStream.println(PrintStream.java:96)
     at MyBot.main(MyBot.java:5)
     at
       com.ridgesoft.intellibrain.StartupThread.run(StartupTh
       read.java:31)
```

This is the type of output you will see when your program causes an exception to be thrown. In this case, the exception is a “NullPointerException.” The lines that follow indicate exactly which statements in your program resulted in the exception being thrown. This is a stack trace. It shows that your program was executing the PrintStream class’s print method at line 44 of a file named PrintStream.java when an attempt to use a null reference (null pointer) occurred. This class happens to be in the RoboJDE class library and is most likely not the source of problem, it's just where the problem showed up. The next line of the stack trace shows the println method called the print method. This is also in the PrintStream class. The third line of the stack trace indicates line 5 of your MyBot class called the println method.

4. Click the Edit tab in the lower left corner of the RoboJDE GUI.

5. Type Ctrl-G at the keyboard.

6. Enter 5, the line indicated in the stack trace, in the Go to Line dialog and then click OK.

This will show you the line in your program that was executing when the NullPointerException occurred. Examining this line you will see it does indeed call the println method, as the stack trace indicated. This is a line you just modified. The NullPointerException must be due to this change.

When you added the new field to your program, we neglected to tell you to initialize the field with your name; therefore, the field is null. This is a bug.

7. Correct the bug by initializing the field with a text string containing your name.

```
private static String myName = "Mr. Roboto";
```

8. Click the load button.

9. Click the run button.

   Observe your program once again works correctly.

10. Switch the power off.

Debugging Using Print Statements

Frequently your programs will compile and run just fine but still not work the way you expect. Often, the best way to solve these types of problems is to add print statements to your program. This will enable you to better understand what your program is doing. Being able to peer into your robot's brain is such a valuable debugging and test tool that it is a good idea to start by implementing these features first, knowing they will come in handy as you develop the main features of your program. For example, when working with a new sensor, a great place to start is by displaying the sensor's reading on the LCD screen. This allows you to check that the sensor is connected correctly and functioning properly.

Use the following procedure to add a print statement to your program indicating when the STOP button has been pressed:

1. Add the following print statement to your program immediately after the statement to wait for the button to be pressed.

```
System.out.println("STOP pressed!");
```

2. Click the load button to build and download your program.

3. Click the run button.

   Observe your program has printed your name in the RoboJDE Run window.

4. Press the STOP button.

   Notice, unfortunately, the new message you just added did not display. Your program is not working quite the way you may have expected.

5. Add another print statement just before the statement to wait for the button to be pressed.

   ```
   System.out.println("Waiting for STOP");
   ```

6. Build and run your program again.

   Observe your new message does show up in the RoboJDE run window and on the LCD screen, but the message indicating the STOP button has been pressed still does not display.

   It turns out your program has a minor bug in it. The virtual machine, which executes your program, assumes by default it is responsible for monitoring the STOP button. When the button is pressed the virtual machine immediately stops your program rather than letting it continue. In this case, the new print statement never executed because the virtual machine stopped your program before it got to the print statement. This problem is easy to fix.

7. Insert the following statement just before the statement that prints “Waiting for STOP:”

   ```
   IntelliBrain.setTerminateOnStop(false);
   ```

   This tells the virtual machine it should not terminate your program when the STOP button is pressed.

   Your program should now be similar to the following:

   ```
   import com.ridgesoft.intellibrain.IntelliBrain;
   public class MyBot {
      private static String myName = "Mr. Roboto";
      public static void main(String args[]) {
         System.out.println(myName);
         IntelliBrain.setTerminateOnStop(false);
         System.out.println("Waiting for STOP");
         IntelliBrain.getStopButton().waitPressed();
         System.out.println("STOP pressed!");
      }
   }
   ```

8. Load and run your program.

   Observe, your program now prints all of the messages that you expect. You have debugged the problem!

Other Methods of Debugging

There are many other ways to debug your programs. In later chapters you will learn about programming the IntelliBrain 2 robotics controller's Light Emitting Diodes (LEDs) and its buzzer. These provide additional means of indicating what your program is doing.

Resources

To finish this section about how to program you IntelliBrain-Bot, take a look at the following documents that you can find in web page

http://www.ridgesoft.com/robojde/robojde.htm

- RoboJDE Datasheet (gives an overview of the features of the The RoboJDE Java-enabled robotics software development environment).
- RoboJDE User Guide (is the manual of the software development environment).
- Overview The IntelliBrain 2 API (gives an overview of the class library to program the controller).
- Full API (the API Javadoc)
- Java benefits and (gives an overview of the Benefits of Java-enabled Robotics).
- Java Primer (r provides a very brief overview of the Java programming language).

Consider these documents important reference material to use during the programming of the robots throughout the course.