



Chapter #1: Assembling and Testing Your J-Bot

About Robotics Competitions and Robot Development

Students in high schools and colleges preparing their entries for various robotics competitions get first-hand exposure to the engineering occupation. They start by working in teams developing a Robot's subsystems. A robot's subsystems include its motors, sensor arrays, microprocessor, and mechanical linkages. Next they test and trouble-shoot the subsystems. Then comes system integration, the process of making all the Robot's subsystems work together.

Once the testing and trouble-shooting is finished at the subsystem level, a robot's subsystems have to be connected to and controlled by a microprocessor. The process of getting all the subsystems (including the microprocessor) to work together to make the robot perform its assigned task list is called system integration. System integration can be tricky to begin with, but robotics teams who skipped any of the testing and troubleshooting at the subsystem level often have much larger problems with their system integration. Many a late night can be spent trying to get the robot to work the way it's supposed to. If bugs are hiding in the subsystems when you're trying to do system integration, it only compounds the problems.

Even when testing and troubleshooting is performed for each subsystem, it can still be the most difficult part of robot development. For example, a group at a recent robotics competition spent five hours trying to get a Sumo wrestling robot to work right with no luck. Later, by utilizing the Javelin's Debug Terminal, the testing and troubleshooting took less than 5 minutes.

Testing and troubleshooting at each phase of robot development is a skill that one gets better at with practice. By following the instructions in the activities in this student workbook, you'll get a taste of testing and troubleshooting while putting your J-Bot together and getting it up and running. With practice, you'll enjoy more five-minute troubleshooting times and less of the five-hour variety.

This chapter is separated into four activities:

1. J-Bot Parts and Tools
2. J-Bot Mechanical Assembly
3. Testing PC - J-Bot Communication
4. Building and testing a Speaker Alarm - Intro to Virtual Peripherals

Each of these activities involves discrete steps to get the J-Bot up and running. First, check to make sure you have all your parts. Next, put the mechanical parts together. After that, test the microprocessor subsystem. Then test each servo motor individually. Then, make the servo motors work in unison. Last, but certainly not least, calibrate the pre-modified servos. By carefully following the instructions in these first five activities, you ensure that your microprocessor and motor subsystems are working reliably. The task in later chapters will be to develop and test a variety of sensors and integrate them with the rest of the J-Bot's subsystems. In Chapters 5-7, you'll isolate and test the sensors before writing JAVA programs that integrate the sensor subsystems. For example, in chapter 5, you'll first construct and test whiskers, sensors that tell the J-Bot when it's bumped into something. Once the testing and troubleshooting is complete, you'll move on to writing JAVA programs that make use of the whisker input signals for directing the J-Bot's motion.

Activity #1: J-Bot Parts and Tools

Let's get started by taking an inventory of the tools and parts we'll need to get through the activities in this student workbook. For starters, all activities in this student workbook require a personal computer (PC) with the Windows 95/98/... operating system. You'll also need a few simple hand tools, all of which are common and can be found in most households, and school shops. They can also be purchased at local hardware stores. The parts for the J-Bot are either included in the J-Bot full kit or in a combination of the JSDB Full Kit and the J-Bot parts kit. See Appendix A: J-Bot Parts Lists and Sources for more information.

The Simple Hand Tools

Recommended Tools

The row of tools in
 1 recommended for
 A es in Chapter #

Phillips #1 point
 screwdriver
 Combination wrench

The tools shown on the bottom row
 will come in handy for the
 activities from Chapter #2
 onward.

(1) Small needle nose
 pliers
 (1) Wire cutter/stripper



Figure 1.1: Recommended tools.

J-Bot Parts Inventory

- ❑ Before getting started, take an inventory of the parts in your kit. Appendix A: J-Bot Parts Lists and Sources will tell you how many of each part should be in your kit. For help with identifying each part, use the back cover of this text; it has labeled pictures of all of the J-Bot parts.
- ❑ Gather the parts shown in Figure 1.2 and set them aside for use as you go through the rest of the activities in this chapter.

Chapter #1 Parts List:

A	(1)	J-Bot chassis
B	(1)	Battery pack
C	(2)	Parallax Pre-Modified Servos (labeled PM)
D	(2)	Plastic wheels
E	(1)	Polyethylene ball
F	(2)	9/32" Rubber Grommets
G	(1)	13/32" Rubber Grommet
H	(1)	Board of Education and Javelin
I	(2)	O-ring tires
J	(1)	Cotter pin
K	(10)	4-40 locknuts
L	(2)	4-40 flathead screws
M	(8)	3/8" 4-40 screws
N	(8)	1/4" 4-40 screws
O	(4)	1/2" Standoffs
P	(1)	Serial cable
Q	(4)	AA alkaline batteries
R	(1)	Parallax CD

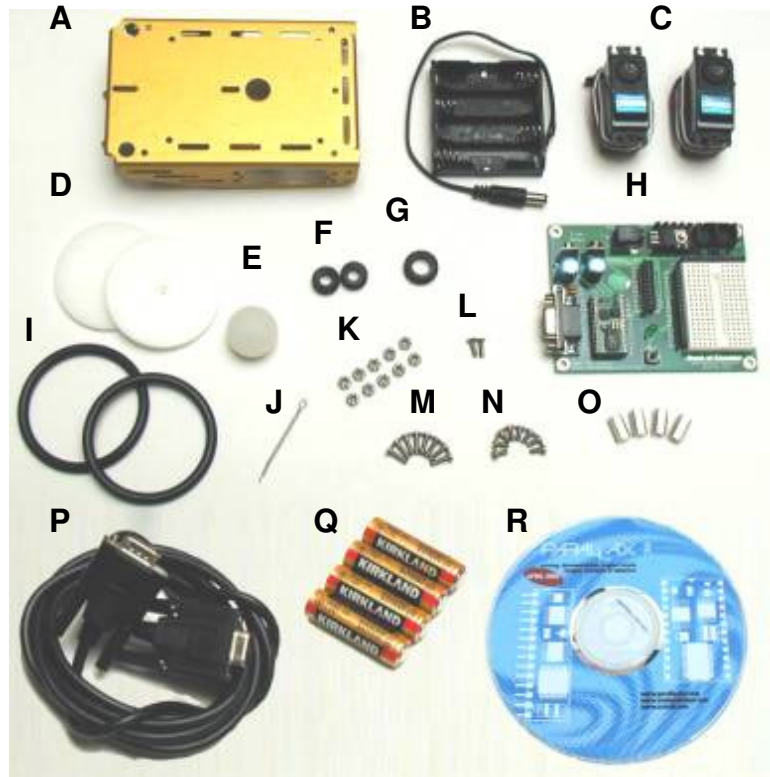


Figure 1.2: Chapter #1 parts.

Activity #2: J-Bot Mechanical Assembly

This section breaks assembling the J-Bot into steps. In each step, you gather a few of the parts, and then assemble them so that they match the pictures. Each picture has instructions that go with it; make sure to follow them carefully.

1

Mounting the Topside Hardware

Figure 1.3 shows the J-Bot chassis, topside hardware and mounting screws.

Parts List:

- (1) J-Bot Chassis
- (4) Standoffs
- (4) 1/4" 4-40 Screws
- (2) 9/32" Rubber grommets
- (1) 13/32" Rubber grommet

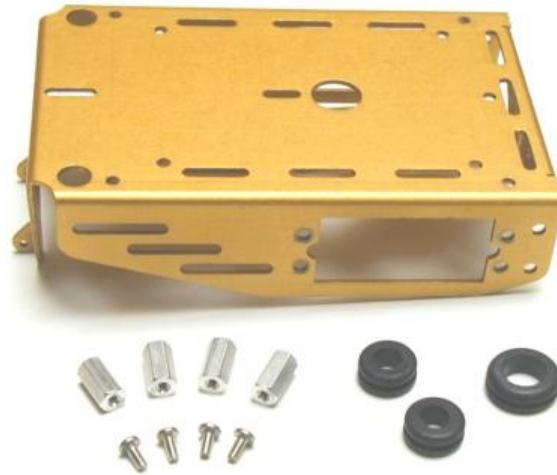


Figure 1.3: Chassis and topside hardware.

Assembly:

Figure 1.4 shows the topside hardware attached to the J-Bot chassis. Each rubber grommet has a groove in its outer edge that holds it in place in a hole on the top of the J-Bot chassis.

- ❑ Insert the 13/32" rubber grommet into the hole in the center of the J-Bot chassis.
- ❑ Insert the two 9/32" rubber grommets into the two corner holes as shown.
- ❑ Use the four 1/4" 4-40 screws to attach the four standoffs to the chassis as shown.



Figure 1.4: Topside hardware assembled.

Removing the Servo Horns

Get the two Parallax pre-modified servos from your parts kit, shown in Figure 1.5. Each servo has a horn attached to its output shaft by a Phillips screw.

Parts List

- (2) Pre-modified servos

Figure 1.6 shows the dehorned servos.

- ❑ Unscrew each of the Phillips screws, then pull each servo horn upwards and off of the servo output shaft.
- ❑ Save the screws for attaching the J-Bot wheels.

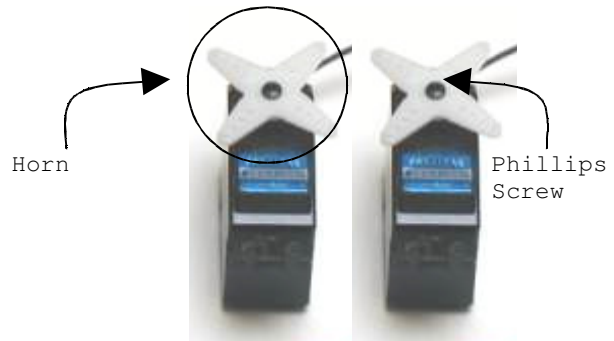


Figure 1.5: Parallax pre-modified servos.



Figure 1.6: Pre-modified servos dehorned.

Mounting The Servos

Parts List:

Figure 1.7 shows the pre-modified servos and servo mounting hardware.

- (1) Partially assembled J-Bot chassis
- (2) Servos
- (8) 3/8" 4-40 screws
- (8) 4-40 locknuts



Stop

If you have not already checked the labeling on your servos, do that now. Turn to page 1 and follow the instructions.



Figure 1.7: Servos and mounting hardware.

Assembly:

Figure shows the servos mounted on the chassis.

- ❑ Use the eight 3/8" 4-40 screws and locknuts to attach each servo to the J-Bot chassis as shown.

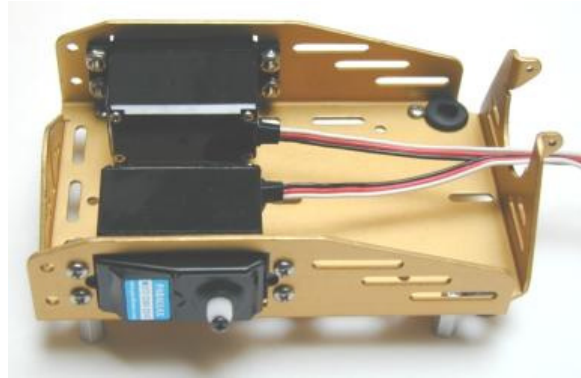


Figure 1.8: Servos mounted on chassis.

Assembly:

Figure 1.9 shows the IR wheel encoder detectors mounted on the chassis. Now would be the perfect time to mount these sensors. The IR wheel encoder detectors are not included in the Boe-Bot kit. You need to purchase the Boe-Bot Digital Encoder Kit (Part #28107) from Parallax, Inc. The IR wheel encoders are needed for exercises later in this book. However, the IR wheel encoders can be easily added later.



Figure 1.9: IR wheel encoder detectors mounted on chassis.

Mounting the Battery Pack

Figure 1.10 shows the battery pack and mounting hardware to be added next.

Parts List:

- (1) Partially assembled J-Bot chassis.
- (1) Empty battery pack
- (2) Flathead 4-40 screws
- (2) 4-40 locknuts



Figure 1.10: Battery pack and mounting hardware.

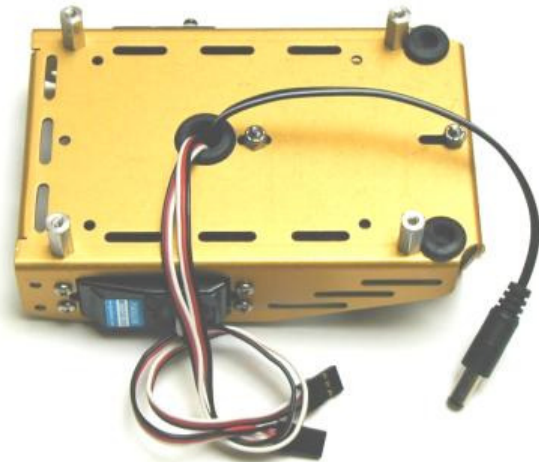
Assembly

Figure 1.11 shows the J-Bot chassis with the battery pack mounted (a) from the underside and (b) from the topside.

- ❑ Use the flathead screws and locknuts to attach the battery pack to underside of the J-Bot chassis as shown in Figure 1.11 (a). Make sure to insert the screws through the battery pack then tighten down the locknuts on the topside of the chassis.
- ❑ Pull the battery pack's power cord through the hole with the largest rubber grommet in the center of the chassis.
- ❑ Pull the servo lines through the same hole.
- ❑ Arrange the servo lines and supply cable as shown in Figure 1.11 (b).



Figure 1.11: (a) Battery pack installed



(b) wires pulled through.

Socketing the Javelin.

Figure 1.12 shows the Javelin to the left of the Javelin Stamp Demo Board

Parts List:

- (1) Javelin
- (1) Javelin Stamp Demo Board

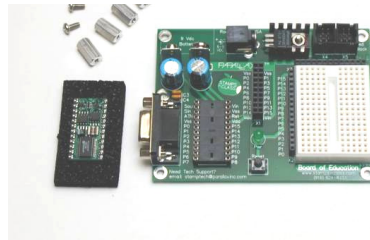


Figure 1.12: Javelin and Board of Education

FYI Board of Education is abbreviated BOE.

Figure 1.13 shows the Javelin mounted in its socket on the JSDB. The Javelin has a half-circle printed in the center of its top edge. This is meant to serve as the reference notch common on many integrated circuits. When placing the Javelin in its socket on the JSDB, make sure this half-circle is closest to the Sout and Vin labels. As a second check, make sure the largest black chip with the label SX48 is at the bottom, between the P7 and P8 labels.

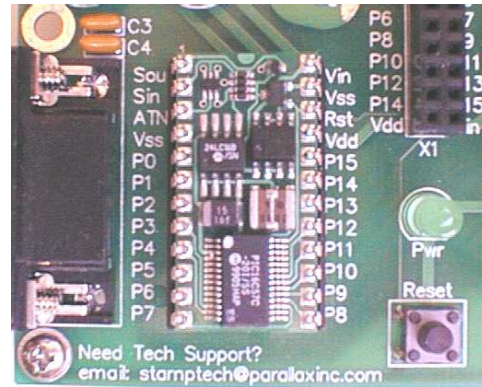


Figure 1.13: Javelin inserted into its socket on the BOE.

- ❑ If your Javelin and JSDB were packaged separately, plug the Javelin into its socket on the JSDB as shown in Figure 1.13. Make sure the pins on the Javelin line up with the holes in the socket, then press down firmly on the Javelin with your thumb. The Javelin's pins should sink into socket holes by about a quarter-inch.

Attaching the Board of Education to the J-Bot Chassis

Figure 1.14 shows the Board of Education, Javelin and mounting hardware.

Parts List:

- (1) Partially assembled J-Bot (not shown)
- (1) Board of Education with Javelin
- (4) 1/4" 4-40 screws

Assembly:

Figure 1.15 shows the Board of education attached to the J-Bot chassis with the servos plugged into the servo ports.

- ❑ Make sure the white breadboard on the Board of

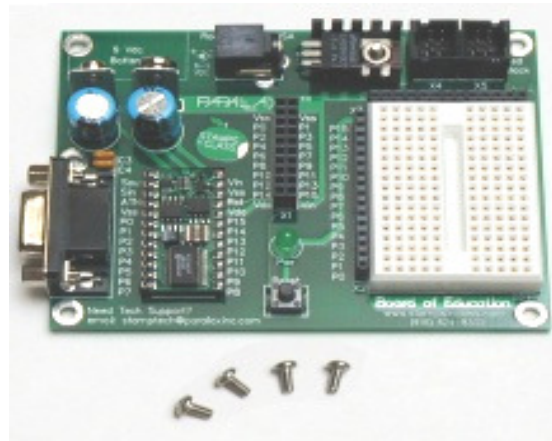


Figure 1.14: BOE with Javelin and mounting screws.

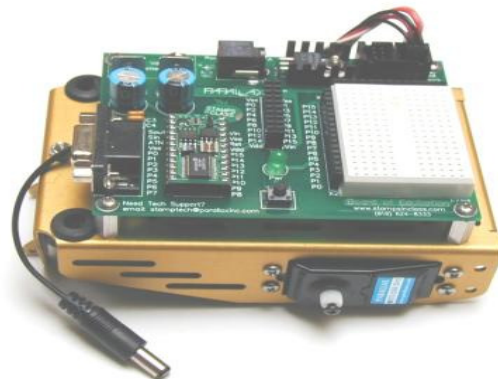


Figure 1.15: BOE attached to chassis.

Education is above where the servos are mounted on the chassis.

- ❑ Use the four 1/4" machine screws to attach the Board of Education to the standoffs.

Figure 1.16 (a) shows a close-up of the servo ports on the BOE. The numbers along the top indicate the servo port number. If you connect a servo to servo port 12, it means the servo's control line is connected to I/O line P12. I/O line P12 is a metal trace on the BOE that connects the top servo port pin to the Javelin's I/O pin P12.

The labels to the right of the servo port are for making sure your servo gets plugged in properly. Figure 1.16 (b) shows a servo plugged into servo port 12 so that the black wire lines up with the "Black" label, and the red wire lines up with the "Red" label. Although the topmost wire is labeled "White" in Figure 1.16 (b), it could either be white or yellow.

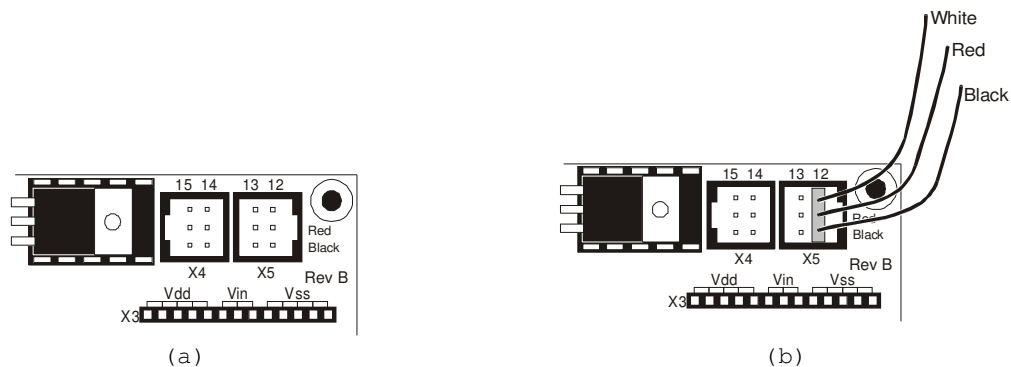


Figure 1.16: Servo ports on the BOE (a) before, and (b) after plugging in servo port 12.



Make sure the "Black" and "Red" labels to the right of the servo port line up with the servo connector's black and red wires before plugging in a servo.

- ❑ Plug the servo that you can see in Figure 1.7 into servo port 12, and plug the other servo into servo port 13.

TIP

The BOE Rev A does not have built-in servo ports. If you can not find the servo ports shown in Figure 1.16, go to Appendix F: Building Servo Ports on the Rev A Board of Education.

The Wheels

Figure 1.17 shows the J-Bot's wheel parts and mounting hardware.

Parts List:

- (1) Partially assembled J-Bot (not shown)
- (1) 1/16" Cotter pin
- (2) O-ring tires
- (1) 1" Polyethylene ball
- (2) Plastic machined wheels
- (2) Screws that attached the servo horns, which were set aside in the Removing the Servo Horns step.

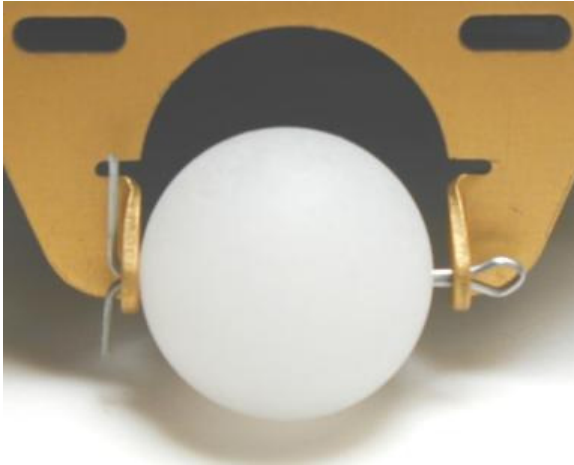


Figure 1.17: Wheel parts.

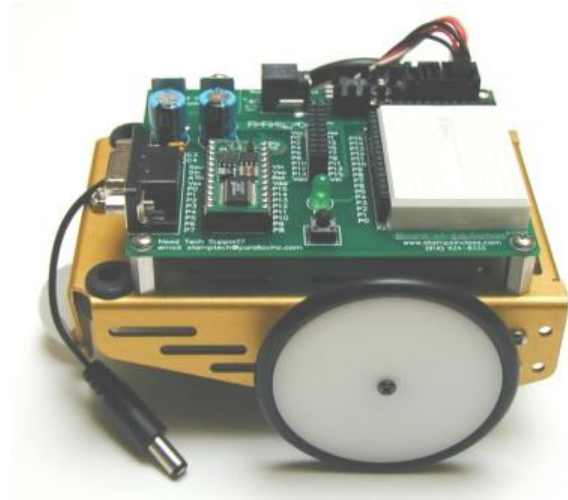
Assembly:

Figure 1.18 (a) shows the tail wheel attached to the J-Bot chassis with a cotter pin, and Figure 1.18 (b) shows one of the front wheels attached to a servo's output shaft.

- ❑ The plastic ball is used as the J-Bot's rear or tail wheel, and the cotter pin is its axle. Run the cotter pin through the holes in the tail of the J-Bot chassis so that it holds the one-inch plastic ball in place as shown in Figure 1.18 (a).
- ❑ Seat each o-ring tire in the groove on the outer edge of each plastic wheel.
- ❑ Each plastic wheel has a recess that fits on a servo output shaft. Press each plastic wheel onto a servo output shaft making sure the shaft lines up with and sinks into the recess.
- ❑ Use the machine screws that you saved when you removed the servo horns to attach the wheels to the servo output shafts.



(a)



(b)

Figure 1.18: (a), Tail wheel mounted on J-Bot chassis, and (b), front wheel mounted on servo output shaft.

Getting Connected

Figure 1.19 shows the parts you'll need to make your PC communicate with your Javelin.

Parts List:

- (4) 1.5 V AA batteries
- (1) Serial Cable
- (1) Parallax CD



Figure 1.19: parts you'll need before your first program.

Assembly:

Figure 1.20 shows the battery pack before and after the batteries are loaded.

- ❑ Load the batteries into the battery pack so that the polarity symbols on each battery match those printed on the inside of the battery pack.



Always use AA 1.5 V batteries.

Do not use 1.2 V (typically rechargeable) batteries.

If you are considering using an AC adaptor that you can plug into the wall to power your J-Bot, see page 2 for details on which ones are compatible with your servos.



1

Figure 1.20: Battery pack without/with batteries.

Figure 1.21 shows (a), the serial cable connected to a COM port on the back of a PC, and (b) the serial cable and battery pack connected to the BOE.

- ❑ Plug the female end of the serial cable into one of your computer's unused serial ports.
- ❑ Plug the male end of the serial cable into the DB9 socket on the BOE.

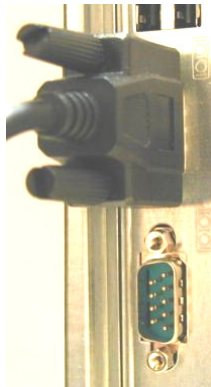
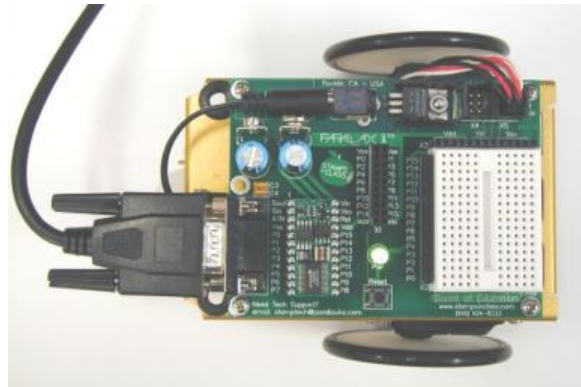


Figure 1.21: (a), Serial cable connected to com port,



(b) BOE connected to serial cable and battery pack.

- ❑ Plug the battery pack back into the BOE while watching the green light on the BOE for problems. Unplug the battery pack immediately if you see any of the warning signs listed below.

Warning Signs:



If the green light doesn't come on, looks unusually dim, or flickers, disconnect the battery pack immediately and check your wiring. Any of these warning signs could indicate a wiring problem that could be dangerous to your servo and/or your Javelin.

1

- ❑ To extend the life of your batteries, unplug the battery pack's barrel plug from the BOE's barrel jack. This will disconnect power from the Board of Education and the servo motors. You will need to plug the power back in when you are ready to run your first JAVA program in Activity #3.

Activity #3: Testing PC - J-Bot Communication

The Javelin IDE is the software you'll be using to program the J-Bot's Javelin on-board computer. The Javelin IDE has an integrated debugger and Messages window in addition to the ability to edit Java source files. You can use the Messages window to display messages received from the Javelin and also to send messages to the Javelin. Messages are sent to the Messages window using the `System.out.println` function.

If you have used the Basic Stamp IDE then you are in for a surprise. The Javelin IDE is much more powerful. It supports multiple breakpoints and it is possible to single step through a program. Global, local and stack variables can be examined. Overall, the development environment is on par with more sophisticated cross platform development tools.

Software and First Program

This section covers the steps for:

- Installing the Javelin IDE
- Using the Javelin IDE to establish PC - Javelin communication
- Running a sample JAVA program that uses the **System.out.println** method

Note: These instructions are for installing the Javelin IDE from the Parallax CD. A copy of the Parallax CD can be requested from stampsinclass@parallaxinc.com. You can also get the latest version of the Javelin IDE (It's free!) from the Downloads page of www.parallax.com.

Software

- ❑ If you have not already done so, load the Parallax CD into your computer's CDROM drive.

The Parallax CD has a browsing program called the Welcome application that runs automatically after the CD is placed in your computer's CDROM drive. Figure 1.22 (a) shows the browser as it appears the first time the CD is placed in the computer's drive. Figure 1.22 (b) shows the browser as it normally appears when you run the Welcome application.

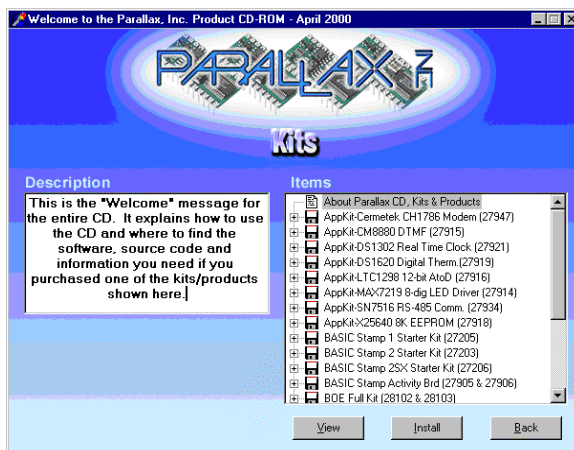
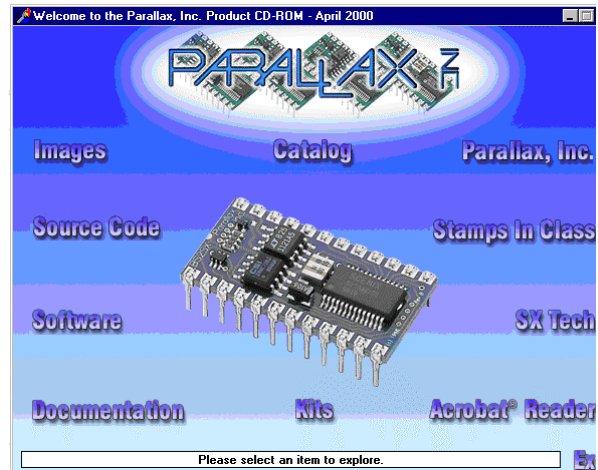


Figure 1.22: Welcome application (a)
Kits page, and



(b) Parallax page.

TI
P

If the Welcome application did not run automatically, here's how to run it manually: Click the Start button on your Windows taskbar and select Run. When the Run window appears, enter the CDROM drive letter, followed by a colon, a backslash, and the name "Welcome.exe."

For example, if the drive letter for your computer's CDROM drive is D, type in "D:\Welcome.exe." Click the OK button, and the Welcome application will run.

- ❑ If this is not your first time running the Welcome application, the Parallax page shown in Figure 1. (b) will display instead of the Kits page. Skip the next checklist instruction.
- ❑ If this is your first time running the Welcome application, a text document about the Parallax CD will automatically display. When you're finished reading the text document, minimize it or drag it out of the way so that you can see the Kits page. Click the "Back" button on the bottom right of the kits page to get to the Parallax page.
- ❑ Click the Software link.
- ❑ Click the + next to the Javelins folder, then click the + next to the Windows 95/98... folder, then click the diskette labeled Javelin Stamp.
- ❑ Click the Install button, and select "Yes" when the Confirm window asks you if you want to "Install selected files to "C:\Program Files\Parallax Inc\Javelin Stamp IDE"
- ❑ If additional prompts appear, answer them as directed.

After installing the software, run it by following these steps:

- ❑ Select Javelin Stamp IDE from the Start menu or double-click the Javelin Stamp IDE icon on the desktop

The Javelin IDE window, similar to Figure 1.23 (a), will appear next. It will help to know the version number of the software you are using before you check to see if the Javelin IDE is communicating with the Javelin.

- ❑ Click the Help menu and select About...
- ❑ When the About window appears, make a note of the Version number.

Before attempting to run your first program, it's important to check and make sure the Javelin IDE can communicate with the Javelin.

- ❑ Plug the battery pack's barrel plug back into the barrel jack on the JSDB. Verify the green light on the Board of Education lights up.
- ❑ Click the Project menu, and select Test Connection as shown in Figure 1. (b).

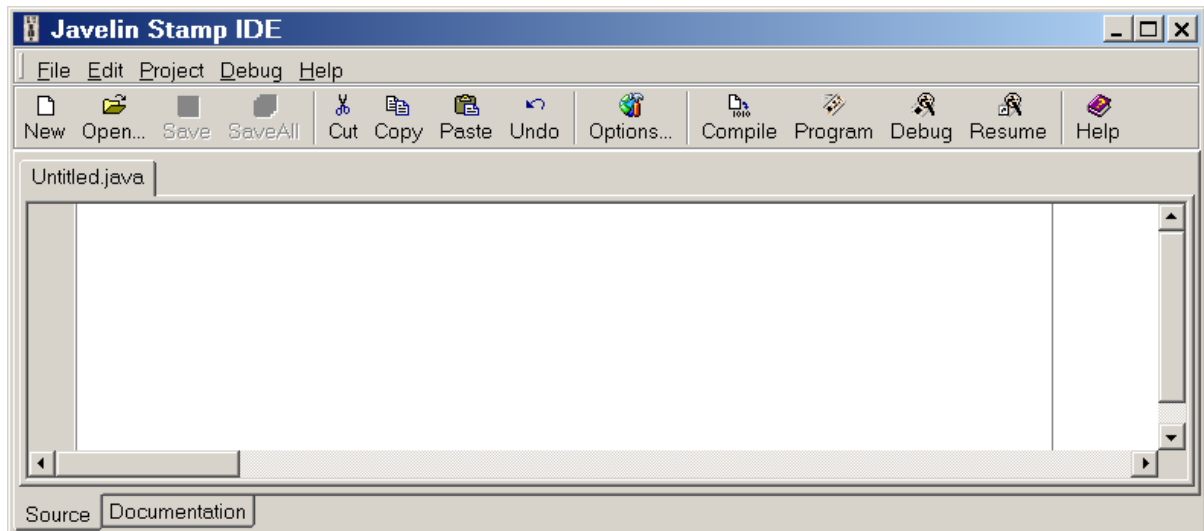
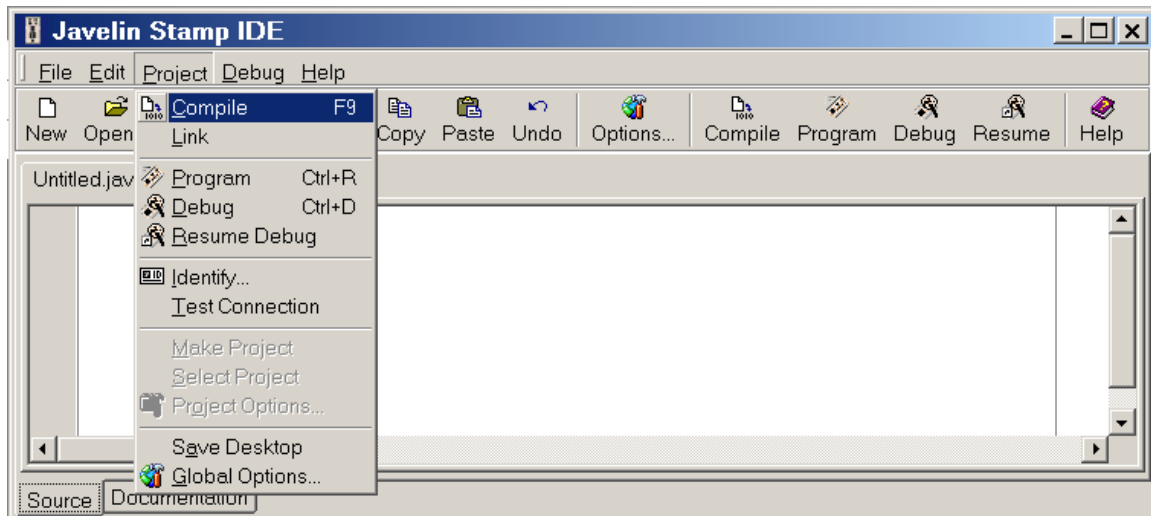


Figure 1.23: (a) Javelin IDE,



(b) Javelin IDE with Compile selected.

First Program

Your first program will demonstrate the Javelin's ability to communicate with the outside world using the Debug Terminal. This handy terminal can be used for two-way communication between your PC and the Javelin. For now, we'll focus on programming the Javelin to send messages to the PC.

- ❑ Type Program HelloWorld.java into the Javelin IDE as shown in Figure 1.24 (a).
- ❑ Click Project and select Program. The Message window should appear in a second window, as shown in Figure 1.24 (b).

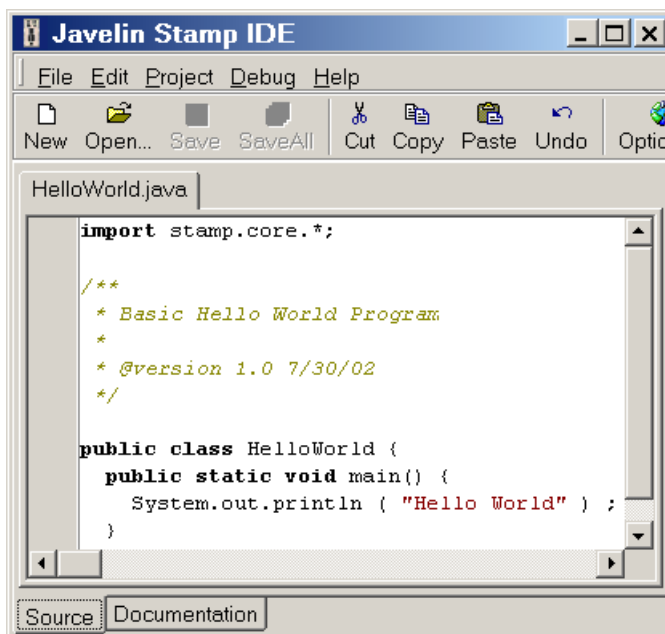
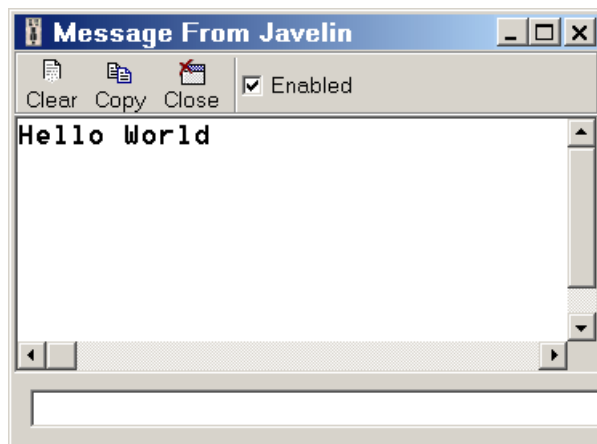


Figure 1.24: (a) Javelin IDE



(b) Messages window.

How "Hello world!" Works

Readers should be familiar with Java and the Java class structure. The `import` statement is used to indicate other classes that will be used by this class. In this case, the `System.out.println` statement found later in the file will use the `System` class. The following statement allows this class to be referenced.

```
import stamp.core.*
```

Java supports two types of comments. The first is shown in the `HelloWorld.java` program. In this case, `/*` and `*/` bracket the comment text. The `/**` is a special form that allows comments in the program to be used to generate documentation automatically. It uses a program called `javadoc` that scans the file and extracts these comments. It then takes this information and generates HTML documentation files. Special prefixes provide a way to insert specially formatted information such as the version number noted by `@version` as shown below.

```
* @version 1.0 7/30/02
```

The other form of comments prefix the comment text with `//`. This type of comment proceeds to the end of the line. If a comment is on the next line then another `//` must prefix it.

The next set of lines defines the `HelloWorld` class and the `main` method. A class that will be executed directly must have this method defined. Most other classes will not have this method. The `main` method is called when the program starts. In this case the following line is executed:

```
System.out.println ( "Hello World" ) ;
```

it sends the "Hello World" message to your PC by way of the serial cable. The "Hello World" message is a text string, which is one of several types of output data the Javelin can be programmed to send using the `System.out.println` function.

Your Turn

The best way to get a better feel for what you can do with the `System.out.println` function is to try the examples in the Javelin Stamp User's Manual.

Activity #4: Building and testing a Speaker Alarm - Intro to Virtual Peripherals

In this activity, you will program the Javelin to sound an alarm. This uses the pulse width modulation (PWM) virtual peripheral. The Javelin supports a number of virtual peripherals. Six may be active at one time. Virtual peripherals are covered in the Javelin Stamp User's Manual. Most will be used in this book at various points. Many can be used for different purposes. For example, the PWM virtual peripheral will also be used to drive the servos on the J-Bot.

The virtual peripherals supported by the Javelin are:

Background Virtual Peripherals

UART	Buffered serial port support
PWM	Pulse width modulation
Timer	32-bit timer
DAC	1-bit digital-to-analog converter
Delta/Sigma ADC	Analog-to-digital converter

Foreground Virtual Peripherals

Pulse Count	Count number of pulses
Pulse Width Measurement	Measure the width of a pulse
Pulse Generation	Generate a fixed length pulse
RC Timer	Measure RC discharge rate
SPI master	SPI communication link

Background peripherals operate in the background while the Java program is running. This provides a limited form of multitasking. Only six background virtual peripherals can be active at one time although one Timer virtual peripheral will support any number of Timer objects. The limitation is set because the Javelin memory used for the virtual peripherals is limited and not part of the RAM used for Javelin code and data.

The foreground virtual peripherals execute only when the appropriate method is called. For example, the pulse count virtual peripheral will count the number of pulses seen on an input pin for a fixed amount of time. The next instruction in the Java application will be executed when the time has expired. Any number of foreground peripherals can be created at one time.

Virtual peripherals are loaded into memory by creating a virtual peripheral object. To create an object you need to reference its class definition. This is done using the import statement. The virtual peripherals are in the stamp.core package. The following statement used in the Hello World program does the trick.

```
import stamp.core.*
```

All the background virtual peripherals are implemented via their own class. For example, the PWM class is used to create a pulse width modulation object. Some of the foreground virtual peripherals are created in the same fashion while others are implemented in the CPU class. In the latter case, the peripherals are accessed using static CPU class methods. For example, the pulse count virtual peripheral is accessed using:

```
CPU.count ( nTime, CPU.pin0, true ) ;
```

In this case, the parameters are the amount of time to wait, the pin to check and the edge to count. The last value indicates the rising edge is counted.

No Garbage Collection So Don't Make Garbage

The Javelin does not support garbage collection which is a feature found in most Java implementations. This means that once an object is allocated it will always take up memory space even when it can no longer be referenced. This is called a memory leak.

Memory leaks may not be a problem for small applications that run for a short amount of time because there is enough memory in the Javelin to handle these objects. Unfortunately, this can be a problem when a robot runs for a long period of time. Even a small memory leak of a few bytes can cause a program to terminate after a few minutes if the allocation occurs often.

This means that a good Javelin programming technique is not to dereference an object especially when an object like it may be needed later. In general, this means that all objects should be allocated when the program starts. If objects need to be created, used and then discarded then it is best to keep track of discarded objects in a list and new objects of that type should be allocated by removing an unused object from the list.

Virtual peripheral objects are like any other object and should be managed accordingly. For this chapter, it means that allocating a tone generator object when a tone needs to be generated is not such a great idea. Instead, create one tone generator object and use it throughout the life of the program that uses the tone generator.

How Tone Generation Works

The small speaker that comes in the J-Bot kit will be used with the PWM virtual peripheral to generate tones. While many tone generators use a sine wave to drive the speaker, the Javelin can only generate a square wave. This is not much of a problem though

and it is possible to include a capacitor in the circuit to improve the output.

Figure 1.25 shows the circuit that is with the speaker. Any of the Javelin's pins can be used but we use pin 2 for this example. Note that pins in Javelin Java are specified as CPU.pin2 instead of the number 2. Using a number can be a major problem with many Javelin applications because the parameter type used is normally an integer and 0 is a valid integer.

One of the wires on the speaker is connected to Vdd that is a positive 5 volts. The other is connected to pin 2. The tones are generated by raising and lowering the output value of pin 2. When the output is high then no current flows through the speaker. When the output is low then current flows through the speaker and the pin. A single pulse does not generate much sound because the time is too short to hear. This is why a number of pulses must be generated large.

Instead of sending a fixed number of pulses it is easier to send an arbitrary number of pulses for a fixed amount of time. This is how our tone generator object will work. Timing is easy with the Javelin so this approach is preferable.

About Time Measurements and Voltage levels

<p>Milliseconds and Microseconds</p> $1\text{ ms} = \frac{1}{1000} \text{ s} = 1 \times 10^{-3} \text{ s}$ $1\text{ }\mu\text{s} = \frac{1}{1,000,000} \text{ s} = 1 \times 10^{-6} \text{ s}$
--

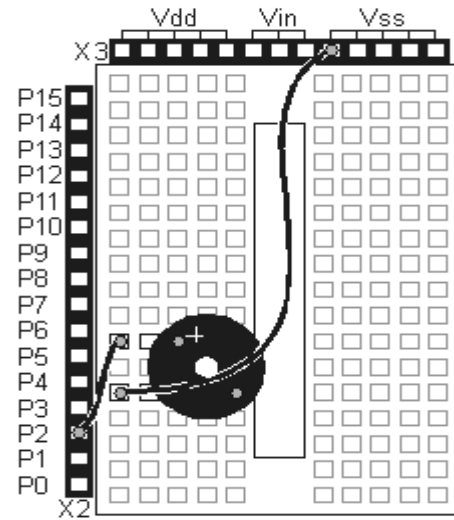


Figure 1.25: Tone generator wiring.

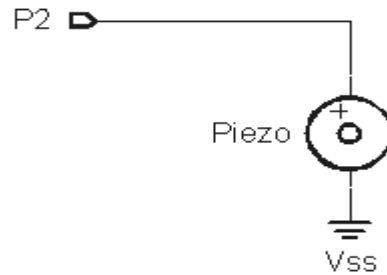


Figure 1.26: Tone generator schematic.

Voltages and BOE Labels

$V_{ss} = 0 \text{ V}$ (ground)

$V_{dd} = 5 \text{ V}$ (regulated)

$V_{in} = 6 \text{ V}$ (unregulated)

1

Throughout this student workbook, amounts of time will be referred to in units of seconds (s), milliseconds (ms), and microseconds (μs). Seconds are abbreviated with the lower-case letter s. So, one second is written as 1 s. Milliseconds are abbreviated as ms, and it means one one-thousandth of a second. One microsecond is one one-millionth of a second. The Milliseconds and Microseconds box to the right shows these equalities in terms of both fractions and scientific notation.

A voltage level is measured in volts, which is abbreviated with an upper case V. The BOE has sockets labeled V_{ss} , V_{dd} , and V_{in} . V_{ss} is called the system ground or reference voltage. When the battery pack is plugged in, V_{ss} is connected to its negative terminal. As far as the BOE, Javelin and serial connections to the computer are concerned, V_{ss} is always 0 V. V_{in} is unregulated 6 V, and it's connected to the positive terminal of the battery pack. V_{dd} is regulated to 5 V by the BOE's onboard voltage regulator, and it will be used with V_{ss} to supply power to circuits built on the BOE's breadboard.



Only use the V_{dd} sockets above the BOE's breadboard for the Activities in this workbook. Do not use the V_{dd} on the 20-pin app-mod header.

Now back to the tone generator. The PWM virtual peripheral sends a "pulse train," and an example of one is shown in Figure 1.27. The Javelin can be programmed to produce this waveform using any of its I/O pins. In this activity, we'll start with I/O pin 0. The high and low times within the pulse train will be the same. Other uses of the PWM virtual peripheral may use different times for each. For example, servo control uses short high pulses and long low times.

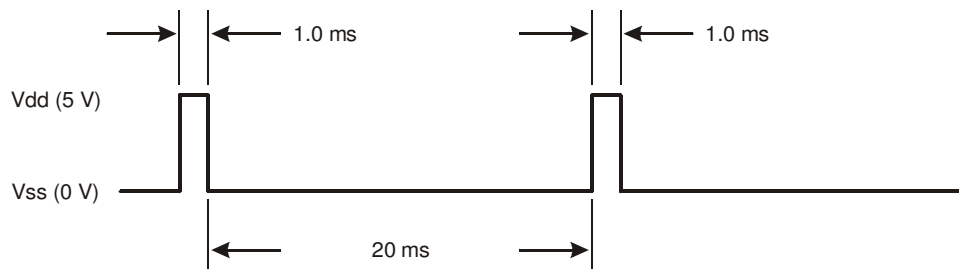


Figure 1.27: Pulse train.

While the name of the tone generator class might be `ToneGenerator`, we will be calling it `FREQOUT`. This is also the name of the command used in the Basic Stamp programming language. We will be defining another tone generator for our multitasking environment and use a more descriptive name there.

Remember Pulse cycle time is what controls the tones being generated. Lower tones have a longer cycle time. Higher tones have a shorter cycle time.

The `FREQOUT` class in the `FREQOUT.java` file extends the `PWM` class. It can be found in the `stamp.core` package versus the Hello World program that is in the sample program listings. This allows other applications to easily use the class. The following is the source code for the `FREQOUT` class.

The `FREQOUT.java` file starts out with a javadoc comment between the `/**` and `*/` comment specifiers. This information will be contained in the Java documentation about the `FREQOUT` class. Next there is the package statement.

```
package stamp.core;
```

This indicates that the class will be part of the `stamp.core` package. It means that the `FREQOUT` class can use any other classes in the package including the `PWM` class which it extends. This is the next statement:

```
public class FREQOUT extends PWM {
```

Within the outer brackets is the class definition. The `FREQOUT` class supports all the `PWM` methods so a `FREQOUT` object can be used like a `PWM` object. The `FREQOUT` class also defines its own set of methods in addition to using the methods of the `PWM` class. The entire list of `PWM` methods can be found in the online help. We discuss only those used with the `FREQOUT` class.

Note that each method has a javadoc comment (`/**` to `*/`) in from of it. This is how the online documentation is created. The general format is a short description followed by the parameters to the method and then a description of a return value if the method returns a value. The parameter comments are prefixed by `@param` while the return values are described using `@returns`.

The first method is the `FREQOUT` constructor. It does not return a value. Its parameter is the pin to be used for output. It calls the super method which is the constructor for its superclass, the `PWM`. This is the usual constructor used with the `FREQOUT` class because it does not start sending out pulses on the designated pin until subsequent methods are called.

The next method is also a constructor that uses the same arguments as the `PWM` constructor. If this constructor is used the `PWM` virtual peripheral object will start sending pulses immediately.

Next we get to a useful method that can be called after a `FREQOUT` object is created. This is the `setFrequency` method. It takes one parameter, the frequency in hertz. It uses this value to calculate the cycle time. There is a check for a zero parameter value to prevent a divide by zero error then the `halfCycleTime` is set. This is a reasonable alternative to use the more complex catch/throw error handling semantics of Java. The cycle time is then used with the `PWM` update method. Note that the method can be called without an object reference since it the `PWM` object is part of a `FREQOUT` object.

Setting the frequency will only cause a tone to be generated if the `PWM` object within `FREQOUT` object is running. This is done using the `PWM` start method. It is possible to use the `FREQOUT` object using this combination but it is easier to use the next method, `freqout`.

The `freqout` method takes two parameters. The first is the frequency that will be passed to the `setFrequency` method call. The `PWM` virtual peripheral is then started. This generates a series of pulses until the stop method is called. This will be time milliseconds later because the `CPU.delay` method is called. This method uses the delay virtual peripheral that is built into the `CPU` class that is also part of the `stamp.core` package.

This class can be compiled by opening it in the Javelin IDE and then selecting Project menu and then the Compile item. This generates the `FREQOUT.class` file. This is already done when the class was installed since it is part of the standard package so we don't have to do it here. Instead we can move onto the sample program that utilizes the `FREQOUT` class, `TestFREQOUT.java`.

This is a short program because most of the work is being done by the `PWM` and `FREQOUT` classes. The import statement is included so the `PWM` and `FREQOUT` classes in the `stamp.core` package are available. The `System` class is also available. There is the usual javadoc comment at the beginning of the program followed by

the TestFREQOUT class definition. Note that this class does not extend another class. Also, unlike the PWM class, there is no constructor method, only the main method which is defined as public static void. The inclusion of the static keyword means that this is a class method versus an object method used in the FREQOUT class. This particular format is necessary for any main application such as the Hello World application discussed earlier.

The main method creates a new FREQOUT object and the reference to it is called freqout. The names can be the same because Java is case sensitive otherwise the object reference variable would conflict with the class name.

In any case, we now have a FREQOUT object that is associated with pin 0. The System.out.println will display the text string in the message window and then the program will cause two different short tones to be generated. The program then ends.

The program can be run again using the Javelin IDE or it will run again if the reset button on the BOE board is pressed.

In general, this is how applications will be constructed in this book. The classes for the objects needed for an application will be defined first. The main program file with a main method will be defined last. This program can then be run to test the other classes. For now, we wrap up this first exercise.



Summary and Applications

Congratulations on the construction and operation of your J-Bot! Through following the procedures in this chapter, you may have had your first taste of testing and troubleshooting at the system and subsystem levels. Lots of other essential topics were covered that will

get used and re-used throughout this text. For example, the Message window will be your best and most used tool for testing and troubleshooting each circuit as well as many upcoming programs.

The JAVA programming language was introduced along with some example programs to get you started with the Message window and with the J-Bot. The Javelin also supports more advanced debugging features like breakpoints and single stepping. These are covered in more detail in the Javelin Stamp User's Manual. Check it out if you have not already used these debugging features.

Real World Example

From the space shuttle all the way down to the J-Bot, isolating and testing subsystems during each phase of development is critical to make sure the whole thing runs when it's put together. More importantly, isolating and testing each subsystem minimizes the time spent on, and difficulty level of, troubleshooting. At the beginning of the chapter, the problems associated with not iteratively developing and testing were discussed. Imagine if nobody tested the Space Shuttle's subsystems before putting it together. It would take hundreds of years for NASA to get all their problems sorted out!

Whether it's robotics competitions, product development, or space programs, subsystem and system level development and testing is the way to avoid unnecessary delays when working from the beginning to the end of a project. Especially in product development, groups of engineers develop systems and subsystems. Often, it's not until late in the design cycle that the system level testing and system integration occurs. Sometimes, all a design team knows are the input and output (I/O) requirements of their particular module in the project. Regardless, engineering design teams still have to iteratively develop, simulate (which we did not do here), and test the subsystems within the project module they are working on.

J-Bot Application

One item you'll investigate in the Questions and Projects section is what happens when the wiring of the servos gets changed. How

does this get handled? It involves more changes than you might think. For example, if you were to unplug a servo from servo port 12 and then plug it into servo port 15, you can't just change the drawing that shows what port to plug it into. The schematic, which is the preferred method of communicating wiring information, has to be changed, but so do all the program listings. At some point you might want to add more servos to function as grippers. Although a gripper design is not included in this text, Questions and Projects has exercises that will prepare you for connecting servos to different ports.



Questions and Projects

Questions

1. Explain the two things the `System.out.println` command does.
2. What are the different types of output data that can be used with the `System.out.println` command? Hint: Use the Javelin Manual to answer this question.
3. The pulse widths in the example program are generated based upon a desired frequency. What happens if the frequency computation is changed?
4. Why did the `FREQOUT` class extend the `PWM` class?

Exercises

1. Add a filter capacitor across the speaker. Does this improve the sound quality?
2. Determine the frequency range of useful tones. Tones at the lower and upper end of the spectrum may not be rendered as well as those in the middle of the spectrum.
3. Implement a `NewFreqout` class that provides the same tone generation capability as `FREQOUT` but does not extend the `PWM` class. Note, the `NewFreqout` class does not have to implement all the `PWM` methods, only the `FREQOUT` `setFrequency` and `freqout` methods.

Hint: Allocate a `PWM` object and assign it to an object variable.

Projects

1. The pulse width generation in the examples is based on frequency but must musical instruments generate only a fixed number of frequencies designated as notes. Create an application that uses logical note names to generate tones.

Hint: Retain and extend the `PWM` tone generator class. Use the Java switch statement to determine the note to be played and its matching frequency.

