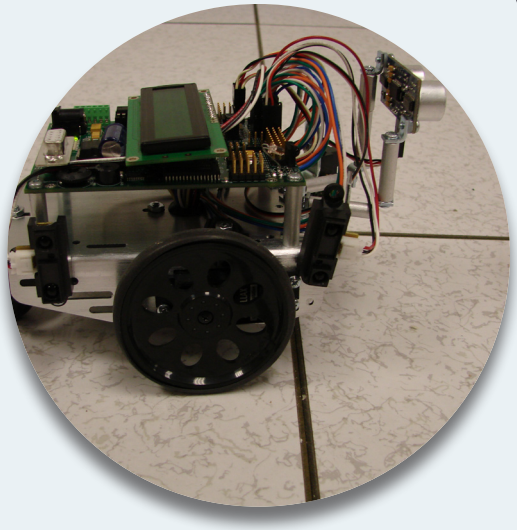
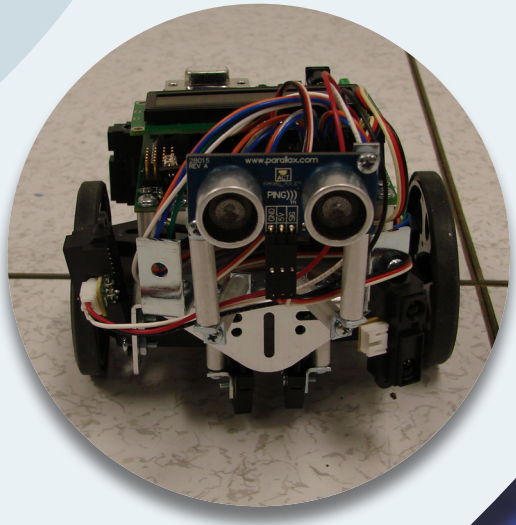
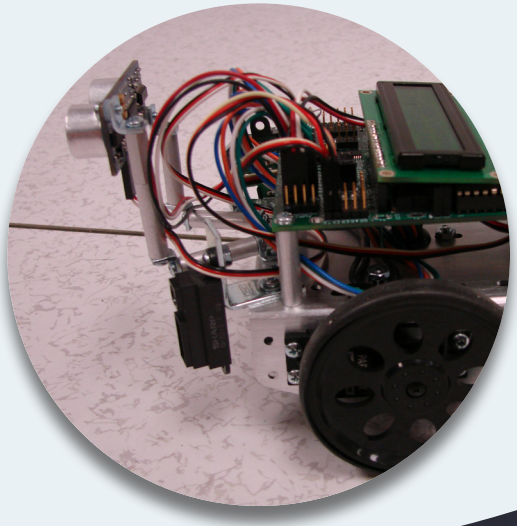
Camp Guide



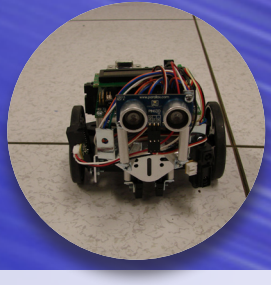




Robotics 08

- Revision 1 - June 08

Pre-Camp
PREPARE



Materials Needed

Day One: Robots, Compatible Computers, Student Files , Keynote Files
Day Two: Robots, Compatible Computers, Student Files , Keynote Files
Day Three: Robots, Compatible Computers, Student Files , Keynote Files, Doorway, Dark Colored Paper
Day Four: Robots, Compatible Computers, Student Files , Keynote Files, Boxes
Day Five: Robots, Compatible Computers, Student Files , Keynote Files, Maze

The Model Bot

The robot used to test the demo and exercise files is a standard Intellibrain Bot with the following modifications:

1. Front IR Range Sensors Moved to Right Side of Bot (See Picture in Day One Keynote)
2. Sonar Sensor Connected to DigitalIO 4

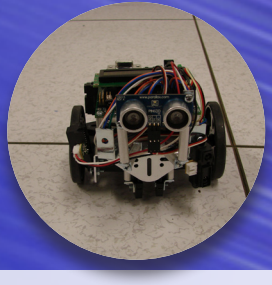
Connections:

Right Side Front IR Range Sensor - Analog Input 2
Right Side Rear IR Range Sensor - Analog Input 1
Servo Left Wheel - getServo(1)
Servo Right Wheel - getServo(2)
Sonar Range Finder - DigitalIO 4
Left Line Sensor - Analog Input 6
Right Line Sensor - Analog Input 7

RoboJDE Requirements

Robo JDE requires Windows XP or 2000 computers only. Computers must also have a Serial (RS232) port in order to connect with the robot.

Pre-Camp
PREPARE



Building A Maze

The last day's activity requires you to build a maze. When building your maze keep in mind that the hallways of the maze should be around 16 inches wide at all points. The Intellibrain lacks accuracy and must have space to move about. The other restriction is that all walls the Intellibrain follows must be at least 8 inches long continuously. If the walls are not long enough, the Intellibrain will not have time to register them and will react in error. You can see sample maze plans in the **Extras** folder of the Resource Disc.

How To Use This Guide

This guide provides many resources which will help you perform the programming portion of the camp. Each day has a Prepare section before it, followed by goals of the day, a keynote outline, and then a line by line analysis of the code from that day.

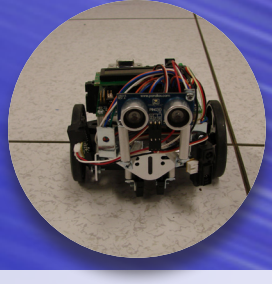
The keynote outline section is color coded. Text in black is written as it could be spoken to students. Text in green informs the leader of additional information and should not be read aloud.

Notes on Resource Disc

The Resource Disc provides you with all of the files you will need for the programming sessions. Keynote presentations are availible in many different formats (Key, PDF, HTML, Flash, Quicktime, Image Sequence, and PPT), unless you need to make a change, it is best to NOT use the .ppt versions.

Other resources include student files, solution files, and video demos of the hands on exercises. Additionally, in the Extras folder you will find the StudentWeb folder. This is a folder which can be uploaded to a web server. It will allow students to download the files they need in a zip format for each day, as well as give access to HTML viewing of each day's keynote.

Day One
GOALS



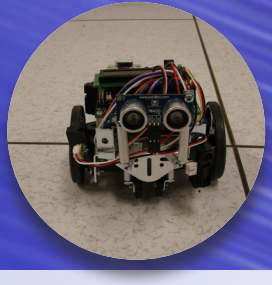
Key Concepts

- Introduction to Java
- Introduction to RoboJDE
- Discovery of Intellibrain Abilities
- Running a Program on the Intellibrain

Goals of the Day

- Writing a Program
- Downloading a Program
- Running a Program
- Using Servos & LCD Display

Day One
PREPARE



General

Before day one, you will need to review the general prep materials for the camp. You can find these materials at the beginning of this guide. Day One is basically an introduction to Java, as well as the RoboJDE environment. Your main goals for this day are to teach students how to use RoboJDE, write three simple programs and run the three programs. It is a good idea to review the keynote slides and outline ahead of time as well. You will need to be familiar with all the programs, demos, and examples ahead of time. You will also need to be able to fully explain RoboJDE.

Demos

Day One contains one demo. This demo is purely for the purpose of demonstrating the functionality of RoboJDE and the downloading process. You need to plan this demo on your own, as you should write it from scratch in front of the students. It should be something simple and easy to write quickly, but also demonstrate functionality of the robot. A good idea would be to write a program which makes the robot move forward. You can find an example program in the **line by line** section following day one.
NOTE: Please do NOT simply open this program and run it during the demo, this demo needs to show the full process.

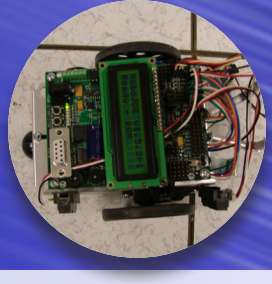
Programs

Students will need the **DayOne** folder found in the **StudentFiles** section of the Resource Disc. This folder contains files related to the programs:

HelloWorld - displays text to the LCD Display
DriveAround - drives the robot around using servo controls
Beeper - makes the robot beep as it drives in reverse

You need to review these programs ahead of time so you can assist students. As usual you can find a detailed description of the code in the **line by line** guide. Additionally, you can review the solution files for insight into the programs.

Day One
BEGIN



Getting Started

Get started by welcoming everyone to the programming/lab portion of the camp. Take some time to get students setup with computers, logins, student file packs and any other material they will need. Explain the purpose of the hands on programming portion of the camp, and share some of the exciting things you have done or seen done through programming. Once everyone is ready, begin the session by following the Day One Keynote outline.

Distribution of Materials

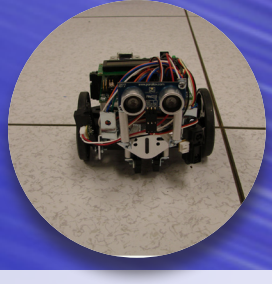
You may choose to distribute robots before you start the Day One session. A full guide concerning the materials each student needs, as well as the instructions for assembling our specific robot are available in the **Pre-Camp** section. Robots will be needed for Day One activities, so if you do not choose to distribute them ahead of time, you will need to do so during the presentation.

Student Files

Students will need the **DayOne** folder found in the **StudentFiles** section of the Resource Disc. This folder contains files related to the programs:

HelloWorld, DriveAround, and Beeper

Day One
KEYNOTE



Keynote

The Keynote for Day One is found on the Resource Disc at the path **/Presentations/Day-One/**. Today's keynote contains 26 Slides and will take an estimated hour to hour and a half to complete. Note that examples are embedded in the keynote.

Keynote Outline

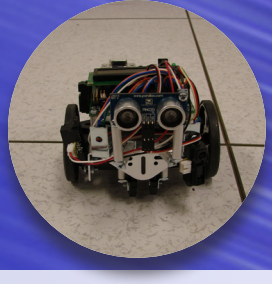
Slide One - Title

Slide Two - Introducing Intellibrain

Take some time to discuss the Intellibran bot's design, don't go too far into discussion as the next slides also discuss the bot's functionality.

Slide Three - Introducing Intellibrain

The Intellibrain was designed by RidgeSoft mainly for educational use, as well as for hobbyist. RidgeSoft has created a custom development environment known as RoboJDE which allows the Intellibrain to be programmed in the Java programming language. The RoboJDE is available for download at RidgeSoft.com: the lite version is free and has some limitations as to the complexity of your program, you can also purchase a license for the full version. While the Java language is cross platform, RoboJDE, at the time of writing, only runs on the Microsoft Windows operation system.



Keynote Outline(cnt'd)

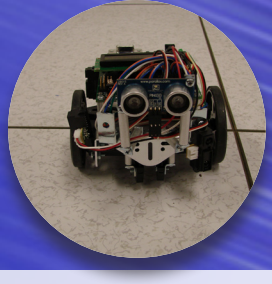
Slide Four - What Can it do?
The Intellibrain has many different forms of input and output available to it. You can add different components by simply mounting them to the robot's frame and then attaching the cable to the appropriate connector.

Main Output Devices
SERVOS - Servos are motors used to move the wheels of the robot. The can move forward and backward at different intensities.
LCD DISPLAY - The LCD display is capable of displaying lines of text to the two lines of the display built onto the Intellibrain main board.
SPEAKER - The speaker is capable of playing different frequencies of sound.

Main Input Devices
SONAR - The sonar sensor is mounted on the front of the robot, allowing it to read the distance between it and an object it is approaching. The sonar sensor can read fairly accurately between the ranges of 3 inches and 20 feet .
INFRARED - The infrared sensors also measure range between the robot and an object. The are less accurate than the sonar sensors. Infrared sensors use light and angles to calculate the reading of distance. Infrared sensors work best at distances 4 inches to 31 inches
PHOTO REFLECTORS - Photo reflectors are able to recognize a change in color or intensity of an object being read.

Slide Five -How?
The intellibrain is programmed in Java via the RoboJDE environment. The program is compiled, then downloaded to the robot's flash or ram memory via a DE-9 (Serial) cable. Once the program is loaded it can then be run.

Slide Six -Taking a Look At Java
Java is a higher level programming language first released in 1995. Java is owned by Sun Microsystems and continues to grow in ability and efficiency with each release. The language is partially based on C or C++. Java is a programming language used to demonstrate the abilities of Object Oriented Programming. It is easy to use, but can become fairly complex. Java has become widely accepted.



Keynote Outline(cnt'd)

Slide Seven - Java Characteristics

Java is exceptional in that it is a cross-platform language. This means that Java will run on a variety of operating systems, without any extra effort from the programmer. Java is programmed in a easily understandable syntax, then converted by the complier into a byte-code, which is then interpreted by the Java Virtual Machine to run the program. The Java Virtual machine exist for many different platforms. As a rule: if a system has a Java Virtual Machine, Java programs will run on it.

Slide Eight - Java Characteristics

Error Checking

Java provides extensive error checking both at compile time and runtime, allowing for solid program development. Errors at runtime are easily manageable and traceable.

Garbage Collector

The garbage collector automatically cleans up behind your program, adding both security and efficiency. The collector frees up memory that is not needed and adds security while doing so.

Access Restrictions

Java posses many utilities, some controlled by the user, some buried in the system, which allow data to be accessed by only the appropriate programs, users, etc.

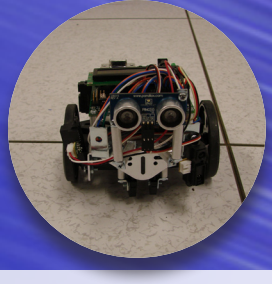
Slide Nine - Java Characteristics

Multi threaded

Java allows the programmer to create multiple threads, running different functions simultaneously. Management of these processes is done by Java automatically, the programmer can also declare priority for the tasks.

New Libraries

The Java language is always expanding, as anyone can write libraries for use with it. This allows the language to grow, and the code to be reused-- making programming easier.



Keynote Outline(cnt'd)

Slide Ten - Java Data Types

Extreme details provided, in case of questions. A brief summary should be sufficient.

From Java Site (http://java.sun.com/docs/books/tutorial/java/nutsandbolts/datatypes.html)

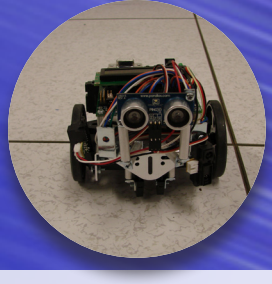
- byte: The byte data type is an 8-bit signed two's complement integer. It has a minimum value of -128 and a maximum value of 127 (inclusive). The byte data type can be useful for saving memory in large arrays, where the memory savings actually matters. They can also be used in place of int where their limits help to clarify your code; the fact that a variable's range is limited can serve as a form of documentation.

- short: The short data type is a 16-bit signed two's complement integer. It has a minimum value of -32,768 and a maximum value of 32,767 (inclusive). As with byte, the same guidelines apply: you can use a short to save memory in large arrays, in situations where the memory savings actually matters.

- int: The int data type is a 32-bit signed two's complement integer. It has a minimum value of -2,147,483,648 and a maximum value of 2,147,483,647 (inclusive). For integral values, this data type is generally the default choice unless there is a reason (like the above) to choose something else. This data type will most likely be large enough for the numbers your program will use, but if you need a wider range of values, use long instead.

- long: The long data type is a 64-bit signed two's complement integer. It has a minimum value of -9,223,372,036,854,775,808 and a maximum value of 9,223,372,036,854,775,807 (inclusive). Use this data type when you need a range of values wider than those provided by int.

- float: The float data type is a single-precision 32-bit IEEE 754 floating point. Its range of values is beyond the scope of this discussion, but is specified in section 4.2.3 of the Java Language Specification. As with the recommendations for byte and short, use a float (instead of double) if you need to save memory in large arrays of floating point numbers. This data type should never be used for precise values, such as currency. For that, you will need to use the java.math.BigDecimal class instead. Numbers and Strings covers BigDecimal and other useful classes provided by the Java platform.



Keynote Outline(cnt'd)

double: The double data type is a double-precision 64-bit IEEE 754 floating point. Its range of values is beyond the scope of this discussion, but is specified in section 4.2.3 of the Java Language Specification. For decimal values, this data type is generally the default choice. As mentioned above, this data type should never be used for precise values, such as currency.

- boolean: The boolean data type has only two possible values: true and false. Use this data type for simple flags that track true/false conditions. This data type represents one bit of information, but its "size" isn't something that's precisely defined.

- char: The char data type is a single 16-bit Unicode character. It has a minimum value of '\u0000' (or 0) and a maximum value of '\uffff' (or 65,535 inclusive).

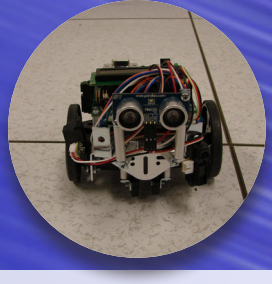
Class type - custom made by user, or supplied by Java library. Consist of many primitives, class types, and methods.

Slide Eleven - Class Types

Class types consist of a collection of data and functions which can be performed on data. Class types may have data members of primitive types or even other Class types. Additionally Classes can also contain functions, known as methods, which perform some sort of operation on data members or on supplied values.
Class types greatly help relate computer science tasks to real life situations, as well as manage data easily.

Slide Twelve - Class Type Example

Here is an example of a CellPhone Object. It represents something a cell phone company might use in a customer management program. For example, each cell phone in real life has a phone number, serial number, activation state, and a reference to a Account Object. An account object might store some values including the customers first name and a reference to the customers latest bill object. The bill object might contain a ID and an amount due. This isn't a complete example, but it does demonstrate the ability of Java to easily relate to the real world.



Keynote Outline(cnt'd)

Slide Thirteen - Class Type Example

Take a look at the two methods now included in the CellPhone object. One is used to set the activation state, based on a value supplied to it. The other returns a reference to the customer object associated with the phone to whatever calls it. These methods are both related to an object and can only be called by an object of that class.

Slide Fourteen - A Java Program

This is an example of a simple java program and its output. Lets look at the different parts of the program:

Comments

Comments can go anywhere in your program, they are ignored by the complier you can denote a line for a comment by a "//" or by starting with a "/*" to open a multi-line comment and then closing it with a "*/"

Import Statements

Import statements are used to tell java which libraries it many need to access in order to complete some of the functions in the program. The import in this example isn't needed for this program to run, but in our robot programs we will need imports.

Public Class

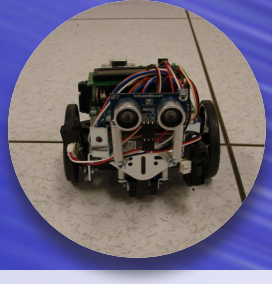
The next line opens the program by declaring "public class" followed by the file name of the Java file. Everything in the program must be enclosed in the {}'s after this statement.

Main Method

The main method is always declared just as it is shown, though it is legal to make a few variations it's best to keep it this way. Next there are opening braces again, and the body of the program continues. Everything in the main method is automatically executed as soon as the program is executed. Other methods can follow this method, but they are not executed unless called. The main method returns no value, it is void.

String name=

A variable is declared by first typing the type, then a name, then = , the value, and finally the semicolon.



Keynote Outline(cnt'd)

sayHi()
This is a reference to the method below we will look at this later.

System.out.println()
This line outputs some text to the screen

sayHi() Method
The sayHi method simply calls the println method and outputs the text Hi!

Slide Fifteen- RoboJDE
Take some time to explain RoboJDE. This keynote guide does NOT provide a full set of instructions for RoboJDE, only the highlights of what we will use. You need to be familiar with RoboJDE prior to this session so that you can cover the areas listed below. For a full guide concerning RoboJDE you can visit (Warning: PDF link):

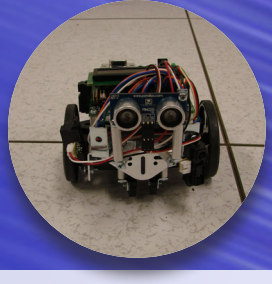
http://www.ridgesoft.com/articles/creating1stprogram/CreatingYourFirstIntelliBrainProgram.pdf

Launching RoboJDE
Show the students how to launch RoboJDE and then begin to explain the interface.

Project v. File
Explain to students the difference between a project file and a file which is a member of a project. There can be many .java (program) files for each .rjp (project) file. Demonstrate how to create a project with one file.

Setting Main Project File
Remind students that RoboJDE must be told which .java file is the main file in the project. Show the students how this is done at the time of project creation, and how it can be changed.

Baud Rate & Connection Port
Demonstrate to students how to change the Baud Rate as well as how to tell RoboJDE which COM port to communicate with.



Keynote Outline(cnt'd)

Ram v. Flash
Explain the difference between flash and RAM memory in the Intellibrain. You will probably only want to use the RAM option during camp, as the robots are limited to the number of times they can be flashed and require extra steps to be flashed successfully.

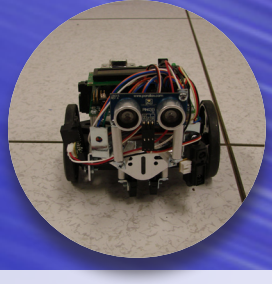
Downloading
Show students the button which is used to download to the robot.

Compiling a File
Show students how to compile a file and where to find the compiler output.

Don't Worry
Tell students not to worry if they couldn't keep up or remember all of the functions, as you will be doing a complete demonstration of the process on the next slide.

Slide Sixteen - Demo
The demo process is entirely up to you. Use the plan you created during the planning time for this day. Remember it is best to do something simple with your robot. Be quick, but take time to answer questions. Also, remember that today's activities end at moving forward and backward using servos, so any questions asked about other operations can be postponed to save time.

Slide Seventeen - LCDDisplay
Gaining access to the LCD display requires importing the library. Followed by creating an object of the type Display is the next step. You can now use the object with the print statement: giving first the line number as an int and then the text as a string.



Keynote Outline(cnt'd)

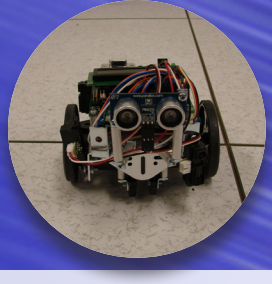
Slide Eighteen - Thread.sleep();
Java executes programs line by line, as quickly as it can, one line and then the next until the end of the program. This is a good thing; however Java can execute our robot's code very quickly in most cases. This sounds positive but consider this:

We create a program which contains one output statement to the screen of the robot, once this statement is executed there are no more statements. Java will execute the statement and then exit the program. This will happen so quickly we will not have time to read the output.

To fix this problem we need to tell Java to not terminate the program after the output, giving us time to read the screen. We could fabricate other statements which would take time to execute, but that could create more problems and unwanted results; additionally this is a messy way to approach the problem. Instead we can simply ask Java to sleep for an amount of time before moving on to the next statement. We do this by calling Thread.sleep(); and supplying an amount of milliseconds.

Slide Nineteen - Hands On Exercise - HelloWorld.java
You will now need to instruct students to begin the first hands on exercise, **HelloWorld.java**. The templates for the students are available in the **DayOne** folder in the **StudentFiles** folder on the Resource Disc. You should already have an understanding of how this program works, but if you run into trouble you can always consult the **line by line guide**, found at the end of the Day One section of this guide. Additionally the solutions are provided in the **DayOne** folder in the **Solutions folder** on the Resource Disc.

Slide Twenty - API
Java libraries are documented by APIs detailing their functions and how to use them.
You can explore the RidgeSoft API in order to see what you can do with the intellibrain. Demonstrate to students how to look up something in the API. Example: Display
1) Launch API via RoboJDE (Blue Book on toolbar)
2) Find Display link on lower left frame of page and click on it
3) Explore different methods available



Keynote Outline(cnt'd)

Slide Twenty-One - Servos
Servos or servo motors are used to move the robot. Each wheel is connected to a servo motor. Servo motors are controlled by setting a value from 0 to 100. 0 is full speed backwards, 50 is stopped, and 100 is full speed forward. Because one motor is mounted backwards on the robot, one motor's forward will be 100 while the other motor's forward will be 0. You can experiment in order to determine which is which.

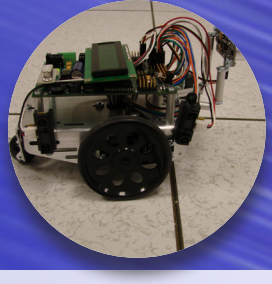
Slide Twenty-Two - Using Servos
Using a servo is similar to using a display. First import the library, then create an object for each motor. You can then control each motor using the setposition method or the off method.

Slide Twenty-Three - Hands On Exercise - DriveAround.java
You will now need to instruct students to begin the second hands on exercise, **DriveAround.java**. The templates for the students are available in the **DayOne** folder in the **StudentFiles** folder on the Resource Disc. You should already have an understanding of how this program works, but if you run into trouble you can always consult the **line by line guide**, found at the end of the Day One section of this guide. Additionally the solutions are provided in the **DayOne** folder in the **Solutions folder** on the Resource Disc.

Slide Twenty-Four - Producing A Beep
You can use the Intellibrain's speaker to create sounds of different frequencies. Same procedure as the LCD Display, and Servos. Import the library, create the object, and then use it. The first parameter on play() is used for tone, the second for duration. play(tone,duration)
Slide Twenty-Five - Hands On Exercise - Beeper.java
You will now need to instruct students to begin the third hands on exercise, **Beeper.java**. The templates for the students are available in the **DayOne** folder in the **StudentFiles** folder on the Resource Disc. You should already have an understanding of how this program works, but if you run into trouble you can always consult the **line by line guide**, found at the end of the Day One section of this guide. Additionally the solutions are provided in the **DayOne** folder in the **Solutions folder** on the Resource Disc.

Slide Twenty-Six - End Day One

Day One
WRAP UP



Wrap Up

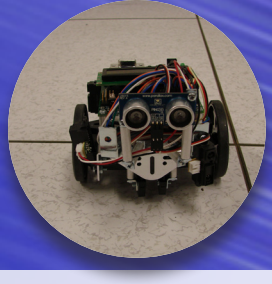
Once students have finished completing the exercises take questions and work to clear up any difficulties which were experienced. Tomorrow students will begin using control structures (loops and conditions) to make their robot behave in more complex manners. Inspire the students to return tomorrow by letting them know tomorrow they will learn how to make their robot turn and drive more efficiently.

Line by Line - Day One

DayOne_Demo1_Example.java

Source And Description:

```
import com.ridgesoft.io.Display; //Import Display library
import com.ridgesoft.intellibrain.IntelliBrain; //Import Intellibrain library
import com.ridgesoft.robotics.Servo; //Import Servo library

public class DayOne_Demo1_Example //Open the class
{

   public static void main(String args[])  // Open the main method
        {try{ //Open the try block, since we are using Thread.sleep();

        Servo leftWheel = IntelliBrain.getServo(1); //An object for the left Servo
        Servo rightWheel = IntelliBrain.getServo(2); // An object for the right Servo

        leftWheel.setPosition(100); //left wheel full speed forward
        rightWheel.setPosition(0); //right wheel full speed forward, because mounting is 0

        Thread.sleep(5000); // Sleep for 5 Seconds
        leftWheel.off(); // Turn left servo off
        rightWheel.off(); // Turn right servo off
   }catch(Throwable t){ t.printStackTrace();}} // Close the main method
} //Close the Class
```



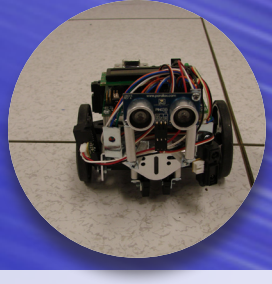
HelloWorld.java

Source And Description:

```
import com.ridgesoft.io.Display; //Import Display library
import com.ridgesoft.intellibrain.IntelliBrain; //Import Intellibrain library

public class HelloWorld //Open the HelloWorld class
{
        public static void main(String args[]) //declare the main method
        {try{                                   //open main method and try block, since we have
                                                //Thread.sleep(), we need a try block.

        Display display = IntelliBrain.getLcdDisplay(); //Create a display object

        display.print(0, “Hello”);  //Print to the display’s top line
        display.print(1, “World!”); //Print to the display’s bottom line

        Thread.sleep(5000); //Sleep the program for 5 seconds, so we can read the output



   }catch(Throwable t){ t.printStackTrace();}} //Close the try block and main method

}       //Close the HelloWorld class
```

Day One

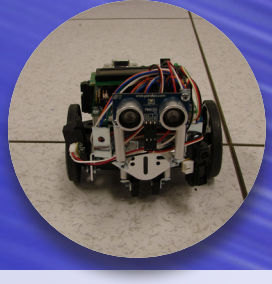
LINE BY LINE



DriveAround.java

Source And Description:

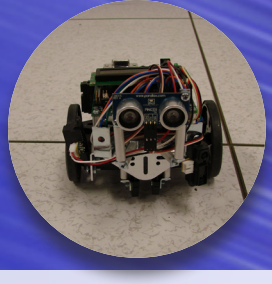
```
import com.ridgesoft.io.Display; //Import Display library
import com.ridgesoft.intellibrain.IntelliBrain; //Import Intellibrain library
import com.ridgesoft.robotics.Servo; //Import Servo library

public class DriveAround                    //Declare the DriveAround Class
{                                           //Open the Drive Around Class
        public static void main(String args[])          //Declare the main method
        {try{                               //Open main method and try bock, for Thread.sleep()

         Display display = IntelliBrain.getLcdDisplay();      //Create a display object
        display.print(0, “Drive Around”);                     //Output to first line
        display.print(1, “Starting….”);                       //Output to second line

        Servo leftWheel = IntelliBrain.getServo(1);           //Create a servo object
        Servo rightWheel = IntelliBrain.getServo(2);          //Create another servo object

        display.print(1, “Driving - F”);                      //Output to second line

        leftWheel.setPosition(100);             //Set left wheel to full speed forward
        rightWheel.setPosition(0);              //Set right wheel to full speed forward

        Thread.sleep(5000);                     //Sleep for 5 seconds

        leftWheel.off();                        //Stop the left wheel
        rightWheel.off();                       //Stop the right wheel

        Thread.sleep(500);                      //Sleep for half a second

        display.print(1, “Driving - R”);        //Output text to second line
```



DriveAround.java - Continued

```
            leftWheel.setPosition(0);           //Set left wheel full power backwards
            rightWheel.setPosition(100);        //Set right wheel full power backwards

            Thread.sleep(5000);                 //Sleep for 5 seconds

            leftWheel.off();                    //Stop left wheel
            rightWheel.off();                   //Stop right wheel

        }catch(Throwable t){ t.printStackTrace();}}  //Close main method and try block

}       //Close DriveAround class
```

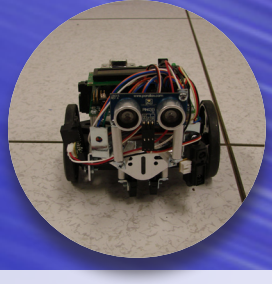
Day One
LINE BY LINE



Beeper.java

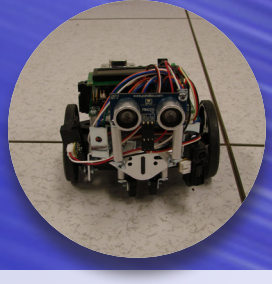
Source And Description:

```
import com.ridgesoft.io.Display; //Import Display library
import com.ridgesoft.intellibrain.IntelliBrain; //Import Intellibrain library
import com.ridgesoft.robotics.Servo; //Import Servo library
import com.ridgesoft.io.Speaker; //Import the Speaker library

public class Beeper          //Declare the Beeper Class
{                            //Open the Beeper Class
  public static final int QUARTER_NOTE = 400;          //These are constants needed to play
   public static final int HALF_NOTE = QUARTER_NOTE * 2;   //the music at the end of this
   public static final int C = 262;                        //program. They were copied
   public static final int C_SHARP = 277;                  //from the RidgeSoft Demo.
   public static final int D = 294;                        //See file for copyright info.
   public static final int D_SHARP = 311;
   public static final int E = 330;
   public static final int F = 349;
   public static final int F_SHARP = 370;
   public static final int G = 392;
   public static final int G_SHARP = 415;
   public static final int A = 440;
   public static final int A_SHARP = 466;

   public static void main(String args[]) //Declare the main method
    {try{                                 //Open the main method and try block, for sleep()

        Display display = IntelliBrain.getLcdDisplay();     //Create a display object
        display.print(0, "Beeper");                         //Output to first line
        display.print(1, "Starting....");                   //Output to second line

        Servo leftWheel = IntelliBrain.getServo(1);         //Create a servo for the leftWheel
        Servo rightWheel = IntelliBrain.getServo(2);        //Create a servo for the rightWheel

        Speaker buzzer = IntelliBrain.getBuzzer();          //Create a speaker object
```



Beeper.java - Continued

```
display.print(1, “Driving - D”);        //Display text to second line of display

leftWheel.setPosition(100);             //left wheel full speed forward
rightWheel.setPosition(0);              //right wheel full speed forward

Thread.sleep(5000);                     //sleep for 5 seconds

leftWheel.off();                        //left wheel off
rightWheel.off();                       //right wheel off

Thread.sleep(500);                      //sleep for half a second

display.print(1, “Driving - R”);        //display text to second line of display

leftWheel.setPosition(0);               //left wheel full speed reverse
rightWheel.setPosition(100);            //right wheel full speed reverse

buzzer.beep();                          //Buzz the buzzer
Thread.sleep(1000);                     //Sleep 1 second
buzzer.beep();                          //Repeat.....
Thread.sleep(1000);
buzzer.beep();
Thread.sleep(1000);
buzzer.beep();
Thread.sleep(1000);
buzzer.beep();
Thread.sleep(1000);                     //.... End Repeat

leftWheel.off();                        //Turn off left wheel
rightWheel.off();                       //Turn off right wheel

display.print(1, “Playing.....”);       //Display text to first line of display
```

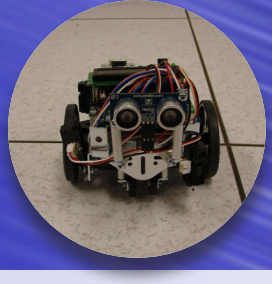


Beeper.java - Continued

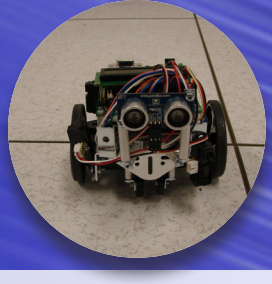
```
        buzzer.play(E, QUARTER_NOTE);          //Buzzer now plays song using the
        buzzer.play(E, QUARTER_NOTE);          //constants which were provided earlier.
        buzzer.play(F, QUARTER_NOTE);          //This was provided from a RidgeSoft file
        buzzer.play(G, QUARTER_NOTE);          //see file for copyright info.
        buzzer.play(G, QUARTER_NOTE);
        buzzer.play(F, QUARTER_NOTE);
        buzzer.play(E, QUARTER_NOTE);
        buzzer.play(D, QUARTER_NOTE);
        buzzer.play(C, QUARTER_NOTE);
        buzzer.play(C, QUARTER_NOTE);
        buzzer.play(D, QUARTER_NOTE);
        buzzer.play(E, QUARTER_NOTE);
        buzzer.play(E, HALF_NOTE);
        buzzer.play(D, HALF_NOTE);
        buzzer.play(E, QUARTER_NOTE);
        buzzer.play(E, QUARTER_NOTE);
        buzzer.play(F, QUARTER_NOTE);
        buzzer.play(G, QUARTER_NOTE);
        buzzer.play(G, QUARTER_NOTE);
        buzzer.play(F, QUARTER_NOTE);
        buzzer.play(E, QUARTER_NOTE);
        buzzer.play(D, QUARTER_NOTE);
        buzzer.play(C, QUARTER_NOTE);
        buzzer.play(C, QUARTER_NOTE);
        buzzer.play(D, QUARTER_NOTE);
        buzzer.play(E, QUARTER_NOTE);
        buzzer.play(D, HALF_NOTE);
        buzzer.play(C, HALF_NOTE);

    }catch(Throwable t){ t.printStackTrace();}}      //Close the main method and try block

}           //Close the Beeper Class
```

Day One

NOTES



Day Two
GOALS



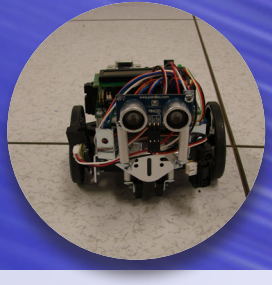
Key Concepts

- Writing Methods
- Loops And Conditions
- Turning the Robot

Goals of the Day

- Write Methods for common functions
- Successfully use loops
- Turn the robot right and left
- Calibrate the Robot's turns

Day Two
PREPARE



General

Before Day Two you will need to meet with your team and discuss how things operated on the previous day and make any necessary changes. Day Two is all about making it easier to move the robot. It is a good idea to review the keynote slides and outline ahead of time as well. You will need to be familiar with all the programs, demos, and examples ahead of time.

Demos

Day Two contains two demos. The demos can be found on the Resource Disc with the path of **/Demos/DayTwo** The first demo shows how to create a method which will drive the robot forwards. You can load this demo into RoboJDE and then complete the empty method and add the statements to the main method. The second demo shows how to use loops, you will probably want to open this demo as a solution and explain the code, rather than attempt to write it during the session . You can find solutions and explanations for both demo programs in the **line by line** section following Day Two.

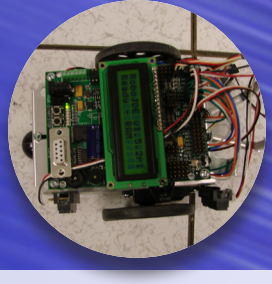
Programs

Students will need the **DayTwo** folder found in the **StudentFiles** section of the Resource Disc. This folder contains files related to the programs:

Turns- breaks common functions into methods, teaches how to turn
Repeat - using loops the robot drives in a square then straight, then a square,....
CustomDrive - students create their own custom driving program

You need to review these programs ahead of time, so you can assist students. As usual you can find a detailed description in the **line by line** guide. Additionally you can review the solution files for insight into the programs.

Day Two
BEGIN



Getting Started

Get started by welcoming everyone back to the programming/lab portion of the camp. Take some time to get any new students setup with computers, logins, student file packs and any other material they will need. Once everyone is ready, begin the session by following the Day Two Keynote outline.

Distribution of Materials

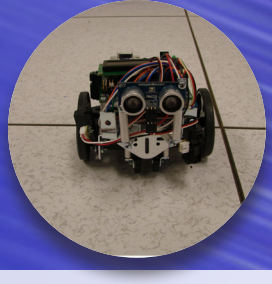
Robots will be needed for Day Two activities, so if you do not choose to distribute them ahead of time, you will need to do so during the presentation. Also, student files will need to be loaded onto student computers.

Student Files

Students will need the **DayTwo** folder found in the **StudentFiles** section of the Resource Disc. This folder contains files related to the programs:

Turns, Repeat, and CustomDrive

Day Two
KEYNOTE



Keynote

The Keynote for Day Two is found on the Resource Disc at the path **/Presentations/Day-Two/**. Today's keynote contains 33 Slides and will take an estimated hour to hour and a half to complete. Note that examples are embedded in the keynote.

Keynote Outline

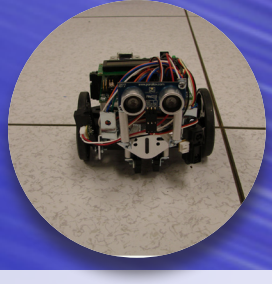
Slide One - Title
Slide Two - Review
Review with the students about what they learned yesterday. You don't need to go into a lot of detail but be sure to remind everyone of the basic concepts. Following this slide is a slide for each bullet.

Slide Three - RoboJDE
Yesterday we learned how to use RoboJDE so we can write, compile, and download programs to our robots. Today we will continue using RoboJDE for these tasks. If you're just joining the camp or don't entirely remember how to use RoboJDE it's alright, as we will have another demo program in a moment which will help refresh your memory.

Slide Four - LCD Display
Yesterday we also learned the three basic steps for using most inputs and outputs associated with our robots. Here are the specific steps for the LCD Display. From this point forward most of your templates will already have the first two steps completed for you. You will still need to keep in mind how to use the print method.

Slide Five - Thread.sleep()
We also realized the importance of having the ability to make the program sleep for an certain amount of time. This allows us to do things like drive forward for an amount of seconds or display text for a certain time period. Today you will see how important sleep times are when making the robot turn correctly.

Day Two
KEYNOTE



Keynote Outline (cnt'd)

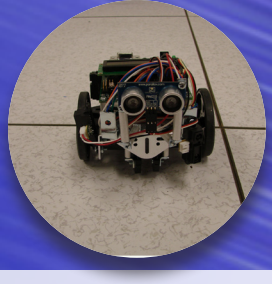
Slide Six - Using Servos
Servo motors move the robot's wheels. Like the LCD display in most templates from today until the end of camp the first two steps in this process will be provided for you. You will still need to remember the details about setting positions and using the off function. Today you'll build your own custom methods to interact with the Servos and make your life as a programmer easier.

Slide Seven - Producing a Beep
We won't be using the beep function in any of our template programs today, but you may want to include it in your Custom Drive program which you'll be developing at the end of this session.

Slide Eight - API
Remember the API for the Intellibrain libraries can be very helpful when you can't quite remember how to use something. You'll see references to the API in your template files. When you get stuck always check to see if the API can help you out.

Slide Nine - Creating Methods
Today we will be making our lives as programmers much easier. We are going to create some methods which will help us perform common operations very quickly. A method can be thought of as a little set of instructions.

Slide Ten - Methods
A program can have as many methods as you want. Any program that runs directly will have a main method. Java knows to execute whatever is in the main method line by line until it reaches the end. We can also add more methods besides the main method, but how will Java know to use them? We must tell Java through the main method. By making calls to other methods from the main method we can redirect Java to use code we have supplied in other methods. If this doesn't make sense yet, don't worry it will once you see a Demo.
There are two types of methods in Java: return and void. Return methods always return a value to whatever called them. Void methods never return a value, but may set some values or print out some data. We will mainly use void methods in our programs.



Keynote Outline (cnt'd)

Slide Eleven - Method Example
Methods, for our purposes, always start with the keyword public followed by static, then the return type: in this case void, and then the method name. After the method name in parentheses are the parameters the method accepts. Parameters are bits of information we tell the method about so it can do its job, they can be any data type. Following this header are open brackets and the method body.

You can think of a method like this:

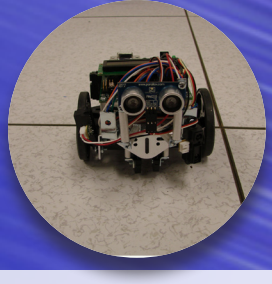
Suppose you (the main method) are trying to solve a riddle, you have some clues (parameters) but you don't know how to use them. Now your friend (the other method) would know how to put these clues together but doesn't have access to the clues. So you email your friend and ask (call) him or her to help you, you attach the clues you have (sending a parameter) to your email so your friend can use them. Once your friend is finished he or she may either email back the answer to you (return method) or post the answer somewhere where you can view it (void method).

This example takes two integers and prints out the custom text with the computation. Now it's time for a demo.

Slide Twelve - Demo
For this demo you will need the demo file for DemoMethod , it can be found in the **Demos** folder in the **DayTwo** Folder. Your task is to complete the drive method. The solution can be found in the **line by line** guide at the end of this day's guide.

The program basically consolidates two lines of code into a method called drive. You can then call this method anytime you wish to drive forward. Inform students their hands on example will also include writing this method so they may want to pay close attention.

Day Two
KEYNOTE



Keynote Outline (cnt'd)

Slide Thirteen - How to Turn?
Engage the students in a group conversation about how they think we could tell the robot to turn right or left. Inform the students there are no methods to do this in the API, we will be writing our own methods to do this in the hands on exercise.

The correct answer is to make the wheels spin in opposite directions, then make the thread sleep for just enough time to complete the turn. The recommended amount of sleep time for a turn is 625 milliseconds.

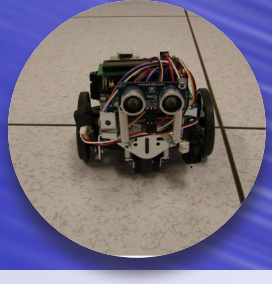
NOTE: This time varies significantly depending on battery life and floor surface. Inform students of this and let them know they will have to spend some time calibrating their code to make the turns work out.

Slide Fourteen - Hand On Exercise
You will now need to instruct students to begin the first hands on exercise, **Turns.java**. The templates for the students are available in the **DayTwo** folder in the **StudentFiles** folder on the Resource Disc. You should already have an understanding of how this program works, but if you run into trouble you can always consult the **line by line** guide, found at the end of the Day Two section of this guide. Additionally the solutions are provided on the Resource Disc with a path of **/Solutions/DayTwo**.

Slide Fifteen - Loops & Condtions
Loops and conditions help us to make our programs more efficient and easier to read. They also reduce the amount of work programmers must do.

Slide Sixteen - Useful Comparison Info
Review with the students each comparison type. The modulus operator is included because it will be used in a hands on exercise. Explain the function of the modulus operator, as most students are unfamiliar with it.

Day Two
KEYNOTE



Keynote Outline (cnt'd)

Slide Seventeen - A Quicker Way
Review with students how statements in java can be simplified, answer any questions they may have.

Slide Eighteen - What is an If Statement
An if statement is a statement which checks the condition of something and then acts appropriately. If the statement provided to the the if statement is true then the block of code the if statement has is executed.

If statements can also have else statements which will be executed if the if statement evaluates to false.

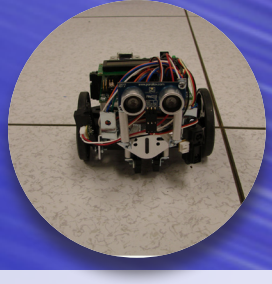
Slide Nineteen - if Statement
Any type of conditional statement can be placed within the parentheses, even multiple statements linked with &&'s or ||'s. If the statement is true the block is executed.

Slide Twenty - If Statement Example
Go over this code with the students, remind them that == is not the same as =.

Slide Twenty-One - What is a Loop
A loop is a controlled way of executing a block of code repetitively. There is always a condition which must be true for the loop to keep running. The condition is like the if statement's condition in that it can be very simple or very complex. There are two main types of loops which we will look at for our purposes.

Slide Twenty-Two - while Loop
Go over this code with the students. Explain the structure of the while loop.

Slide Twenty-Three - While Loop Example
Go over this code with students, taking it one iteration at a time. If necessary use a white board to keep track of each iteration and the output. Explain to students how the count variable was initialized outside the loop.



Keynote Outline (cnt'd)

Slide Twenty-Four - While Loop Complex Example
Go over this more complex code with the students. Note the use of a complex condition linked with && operators. Also explain to students how we can embed if statements within loops, loops within loops, and loops within if statements. Tell students to pay close attention to where count is being incremented in the code block.

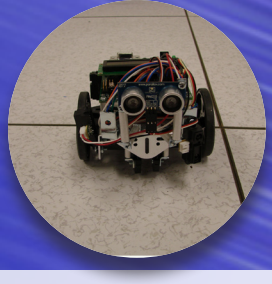
Slide Twenty-Five - While Loop Complex Example
This code is the same as the previous example, but with the count being incremented in a different spot. Compare the two pieces of code. Stress to students how this affects the output. Explain that even the best programmers experience errors at times because something as simple as moving this one statement can change the whole program.

Slide Twenty-Six - What happens
What happens if we run a piece of code like this? The condition never changes, so when does the loop finish?

Slide Twenty-Seven - Infinite Loops
Infinite loops are created when the exit condition of the loop is never met. Most of the time infinite loops are created by accident or by programmer error. We will use infinite loops later to keep our robots moving until we press the stop button.

Slide Twenty - Eight - for Loop
A for loop is another way of looping through some code. The for loop is exceptional in that it allows us to initialize a variable for the loop, set a condition for the loop, and setup the increment which will occur after each iteration all in one statement.

Slide Twenty - Nine - for Loop Example
Explain the code to the students. Once again use a white board to track each iteration and the output. Ask students what would happen if we attempted to output the variable i after the loop brackets, later in the program. Explain that unlike the while loop, the variable i will only work within the for loop, since it was created there.



Keynote Outline (cnt'd)

Slide Thirty - Demo
For this demo you will need the demo file for **DemoLoop** , it can be found in the **Demos** folder in the **DayTwo** Folder. Your task is to complete the drive method. The solution can be found in the **line by line** guide at the end of this day's guide.

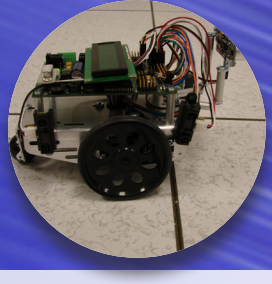
The program basically tells the robot to turn right four times using a for loop, then drive forward, then spin in a circle in an infinite loop.

Slide Thirty-One Hands On Exercise
You will now need to instruct students to begin the second hands on exercise, **Repeat.java**. The templates for the students are available in the **DayTwo** folder in the **StudentFiles** folder on the Resource Disc. You should already have an understanding of how this program works, but if you run into trouble you can always consult the **line by line** guide, found at the end of the Day Two section of this guide. Additionally the solutions are provided on the Resource Disc with a path of **/Solutions/DayTwo**.

Slide Thirty-Two Hands On Exercise
You will now need to instruct students to begin the third hands on exercise, **CustomDrive.java**. The templates for the students are available in the **DayTwo** folder in the **StudentFiles** folder on the Resource Disc. You should already have an understanding of how this program works, but if you run into trouble you can always consult the **line by line** guide, found at the end of the Day Two section of this guide. Additionally the solutions are provided on the Resource Disc with a path of **/Solutions/DayTwo**.

Slide Thirty-Three - End of Day Two

Day Two
WRAP UP



Wrap Up

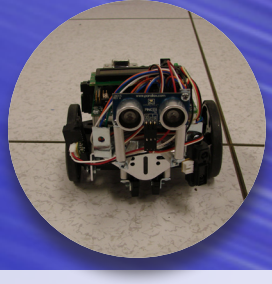
Once students have finished completing the exercises take questions and work to clear up any difficulties which were experienced. Tomorrow students will begin using sensors to read information about the environment around them. Inspire the students to return tomorrow by letting them know: tomorrow they will teach their robots to understand and interact with the world around them.

Line by Line - Day Two

MethodDemo.java

Source And Description:

```
import com.ridgesoft.intellibrain.IntelliBrain; //Import Intellibran libraries
import com.ridgesoft.robotics.Servo; //Import Servo libraries

public class DemoMethod        //Open the DemoMethod class
{
   public static void main(String args[]) //Decalare and open the main method and try block
        {try{
                Servo leftWheel = IntelliBrain.getServo(1); //Create a left wheel servo
                Servo rightWheel = IntelliBrain.getServo(2); // Create a right wheel servo
                drive(leftWheel,rightWheel);                    //Use our drive method to go forward
                Thread.sleep(3000);                             //Sleep for 3 seconds while driving

 }catch(Throwable t){ t.printStackTrace();}}                    //Close the main method and try

   public static void drive(Servo l, Servo r)                   //Create our drive method
   {
        l.setPosition(100);                                 //set servos passed to full power forward
        r.setPosition(0);
    }

}
```



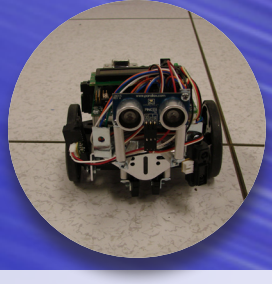
DemoLoop.java

Source And Description:

```
import com.ridgesoft.intellibrain.IntelliBrain; //Import Intellibran libraries
import com.ridgesoft.robotics.Servo; //Import Servo libraries

public class DemoLoop //Declare and open the DemoLoop Class
{
   public static void main(String args[])  //Declare and open main method and try block
        {try{

        Servo leftWheel = IntelliBrain.getServo(1);          //Create a left wheel servo
        Servo rightWheel = IntelliBrain.getServo(2);         //Create a right wheel servo

        for(int i=0;i<4;i++)                                 //Do this 4 times
        {
                turnRight(leftWheel,rightWheel);             //Turn right
                stopBot(leftWheel,rightWheel);               //Stop
                Thread.sleep(1000);                          //Pause for 1 second
        }

        drive(leftWheel,rightWheel);                         //Drive Forward
        Thread.sleep(3000);                                  //For 3 Seconds

        while(true)                                          //Do this until stop is pressed
        {
                turnRight(leftWheel,rightWheel);             //Turn Right
        }
 }catch(Throwable t){ t.printStackTrace();}}          //Close main method and try block

        //Usual motion methods included here (Not Shown on this Document)
}       //Close Class
```

Day Two

LINE BY LINE



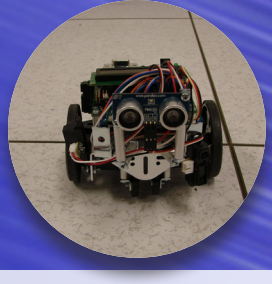
Turns.java (Motion Methods not shown in this document)

Source And Description:

```
import com.ridgesoft.io.Display; //Import Display Library
import com.ridgesoft.intellibrain.IntelliBrain; //Import IntelliBrain Libaray
import com.ridgesoft.robotics.Servo; //Import Servo Libarary

public class Turns                          //Declare and open the class
{
   public static void main(String args[])       //Declare and open main method and try block
   {try{
         Display display = IntelliBrain.getLcdDisplay();      //Create Display object
         Servo leftWheel = IntelliBrain.getServo(1);          //Create left wheel servo
         Servo rightWheel = IntelliBrain.getServo(2);         //Create right wheel servo

         display.print(0,"Turns.java");                       //display some text to first line
         display.print(1, "Driving - D");                     //display some text to second line

         drive(leftWheel,rightWheel);                         //Drive Forward
         Thread.sleep(3000);                                  //For 3 Seconds
         stopBot(leftWheel,rightWheel);                       //Stop
         Thread.sleep(500);                                   //For half of a second
         turnRight(leftWheel,rightWheel);                     //Turn Right
         stopBot(leftWheel,rightWheel);                       //Stop
         Thread.sleep(500);                                   //For half of a second
         drive(leftWheel,rightWheel);                         //Drive Forward
         Thread.sleep(2000);                                  //For 2 Seconds
         stopBot(leftWheel,rightWheel);                       //Stop
         Thread.sleep(500);                                   //For Half of a second
         turnLeft(leftWheel,rightWheel);                      //Turn left
         stopBot(leftWheel,rightWheel);                       //Stop
         Thread.sleep(500);                                   //For half of a second
         backup(leftWheel,rightWheel);                        //Back up
         Thread.sleep(5000);                                  //For 5 seconds
         stopBot(leftWheel,rightWheel);                       //Stop
   }catch(Throwable t){ t.printStackTrace();}}        //Close main method and try block
} //Close class (NOTE: Motion Methods not shown in this document
```

Day Two

LINE BY LINE



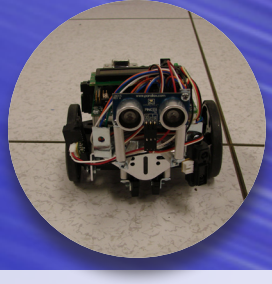
Repeat.java (Motion Methods not shown in this document)

Source And Description:

```
import com.ridgesoft.io.Display; //Import Display Library
import com.ridgesoft.intellibrain.IntelliBrain; //Import IntelliBrain Libaray
import com.ridgesoft.robotics.Servo; //Import Servo Libarary

public class Repeat          //Declare and open the class
{
   public static void main(String args[]) //Declare and open main method and try block
   {try{
        Display display = IntelliBrain.getLcdDisplay();      //Create display object
        Servo leftWheel = IntelliBrain.getServo(1);          //Create left wheel servo
        Servo rightWheel = IntelliBrain.getServo(2);         //Create right wheel servo

        display.print(0,"Turns--");                          //print to first line
        int count=0;                                         //declare count variable set to 0

        while(true) {                                        //Infinite loop start
        if((count%2)==0)                                     //If count is even do the following
        {
                for(int i=0;i<4;i++)                         //Do this 4 times
                {
                drive(leftWheel,rightWheel);                 //Drive forward
                Thread.sleep(1500);                          //For 1.5 seconds
                stopBot(leftWheel,rightWheel);               //Stop
                Thread.sleep(200);                           //for .2 seconds
                turnRight(leftWheel,rightWheel);             //turn right
                stopBot(leftWheel,rightWheel);               //stop
                Thread.sleep(200);                           //pause for .2 seconds
                }
        count++;                                             //Increment the count by 1
        }
        else                                                 //If count is odd
        {
        drive(leftWheel,rightWheel);                         //Drive forward
        Thread.sleep(4000);                                  //for 4 seconds
```

Day Two

LINE BY LINE



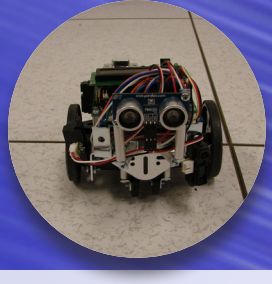
Repeat.java (Motion Methods not shown in this document)

Source And Description:

```
        stopBot(leftWheel,rightWheel);                  //Stop
        Thread.sleep(200);                              //Pause for .2 seconds
        count++;                                        //Increment count
        }
}                                                       //End Infinite While loop
}catch(Throwable t){ t.printStackTrace();}}             //Close main method and try block
//Motion Methods not shown in this docuement
} //Close the Class
```

Day Two

LINE BY LINE



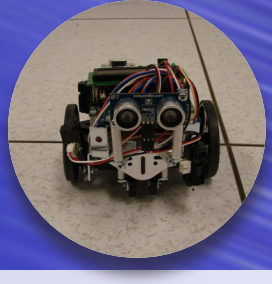
CustomDrive.java (Motion Methods not shown in this document)

Source And Description:

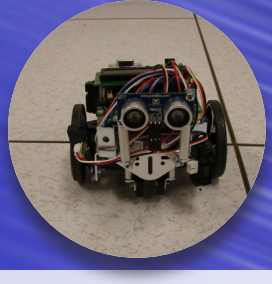
```
import com.ridgesoft.io.Display; //Import Display Library
import com.ridgesoft.intellibrain.IntelliBrain; //Import IntelliBrain Libaray
import com.ridgesoft.robotics.Servo; //Import Servo Libarary

public class CustomDrive //Declare and Open the class
{
 public static void main(String args[]) // Declare and open main method and try block
 {try{

        Display display = IntelliBrain.getLcdDisplay();      //Create display object
        Servo leftWheel = IntelliBrain.getServo(1);          //Create left wheel servo
        Servo rightWheel = IntelliBrain.getServo(2);         //Create right wheel servo

        //Fill in your custom navigation code below

   }catch(Throwable t){ t.printStackTrace();}} //Close main method and try block
//Note Motion methods not shown in this document
} //Close class
```

Day Two

NOTES



Day Three
GOALS



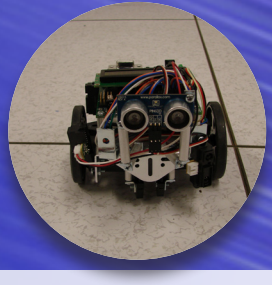
Key Concepts

- Mastering IF Conditions
- Mastering Loops
- Using Sensors

Goals of the Day

- Use a Sonar Sensor
- Use a IR Range Sensor
- Use a Photo Reflector Sensor
- Make Practical Use of Loops and Conditions

Day Three
PREPARE



General

Before Day Three you will need to meet with your team and discuss how things operated on the previous day and make any necessary changes. Day Three is primarily about reading input through different sensors and then making a decision as to what to do about that input. You will need to double check that robots are properly configured to work with examples. You can find the model bot specifications in the **Pre-Camp** section. Working with sensors and the Intellibrain can become frustrating, particularly when working with the IR Range Sensors. Remind students that it's all about having fun and discovering what computer science can do. Perfection is rarely achievable with the Intellibrain. It is a good idea to review the keynote slides and outline ahead of time as well. You will need to be familiar with all the programs, demos, and examples ahead of time.

Demos

Day Three does not contain any demos. Instead you will need to work actively with students in perfecting their sensor based programs.

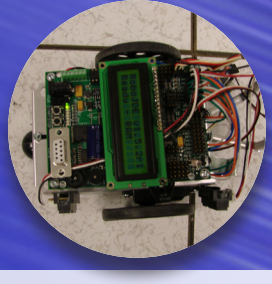
Programs

Students will need the **DayThree** folder found in the **StudentFiles** section of the Resource Disc. This folder contains files related to the programs:

SonarSensor- detects objects and avoids them by turning using Sonar
DoorFinder - drives along a wall finds a door opening and beeps for duration of opening, using IR Range Sensors
Stop - robot drives until finds a dark color, then stops, using Photo Reflectors.The Stop program ***requires you to have some sort of dark poster board*** or surface for the robot to drive onto.

You need to review these programs ahead of time, so you can assist students. As usual you can find a detailed description in the **line by line** guide. Additionally you can review the solution files for insight into the programs.

Day Three
BEGIN



Getting Started

Get started by welcoming everyone back to the programming/lab portion of the camp. Take some time to get any new students setup with computers, logins, student file packs and any other material they will need. Inform students that today will be especially exciting and full of new concepts. Once everyone is ready begin the session by following the Day Three Keynote outline.

Distribution of Materials

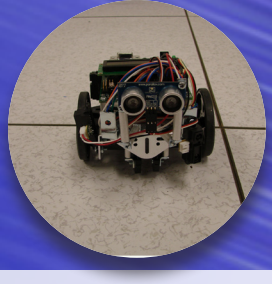
Robots will be needed for day three activities, so if you do not choose to distribute them ahead of time, you will need to do so during the presentation. Remember you need to check the robot configurations ahead of time, or allow students time to do so. Also student files will need to be loaded onto student computers.

Student Files

Students will need the **DayThree** folder found in the **StudentFiles** section of the Resource Disc. This folder contains files related to the programs:

SonarSensor, DoorFinder, and Stop

Day Three
KEYNOTE



Keynote

The Keynote for Day Three is found on the Resource Disc at the path **/Presentations/DayThree/**. Today's keynote contains 20 Slides and will take an estimated hour to hour and a half to complete. Note that examples are embedded in the keynote.

Keynote Outline

Slide One - Title

Slide Two - Review

Yesterday we made things easier for ourselves by breaking common functions down into methods, learning to use loops and experimenting with conditions. Today we will take advantage of yesterday's work as we begin to use sensors to read input about the world around us.

Review with the students about what they learned yesterday. You don't need to go into a lot of detail but be sure to remind everyone of the basic concepts.

Slide Three - Sensors

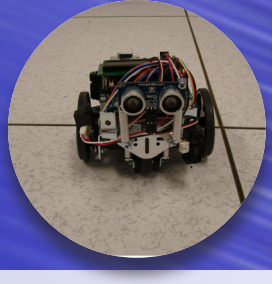
Sensors allow us to give our robot some feedback about the things around it. Up until this point we have only been able to give our robot hard coded static instructions. Today our robot will become dynamic and begin to make decisions for itself. We will set up the rules and the robot will explore its world while following them.

Engage the students in a discussion concerning the usefulness of sensors. Ask students what do they think could be accomplished by using sensors?

Slide Four - Types of Sensors

The Intellibrain has three main sensors available to us. Two sensors relate to sensing the range of an object or wall and the other relates to sensing the contrast between colors.

Day Three
KEYNOTE



Keynote Outline (cnt'd)

Slide Five - Sonar Sensors
The Sonar sensors are mounted on the front of the Intellibrain, as shown here. In order to use the sonar sensors there are a few things we must learn about them.

Slide Six - How Sonar Works
Sonar sensors work by sending out a wave of sound and then measuring how long it takes for that wave to echo back. The Intellibrain sonar equipment can measure anywhere from 3 inches to 20 feet. When working with the sonar sensors always be sure your program sleeps at least 100 milliseconds between pings, otherwise it may get incorrect readings. Also be sure to work in an area where other sonar devices are not in use, as the waves could interfere with each other.

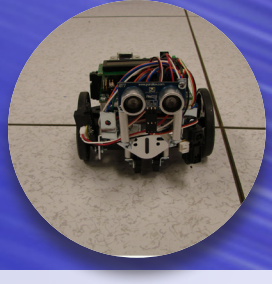
Slide Seven - Using Sonar
To use the sonar sensors you must always import these two statements, and create an object using a statement like this. When you want to get a reading from the sensor you first call the ping method, sleep for 100 milliseconds, then call the getDistanceInches() method and store it into a float variable.

Slide Eight - Sample Sonar Code
Review the code and take any questions the students may have.

We will now move to a hands on exercise where we will build a program which helps our robot avoid hitting obstacles.

Slide Nine - Hands On Exercise
You will now need to instruct students to begin the first hands on exercise, **SonarSensor.java**. The templates for the students are available in the **DayThree** folder in the **StudentFiles** folder on the Resource Disc. You should already have an understanding of how this program works, but if you run into trouble you can always consult the **line by line** guide, found at the end of the Day Three section of this guide. Additionally the solutions are provided on the Resource Disc with a path of **/Solutions/DayThree.**

Day Three
KEYNOTE



Keynote Outline (cnt'd)

Slide Ten - Infrared Range Sensors
The Infrared Range sensors are mounted on the side of your Intellibrain, as shown here. In order to use the IR range sensors there are a few things we must learn about them.

Slide Eleven - How IR Range Works
The IR range sensor works by using an emitter and a receiver. By emitting light out of one and receiving it into the other, the range sensor can calculate, based on the angle of return light how far away an object or wall is. The Intellibrain IR range sensors can measure between 4 and 31 inches. Unfortunately, during testing the IR range sensors have proven the least reliable. That is why we have two mounted on our robot, it increases the likelihood of a more accurate readings.

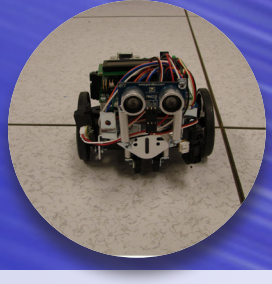
Slide Twelve - Using IR Range
In order to use the IR range sensors you will need to import two libraries. You can then create the RangeFinder objects. The rear sensor is usually connected to Analog Input 1, while the front is connected to input 2. You should check your robot to make sure this holds true. To use the sensors you must ping them, then use the getDistanceInches method to receive their value. We pass null as a second parameter to indicate the power to sensor should always be on.

Slide Thirteen - Sample IR Code
Review the code and take any questions the students may have.

We will now begin a hands on exercise which will use IR range sensors to locate doorways along a wall. The robot will beep when it crosses a doorway and continue to beep until it finds the next wall.

Slide Fourteen - Hands On Exercise
You will now need to instruct students to begin the second hands on exercise, **DoorFinder.java**. The templates for the students are available in the **DayThree** folder in the **StudentFiles** folder on the Resource Disc. You should already have an understanding of how this program works, but if you run into trouble you can always consult the **line by line** guide, found at the end of the Day Three section of this guide. Additionally the solutions are provided on the Resource Disc with a path of **/Solutions/DayThree.**

Day Three
KEYNOTE



Keynote Outline (cnt'd)

Slide Fifteen - Photo Reflector Sensor
The photo reflector sensors are mounted on the bottom of the Intellibrain, as shown here. In order to use the photo reflector sensors there are a few things we must learn about them.

Slide Sixteen - How Photo Reflectors Work
Photo reflectors work by emitting light and calculating the way it bounces back. Instead of measuring distance photo reflectors measure intensity. They can tell the difference between different shades of color.

Slide Seventeen - Using Photo Reflectors
First you must import the AnalogInput library, and then create an object for the photo reflector. The photo reflectors should be attached to inputs 6 and 7. There are two other photo reflectors mounted behind the wheels of the Intellibrain, we will not be using these so be sure you are reading from the right sensors. In order to obtain a reading from the sensor you simply call the sample method, which returns an int type value.

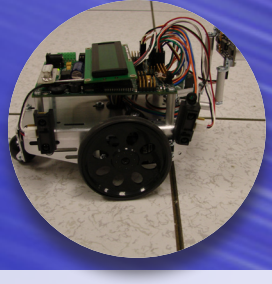
Side Eighteen - Sample Photo Reflector Code
Review the code and take any questions the students may have.

Now we will create a program which will tell our robot to stop when it drives onto a darker area.

Slide Nineteen - Hands On Exercise
You will now need to instruct students to begin the third hands on exercise, **Stop.java**. The templates for the students are available in the **DayThree** folder in the **StudentFiles** folder on the Resource Disc. You should already have an understanding of how this program works, but if you run into trouble you can always consult the **line by line** guide, found at the end of the Day Three section of this guide. Additionally the solutions are provided on the Resource Disc with a path of **/Solutions/DayThree**.

Slide Twenty - End Day Three

Day Three
WRAP UP



Wrap Up

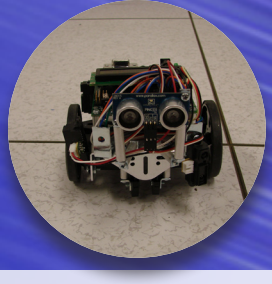
Once students have finished completing the exercises take questions and work to clear up any difficulties which were experienced. Tomorrow students will use what they've learned this week to create behaviors for their robots. Inspire the students to return tomorrow by letting them know tomorrow they will explore how to give their robot behaviors.

Line by Line - Day Three

SonarSensor.java (Motion Methods not shown in this document)

Source And Description:

```
import com.ridgesoft.robotics.SonarRangeFinder; // Import the SonarRange finder library
import com.ridgesoft.robotics.sensors.ParallaxPing; // Import the Parallax  library
import com.ridgesoft.intellibrain.IntelliBrain;// Import the IntelliBrain finder library
import com.ridgesoft.io.Display; // Import the Display finder library
import com.ridgesoft.robotics.Servo; // Import the Servo finder library


public class SonarSensor                        //Open the class
{
   public static void main(String args[])   //Declare and Open the main method and try block
        {try{

        Display display = IntelliBrain.getLcdDisplay();       //Create a display object
        Servo leftWheel = IntelliBrain.getServo(1);          //Create a left wheel servo
        Servo rightWheel = IntelliBrain.getServo(2);         //Create a right wheel servo
                                                  //BELOW: create sonar range finder object
        SonarRangeFinder ahead = new ParallaxPing(IntelliBrain.getDigitalIO(4));

        ahead.ping();                 //Ping the sonar range finder
        Thread.sleep(100);            //Wait .1 seconds for response

        float sonarDistance=ahead.getDistanceInches();    //Store results of sonarDistance
```

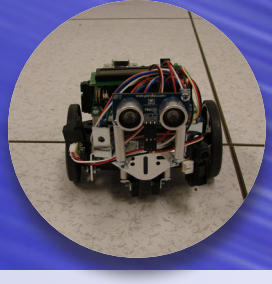


SonarSensor.java (Cnt'd) (Motion Methods not shown in this document)

```
            while(true)                          //Infinite Loop
            {
            drive(leftWheel,rightWheel);         //Drive Forward
            ahead.ping();                        //Check ahead
            Thread.sleep(100);                   //Wait .1 seconds for response
            sonarDistance=ahead.getDistanceInches();        //store distance
            display.print(1, "SR : "+sonarDistance);        //display distance

                  if(sonarDistance<4.5f && sonarDistance>0) //if too close or not close at all
                  {
                        stopBot(leftWheel,rightWheel);      //Stop
                        Thread.sleep(300);                  //For .3 seconds
                        backup(leftWheel,rightWheel);       //Back up
                        Thread.sleep(800);                  // For 8 Seconds
                        turnRight(leftWheel,rightWheel);    // Turn Right
                        stopBot(leftWheel,rightWheel);      //Stop
                        Thread.sleep(500);                  //For .5 Seconds
                  }

            }                                          //End Infinite Loop

   }catch(Throwable t){ t.printStackTrace();}}         //Close the main method and try block
//Motion methods not shown in this document
} //Close the class
```

Day Three

LINE BY LINE



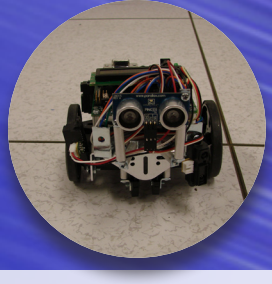
DoorFinder.java (Motion Methods not shown in this document)

Source And Description:

```
import com.ridgesoft.io.Display; // Import the Display finder library
import com.ridgesoft.intellibrain.IntelliBrain;// Import the Intellibrain library
import com.ridgesoft.robotics.Servo; // Import the Servo library
import com.ridgesoft.robotics.RangeFinder; // Import the Range finder library
import com.ridgesoft.robotics.sensors.SharpGP2D12; // Import the Sharp library
import com.ridgesoft.io.Speaker; // Import the Speaker library

public class DoorFinder                    //Open the class
{
   public static void main(String args[])     //Declare and Open the main method and try block
        {try{

        Display display = IntelliBrain.getLcdDisplay();       //Create a Display object
        Servo leftWheel = IntelliBrain.getServo(1);           //Create a left wheel servo
        Servo rightWheel = IntelliBrain.getServo(2);          //Create a right wheel servo
        //BELOW: Create two range finders
        RangeFinder rearFinder = new SharpGP2D12(IntelliBrain.getAnalogInput(1), null);
        RangeFinder frontFinder = new SharpGP2D12(IntelliBrain.getAnalogInput(2), null);
        Speaker buzzer = IntelliBrain.getBuzzer();            //Create Speaker
        float rearDistance=0.0f;                              //Set up rear variable
        float frontDistance=0.0f;                             //Set up front varaible
        display.print(1, “Obtaining Distance”);               //Display text

        while(rearDistance<=0)                                //While invalid reading read
        {
                rearFinder.ping();                            //Ping the Range finder
                rearDistance=rearFinder.getDistanceInches();      //get the measurement
        }
        float maxRearDistance=rearDistance+7.5f;              //Add 7.5 to make max distance
        display.print(1, “RActual: “+rearDistance);           //Print text
        Thread.sleep(3000);                                   //Pause 3 seconds
        display.print(1, “RMAX: “+maxRearDistance);           //Print text
        Thread.sleep(2000);                                   //Pause 2 seconds
```

Day Three

LINE BY LINE

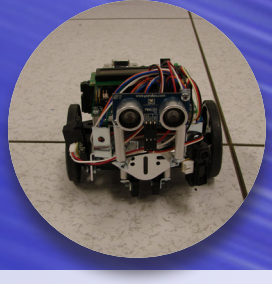


DoorFinder.java (Cont'd) (Motion Methods not shown in this document)

```
        display.print(1, "Obtaining Distance");          //Print Text
            while(frontDistance<=0)                      //While invalid reading read
            {
                frontFinder.ping();                      //Ping front range finder
                frontDistance=frontFinder.getDistanceInches();   //Capture measurement
            }
            float maxFrontDistance=rearDistance+7.5f;        //Add 7.5 to create max range

            while(true)                              //Start infinite loop
            {
            drive(leftWheel,rightWheel);             //Drive
            Thread.sleep(500);                       //Wait .5 seconds
            rearFinder.ping();                       //Ping rear distance
            frontFinder.ping();                      //Pint front distance
            frontDistance=frontFinder.getDistanceInches();   //Record front distance
            rearDistance=rearFinder.getDistanceInches();     //Record  rear distance
            display.print(0,"F: "+frontDistance);            //Print
            display.print(1,"R: "+rearDistance);             //Print

            //BELOW: While reading of both sensors is past max or invalid execute this
            while(((frontDistance>maxFrontDistance) && (rearDistance>maxRearDistance)) || (frontDistance<0 && rearDistance<0))
            {
            drive(leftWheel,rightWheel);             //Drive
            buzzer.beep();                           //Beep
            Thread.sleep(100);                       //Wait .5 seconds
            rearFinder.ping();                       //ping rear
            frontFinder.ping();                      //ping front
            frontDistance=frontFinder.getDistanceInches();   //Record front distance
            rearDistance=rearFinder.getDistanceInches();     //Record rear distance
            }                                        //End loop
            }                                        //End infinite loop
        }catch(Throwable t){ t.printStackTrace();}}          //Close main method and try block
        //Motion Methods not included in this document
} // Close the class
```

Day Three

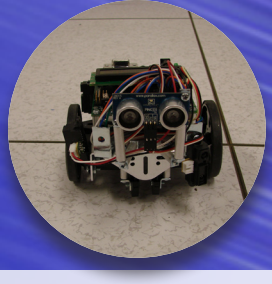
LINE BY LINE



Stop.java (Motion Methods not shown in this document)

Source And Description:

```
import com.ridgesoft.io.Display; //Import Display library
import com.ridgesoft.intellibrain.IntelliBrain;//Import Intellibrain library
import com.ridgesoft.robotics.AnalogInput; //Import Analog Input library
import com.ridgesoft.robotics.Servo; //Import Servo library
import com.ridgesoft.robotics.PushButton; //Import Push Button library
public class Stop                                   //Declare & Open class
{
   public static void main(String args[])       //Declare and open main method and try block
        {try{
        Display display = IntelliBrain.getLcdDisplay(); //Create Display object
        Servo leftWheel = IntelliBrain.getServo(1); //Create left wheel servo
        Servo rightWheel = IntelliBrain.getServo(2); //Create right wheel servo
        AnalogInput leftLightSensor = IntelliBrain.getAnalogInput(6); //Create photo reflector
        AnalogInput rightLightSensor = IntelliBrain.getAnalogInput(7);//Create photo reflector
        PushButton startButton = IntelliBrain.getStartButton(); //Create start button object


                display.print(0,"Place Robot on");      //Print Text
                display.print(1,"Dark Color");          //Print Text
                Thread.sleep(4000);                     //Pause 4 Seconds
                display.print(0,"THEN PRESS");          //Print Text
                display.print(1,"START");               //Print Text
                startButton.waitPressed();              //Pause until start is pressed
                //BELOW: set dark to average of two readers readings
                int dark= (leftLightSensor.sample()+rightLightSensor.sample())/2;
                display.print(0,"Dark Zone:");          //Print Text
                display.print(1,"PR: "+dark);           //Print Text
                Thread.sleep(3000);                     //Pause 3 seconds
                display.print(0,"Place Robot on");      //Print Text
                display.print(1,"Light Color");         //Print Text
                Thread.sleep(4000);                     //Pause 4 seconds
                display.print(0,"THEN PRESS");          //Print Text
                display.print(1,"START");               //Print Text
                startButton.waitPressed();              //Pause until start is pressed
                //BELOW: set current to average of two readers readings
                int current= (leftLightSensor.sample()+rightLightSensor.sample())/2;
```

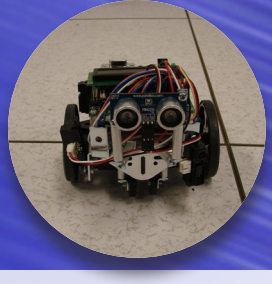


Stop.java (Cont'd) (Motion Methods not shown in this document)

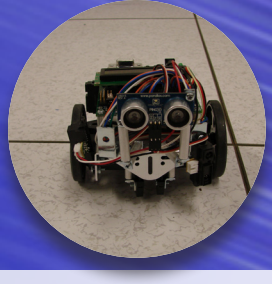
```
                while(current<(dark-70))        //While we are not on the dark spot
                {
                drive(leftWheel,rightWheel);                    //Drive Forward
                display.print(0,"Current Reading:");            //Print Text
                display.print(1,""+current);                    //Print Text
                //BELOW: set current to average of two readers readings
                current= (leftLightSensor.sample()+rightLightSensor.sample())/2;
                Thread.sleep(200);                              //Sleep .2 seconds
                }
  }catch(Throwable t){ t.printStackTrace();}}                   //Close main method and try block
        //Motion Methods not included in this document
} // Close the class
```

Day Three

NOTES



Day Four
GOALS



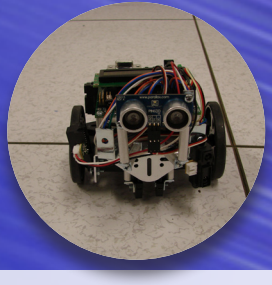
Key Concepts

- Introduction to Behaviors
- Creating Custom Class Types
- Polling
- Flow of Control

Goals of the Day

- Understand Behaviors
- Program Using Behaviors

Day Four
PREPARE



General

Day Four is an incredible leap from the previous three days. Today's topics require students to think differently about the flow of control in a program. You will want to take time to explain how methods hold control until they finish. Today's concepts will be difficult for students without previous Java experience. Spend the majority of your time today explaining the exercise programs It is a good idea to review the keynote slides and outline ahead of time as well. You will need to be familiar with all the programs, demos, and examples ahead of time.

Demos

Day Four does not contain any demos. Instead you will need to work actively with students in perfecting their sensor based programs.

Programs

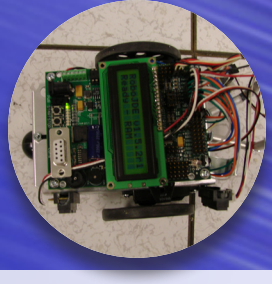
Students will need the **DayFour** folder found in the **StudentFiles** section of the Resource Disc. This folder contains files related to the programs:

ObjectCircle- uses the IR range sensors to circle around an object. Makes use of behaviors. ***You will need some kind of object to circle. Square objects work well, perhaps a box of copy paper.***
Avoid - uses behaviors to avoid obstacles by turning left when they come into sonar range.

You need to review these programs ahead of time, so you can assist students. As usual you can find a detailed description in the **line by line** guide. Additionally you can review the solution files for insight into the programs.

Day Four

BEGIN



Getting Started

Get started by welcoming everyone back to the programming/lab portion of the camp. Let students know upfront that today's activities will be fun but challenging. Acknowledge that these concepts may be hard to understand at first but that camp leaders will help them understand. Once everyone is ready, begin the session by following the Day Four Keynote outline.

Distribution of Materials

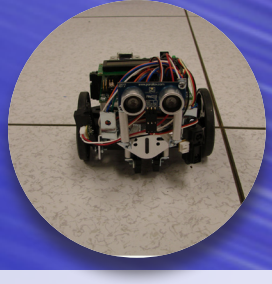
Robots will be needed for Day Four activities, so if you do not choose to distribute them ahead of time, you will need to do so during the presentation. Remember you need to check the robot configurations ahead of time, or allow students time to do so. Also student files will need to be loaded onto student computers.

Student Files

Students will need the **DayFour** folder found in the **StudentFiles** section of the Resource Disc. This folder contains files related to the programs:

ObjectCircle (ObjectCircle.java,TurnRight.java,DriveStraight.java)
Avoid(Avoid.java, TurnLeft.java, DriveStraight.java)

Day Four
KEYNOTE



Keynote

The Keynote for day three is found on the Resource Disc at the path **/Presentations/DayFour/**. Today's keynote contains 15 Slides and will take an estimated hour to complete. Note that examples are embedded in the keynote.

Keynote Outline

Slide One - Title

Slide Two - Review

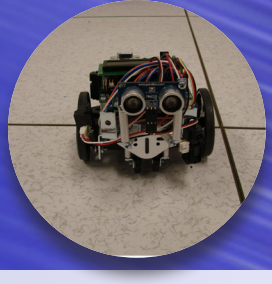
Yesterday we began using our robot's input features. We now know how to read input from the IR range sensors, sonar range sensors and photo reflector sensors. We will continue to put that knowledge to use today as we write behavior based programs.

Review with the students about what they learned yesterday. You don't need to go into a lot of detail but be sure to remind everyone of the basic concepts.

Slide Three - Behaviors

Think about some of the programs we have written over the previous days. We have created programs which tell our robot to do certain things based on certain conditions. We have essentially given our robot the ability to behave. Today we will take it a step further. The Java language along with the RidgeSoft libraries provides us with an easier and more efficient way to manage complex behaviors and multiple behaviors.

Day Four
KEYNOTE



Keynote Outline (cnt'd)

Slide four - How Behaviors Work
When we say we are creating different behaviors for our robot we are basically splitting up all of its possible reactions into different Java files, or better said we are creating an object for each type of behavior. Behaviors can include things like driving straight, turning left, turning right, stopping, or playing a sound.

A behavior object is created by first designing a custom Java class from which it can be created. A behavior object will contain all the variables it needs to access to determine if it should be active as well as the rules it should follow for deciding when to become active.
Once we have all of our behaviors created we create a controller object which will delegate automatically which behavior should be active.

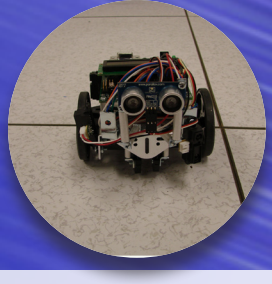
Slide Five - Behaviors Have
A behavior Object contains variables which link to data, like sonar readings; and other objects like Servo motors or the Display. Behaviors always include a poll method which contains the condition which decides if the behavior is active as well as the code to execute when it is active. Behaviors have a constructor which sets up all of the data. And finally they must have a setActive method.

Slide Six - Example Behavior
Review the code on the next few slides explaining and answering questions.

Each behavior must first implement the Behavior interface. It then lists the variables it needs access to. Next a constructor, which is called when this behavior object is created,takes in data and assigns it to the list of variables.

Slide Seven - Example Behavior
The poll method is required for each behavior. The method returns either true or false, as well as performs it's behavior action if it is needed. The if statement decides if the behavior should become the active one.

Day Four
KEYNOTE



Keynote Outline (cnt'd)

Slide Eight - Example Behavior
All behaviors have a setActive method which looks just like this, it is required, even though you will not use it directly. Also on this slide is the drive method we called from the poll method. You can add as many additional helper methods as you need to any behavior.

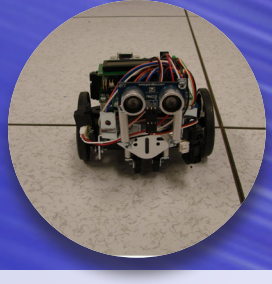
Slide Nine - Controlling Behaviors
Behaviors are controlled from a main class within the main method. First you create all of your behavior objects. Next, using a special type of data structure called an array, you place all of your behaviors into a list. Finally, you create a special object of the type BehaviorArbiter. The Behavior Arbiter will take the list of behaviors and a poll time, it will regulate which behaviors have control of the robot. It does this by asking each behavior if it needs to control the robot over and over. When a behavior needs to control the robot it allows it until a behavior with higher priority needs to control the robot.

Slide Ten - Controlling Behaviors
Explain the code examples of what was just discussed on the previous slide.

Slide Eleven - Object Circling
Our first hands on program today will involve using behaviors to drive our robot around a box object. We will use the formula of driving until the wall of the object runs out, then turning right, and repeating. To do this we will use the IR range sensors mounted on the right side of our robot.

Slide Twelve - Object Circle Example
This exercise will require us to have three classes, one main and two behaviors. You will need to add a little code to each class. It is more important today to read the code provided, as it may have surprises.

Day Four
KEYNOTE



Keynote Outline (cnt'd)

Slide Thirteen - Hands On Exercise
For today's exercises you will probably want to open the student template files on screen and work through the code with the students. ***If enough camp leaders are on hand it may be beneficial to work in small groups or one on one.***

You will now need to instruct students to begin the first hands on exercise, **ObjectCircle.java.** The templates for the students are available in the **DayFour** folder in the **StudentFiles** folder on the Resource Disc. You should already have an understanding of how this program works, but if you run into trouble you can always consult the **line by line guide**, found at the end of the Day Four section of this guide. Additionally the solutions are provided on the Resource Disc with a path of **/Solutions/DayFour**.

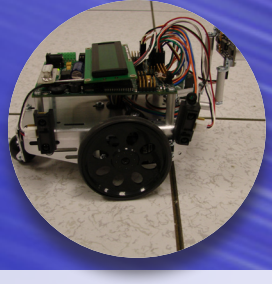
Slide Fourteen - Behavior Based Avoider
Our next task is to create a behavior based avoider. This program will use the sonar sensors to detect objects to close. In reaction to a close object the robot will turn left and continue driving.

Slide Fifteen - Hands On Exercise
For today's exercises you will probably want to open the student template files on screen and work through the code with the students. If enough camp leaders are on hand it may be beneficial to work in small groups or one on one.

You will now need to instruct students to begin the first hands on exercise, **Avoider.java.** The templates for the students are available in the **DayFour** folder in the **StudentFiles** folder on the Resource Disc. You should already have an understanding of how this program works, but if you run into trouble you can always consult the **line by line guide**, found at the end of the Day Four section of this guide. Additionally the solutions are provided on the Resource Disc with a path of **/Solutions/DayFour**.

Slide Sixteen - End of Day Four

Day Four
WRAP UP



Wrap Up

Once students have finished completing the exercises take questions and work to clear up any difficulties which were experienced. Tomorrow students will create one last behavior and then begin running their robots through a maze.

Line by Line - Day Four

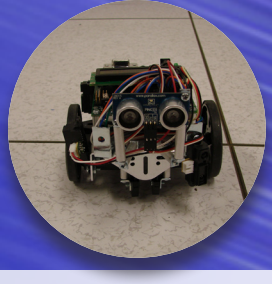
ObjectCircle.java (Motion Methods not shown in this document)

Source And Description:

```
//Import the needed items
import com.ridgesoft.robotics.behaviors.AbstractBehavior2;
import com.ridgesoft.robotics.behaviors.*;
import com.ridgesoft.robotics.*;
import com.ridgesoft.io.Display;
import com.ridgesoft.intellibrain.IntelliBrain;
import com.ridgesoft.robotics.RangeFinder;
import com.ridgesoft.robotics.sensors.SharpGP2D12;
import com.ridgesoft.robotics.Servo;
import com.ridgesoft.robotics.PushButton;
import com.ridgesoft.robotics.SonarRangeFinder;
import com.ridgesoft.robotics.sensors.ParallaxPing;

public class ObjectCircle
{
   public static void main(String args[])
   {try{
         //Create an Object for the LCD Display
         Display display = IntelliBrain.getLcdDisplay();

         //Create The Servo Objects
         Servo leftWheel = IntelliBrain.getServo(1);
         Servo rightWheel = IntelliBrain.getServo(2);
```

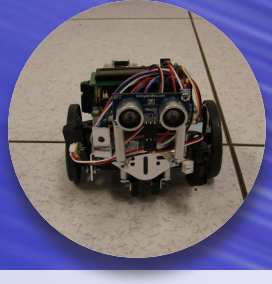
Day Four

LINE BY LINE



ObjectCircle.java (Cont'd) (Motion Methods not shown in this document)

```
//Create the Infrared Range Finder Objects
RangeFinder rearFinder = new SharpGP2D12(IntelliBrain.getAnalogInput(1), null);
RangeFinder frontFinder = new SharpGP2D12(IntelliBrain.getAnalogInput(2), null);

    //Setup Variables for the front and rear Distance readings
    float rearDistance=0.0f;
    float frontDistance=0.0f;
    display.print(1, "Obtaining Distance");

    //Now obtain the current distance, until we get a good reading.
    while(rearDistance<=0)
    {
        rearFinder.ping();
        rearDistance=rearFinder.getDistanceInches();
    }
    //The max distance is the reading +5.5
    float maxRearDistance=rearDistance+5.5f;

    //Print out the data to see
    display.print(1, "RActual: "+rearDistance);
    Thread.sleep(3000);
    display.print(1, "RMAX: "+maxRearDistance);
    Thread.sleep(2000);

    //NOW FRONT DISTANCE, do the same steps
    display.print(1, "Obtaining Distance");

    while(frontDistance<=0)
    {
    frontFinder.ping();
    frontDistance=frontFinder.getDistanceInches();
    }
    float maxFrontDistance=rearDistance+5.5f;
```

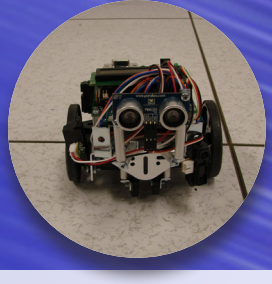


ObjectCircle.java (Cont'd) (Motion Methods not shown in this document)

```
        display.print(0,"Object Circle");
        display.print(1,"Running");

        //Create a behavior of each type
        Behavior driveStraight = new DriveStraight(rightWheel,leftWheel,true);
        Behavior turnRight = new TurnRight(leftWheel, rightWheel, frontFinder, rearFinder,
maxFrontDistance, maxRearDistance, true);

        //Create an array and put all of your behaviors in order of highest priority to lowest
        Behavior behaviors[] = new Behavior[] { turnRight, driveStraight };

//Create an Arbiter and give it the array of behaviors and a poll time in this case .5 seconds
        BehaviorArbiter control = new BehaviorArbiter(behaviors, 500);

        //Give the arbiter a priority
        control.setPriority(Thread.MAX_PRIORITY);

        //Start the process
        control.start();

   }catch(Throwable t){ t.printStackTrace();}} //Close the main method and try block
//Motion methods not shown in this document
} //Close the class
```

Day Four

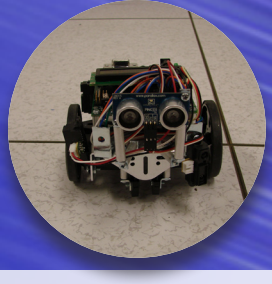
LINE BY LINE



DriveStraight.java (Motion Methods not shown in this document)

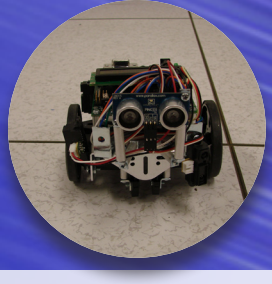
Source And Description:

```
import com.ridgesoft.robotics.behaviors.AbstractBehavior2;
import com.ridgesoft.robotics.behaviors.*;
import com.ridgesoft.robotics.*;
import com.ridgesoft.io.Display;
import com.ridgesoft.intellibrain.IntelliBrain;
import com.ridgesoft.robotics.RangeFinder;
import com.ridgesoft.robotics.sensors.SharpGP2D12;
import com.ridgesoft.robotics.Servo;
import com.ridgesoft.robotics.PushButton;
import com.ridgesoft.robotics.SonarRangeFinder;
import com.ridgesoft.robotics.sensors.ParallaxPing;

public class DriveStraight implements Behavior
{
        //All the stuff we need to access
        private Servo leftWheel;
        private Servo rightWheel;
        private boolean active;

        //Set up a constructor to take in the data
        public DriveStraight(Servo l, Servo r, boolean a)
        {
        leftWheel=l;
        rightWheel=r;
        active=a;
        }

//The poll method will decide if we control the program.
//NOTE: we always want control of the program, becuase we have the lowest priority
//              we know nothing else will be denied control because of us.
```



DriveStraight.java (Cnt'd) (Motion Methods not shown in this document)

```
        public boolean poll()
        {
        //Drive straight and return true
        drive(leftWheel,rightWheel);
        return true;
        }

        public void setActive(boolean b)
        {
        active=b;
        }

        public static void drive(Servo l, Servo r)
        {
                l.setPosition(100);
                r.setPosition(0);
        }

}
```



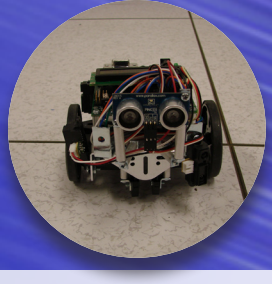
TurnRight.java (Motion Methods not shown in this document)

Source And Description:

```
import com.ridgesoft.robotics.behaviors.AbstractBehavior2;
import com.ridgesoft.robotics.behaviors.*;
import com.ridgesoft.robotics.*;
import com.ridgesoft.io.Display;
import com.ridgesoft.intellibrain.IntelliBrain;
import com.ridgesoft.robotics.RangeFinder;
import com.ridgesoft.robotics.sensors.SharpGP2D12;
import com.ridgesoft.robotics.Servo;
import com.ridgesoft.robotics.PushButton;
import com.ridgesoft.robotics.SonarRangeFinder;
import com.ridgesoft.robotics.sensors.ParallaxPing;

public class TurnRight implements Behavior
{
        //Declare variables we will need to look at for this behavior
        private boolean active;
        private Servo rightWheel;
        private Servo leftWheel;
        private RangeFinder front;
        private RangeFinder rear;

        //MAX
        private float maxRangeFront;
        private float maxRangeRear;

        private float rearDistance;
        private float frontDistance;
```

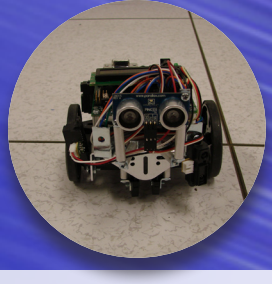
Day Four

LINE BY LINE



TurnRight.java (Cnt'd) (Motion Methods not shown in this document)

```
//Create a Constructior for the behavior which sets up all the varaibles
	public TurnRight(Servo l, Servo r, RangeFinder f, RangeFinder rr, float mF, float mR,
boolean enabled)
	{
		rightWheel=r;
		leftWheel=l;
		front=f;
		rear=rr;
		maxRangeFront=mF;
		maxRangeRear=mR;
		active=enabled;
		rear.ping();
		rearDistance=rear.getDistanceInches();
		front.ping();
		frontDistance=front.getDistanceInches();

	}
	//Create a poll method which will decide if this behavior is active
	//NOTE: this poll behavior actually contains the code which should run if the behavior
	//is active, because of the way the Behavior interface is set up it must be done this way.
//It is possible that this poll method will stall the polling process, but only if it is active -- in
//which case it would be a good thing

	public boolean poll()
	{
	//Check the distances
	rear.ping();
	rearDistance=rear.getDistanceInches();
	front.ping();
	frontDistance=front.getDistanceInches();
	//IF we are too far away from the wall or the wall has disapppeared then we need to act
	if((rearDistance>maxRangeRear || rearDistance<=0) && (frontDistance>maxRangeFront || frontDistance<=0))
	{
```

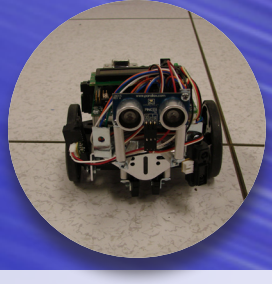


TurnRight.java (Cnt'd) (Motion Methods not shown in this document)

```
        //Turn right
        turnRight(leftWheel,rightWheel);
        //Now as long as there is no wall we need to drive until we find one then release control
        while((rearDistance>maxRangeRear || rearDistance<=0) || (frontDistance>maxRangeFront || frontDistance<=0))
        {
                rear.ping();
                rearDistance=rear.getDistanceInches();
                front.ping();
                frontDistance=front.getDistanceInches();
                drive(leftWheel,rightWheel);
        }
        //finally return true, even though our work is done
        return true;
    }
    else
     {
        //if we didn't need to turn we return false, we don't want control!
        return false;
     }
}

public void setActive(boolean b)
{
        active=b;
}

}
```

Day Four

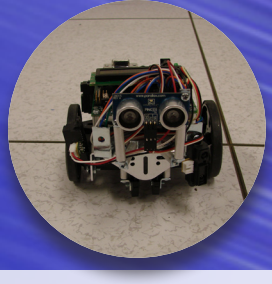
LINE BY LINE



Avoid.java (Motion Methods not shown in this document)

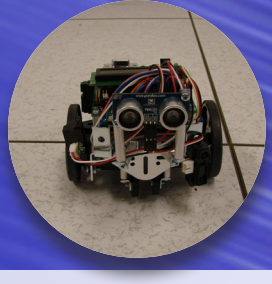
Source And Description:

```
//Import the needed items
import com.ridgesoft.robotics.behaviors.AbstractBehavior2;
import com.ridgesoft.robotics.behaviors.*;
import com.ridgesoft.robotics.*;
import com.ridgesoft.io.Display;
import com.ridgesoft.intellibrain.IntelliBrain;
import com.ridgesoft.robotics.RangeFinder;
import com.ridgesoft.robotics.sensors.SharpGP2D12;
import com.ridgesoft.robotics.Servo;
import com.ridgesoft.robotics.PushButton;
import com.ridgesoft.robotics.SonarRangeFinder;
import com.ridgesoft.robotics.sensors.ParallaxPing;

public class Avoid
{
   public static void main(String args[])
   {try{
         //Create an Object for the LCD Display
         Display display = IntelliBrain.getLcdDisplay();

         //Create The Servo Objects
         Servo leftWheel = IntelliBrain.getServo(1);
         Servo rightWheel = IntelliBrain.getServo(2);

         //Create the Sonar Sensor
         SonarRangeFinder ahead = new ParallaxPing(IntelliBrain.getDigitalIO(4));
         float maxDistance=4.5f;
         float distance=0.0f;
         ahead.ping();
         Thread.sleep(100);
         distance=ahead.getDistanceInches();
```



Avoid.java (Cnt'd) (Motion Methods not shown in this document)

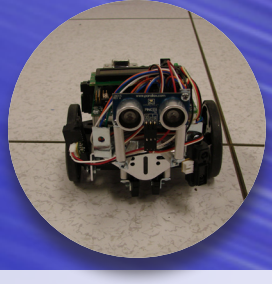
```
        //Create a behavior of each type
        Behavior driveStraight = new DriveStraight(leftWheel,rightWheel,true);
        Behavior turnLeft = new TurnLeft(leftWheel, rightWheel, ahead, maxDistance, dis-
tance, true);

        //Create an array and put all of your behaviors in order of highest priority to lowest
        Behavior behaviors[] = new Behavior[] { turnLeft, driveStraight };

//Create an Arbiter and give it the array of behaviors and a poll time in this case .5 seconds
        BehaviorArbiter control = new BehaviorArbiter(behaviors, 500);

        //Give the arbiter a priority
        control.setPriority(Thread.MAX_PRIORITY);

        //Start the process
        control.start();

  }catch(Throwable t){ t.printStackTrace();}}
//Motion Methods not shown in this document
}
```



TurnLeft.java (Motion Methods not shown in this document)

Source And Description:

```
import com.ridgesoft.robotics.behaviors.AbstractBehavior2;
import com.ridgesoft.robotics.behaviors.*;
import com.ridgesoft.robotics.*;
import com.ridgesoft.io.Display;
import com.ridgesoft.intellibrain.IntelliBrain;
import com.ridgesoft.robotics.RangeFinder;
import com.ridgesoft.robotics.sensors.SharpGP2D12;
import com.ridgesoft.robotics.Servo;
import com.ridgesoft.robotics.PushButton;
import com.ridgesoft.robotics.SonarRangeFinder;
import com.ridgesoft.robotics.sensors.ParallaxPing;

public class TurnLeft implements Behavior
{
	//All the stuff we need to access
	private Servo leftWheel;
	private Servo rightWheel;
	private SonarRangeFinder ahead;
	private boolean active;
	private float maxDistance;
	private float distance;

	//Set up a constructor to take in the data
	public TurnLeft(Servo l, Servo r, SonarRangeFinder ah, float md, float d, boolean a)
	{
		leftWheel=l;
		rightWheel=r;
		active=a;
		ahead=ah;
		maxDistance=md;
		distance=d;
	}
```

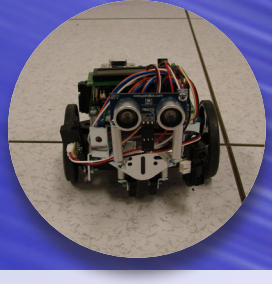


TurnLeft.java (Cnt'd) (Motion Methods not shown in this document)

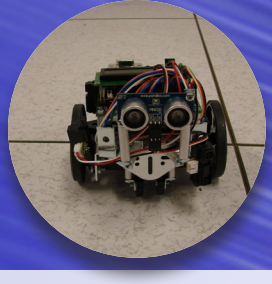
```
        //Polling method, pings the sonar to find out if we need control, does steps and returns
        //true if we do, otherwise returns false

        public boolean poll()
        {
        ahead.ping();
                try{
                Thread.sleep(100); } catch(Throwable t){}
                distance=ahead.getDistanceInches();

                if(distance<maxDistance && distance>0)
                {
                        turnLeft(leftWheel,rightWheel);
                        return true;
                }
                else
                {
                        return false;
                }

        }
        public void setActive(boolean b)
        {
                active=b;
        }
//Motion methods not shown in this document
}
```

Day Four

NOTES



Day Five
GOALS



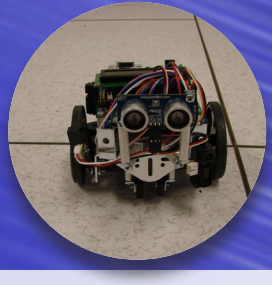
Key Concepts

- **Combining All Camp Concepts**

Goals of the Day

- **Solving a Maze**

Day Five
PREPARE



General

Day Five is focused mainly on the students developing one small behavior for their robot, and then attempting to solve a maze with it. You will need to setup the maze ahead of time. See the **Pre-Camp** section for details about recommedations for mazes. It is a good idea to review the keynote slides and outline ahead of time as well. You will need to be familiar with all the programs, demos, and examples ahead of time.

Demos

Day Five does not contain any demos. Instead you will need to work actively with students in perfecting their sensor based programs.

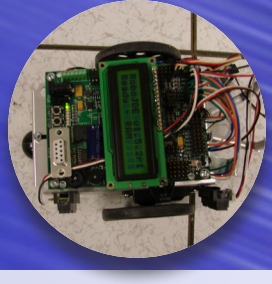
Programs

Students will need the **DayFive** folder found in the **StudentFiles** section of the Resource Disc. This folder contains files related to the programs:

MazeSolver- uses four behaviors to solve a maze. ***You will need a maze setup for this activity.***

You need to review these programs ahead of time, so you can assist students. As usual you can find a detailed description in the **line by line** guide. Additionally you can review the solution files for insight into the programs.

Day Five
BEGIN



Getting Started

Get started by welcoming everyone back to the programming/lab portion of the camp. Tell students that today they will write one more behavior and then spend the rest of the session attempting the maze. Once everyone is ready, begin the session by following the Day Five Keynote outline.

Distribution of Materials

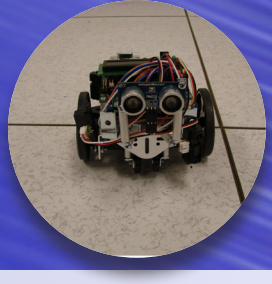
Robots will be needed for Day Five activities, so if you do not choose to distribute them ahead of time, you will need to do so during the presentation. Remember you need to check the robot configurations ahead of time, or allow students time to do so. Also student files will need to be loaded onto student computers.

Student Files

Students will need the **DayFive** folder found in the **StudentFiles** section of the Resource Disc. This folder contains files related to the programs:

MazeSolver (TurnLeft.java,TurnRight.java,DriveStraight.java,StopBot.java MazeSolver.java)

Day Five
KEYNOTE



Keynote

The Keynote for day three is found on the Resource Disc at the path **/Presentations/DayFive/**. Today's keynote contains 8 Slides and will take an estimated thirty minutes to complete. Note that examples are embedded in the keynote.

Keynote Outline

Slide One - Title

Slide Two - Solving A Maze

Today we will be combining all of the knowldge we have gained about programming the Intellibrain robot into solving the problem of a maze. There are several different methods for solving a maze. What do you think, given the Intellibrains ablilites, would be the easiest way to solve a maze?

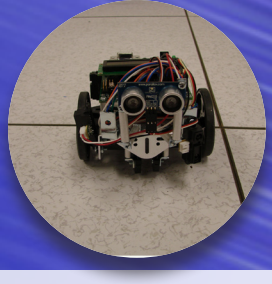
Engage the students in a discussion concerning the easiest way to solve a maze with the Intellibrain.

Slide Three - Options

A maze is typically solved one of three ways. Option number one would be to allow the robot to randomly drive around the maze until it reaches the end. This would work, but it could take hours for the robot to happen upon the finish. The second option involves having the robot follow one wall of the maze until it finds the end. As long as the maze doesn't end on an island then the robot will find the end eventually. The third option is for the robot to make a map of the maze as it drives around, until it can find its way to the finish.

We will choose option number two because, as you will soon find out we already have a lot of tools in place for this method.

Day Five
KEYNOTE



Keynote Outline (cnt'd)

Slide Four - Behaviors We Need
Thinking about solving the issue of following a wall around the maze, we can divide the process into four behaviors. First, we must drive straight. Then, if the wall we are following goes away, there must be a turn, we should turn right and continue following it. But, if we approach another wall head on we must be coming to a left turn or dead end, so we should turn left. And finally, if we arrive at the finish we should stop.

Great news, we've already programmed three of the four behaviors you will need for solving the maze!

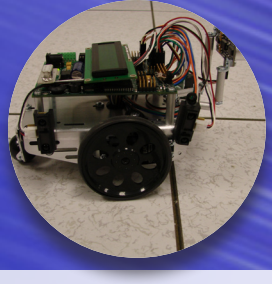
Slide Five - Stop At Finish
In order to add the stop behavior we will utilize the photo reflectors. The end of our maze will be a black sheet of paper, the rest of our maze floor will be a lighter color. By scanning the floor the robot will know when it has reached the end. The StopBot behavior will simply stop the robot when the dark color is found.

Slide Six - Maze Example
Discuss this sample maze and answer any questions students may have about the maze solving process.

Slide Seven - Hands On Exercise
You will now need to instruct students to begin the first hands on exercise, **MazeSolver.java.** The templates for the students are available in the **DayFive** folder in the **StudentFiles** folder on the Resource Disc. You should already have an understanding of how this program works, but if you run into trouble you can always consult the **line by line guide**, found at the end of the Day Four section of this guide. Additionally the solutions are provided on the Resource Disc with a path of **/Solutions/DayFive**.

Slide Eight - End Day Five

Day Five
WRAP UP



Wrap Up

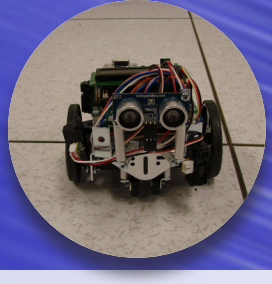
Once students have finished completing the exercises take questions and work to clear up any difficulties which were experienced. Thank the students for spending the week learning about computer science. Remind them that computer science is a large and growing field, robotics is only one small part.

Line by Line - Day Five

MazeSolver.java (Motion Methods not shown in this document)

Source And Description:

```
//Import the needed items
import com.ridgesoft.robotics.behaviors.AbstractBehavior2;
import com.ridgesoft.robotics.behaviors.*;
import com.ridgesoft.robotics.*;
import com.ridgesoft.io.Display;
import com.ridgesoft.intellibrain.IntelliBrain;
import com.ridgesoft.robotics.RangeFinder;
import com.ridgesoft.robotics.sensors.SharpGP2D12;
import com.ridgesoft.robotics.Servo;
import com.ridgesoft.robotics.PushButton;
import com.ridgesoft.robotics.SonarRangeFinder;
import com.ridgesoft.robotics.sensors.ParallaxPing;

public class MazeSolver
{
   public static void main(String args[])
        {try{
                //Create All the objects we Need
                Display display = IntelliBrain.getLcdDisplay();
                Servo leftWheel = IntelliBrain.getServo(1);
                Servo rightWheel = IntelliBrain.getServo(2);
                SonarRangeFinder ahead = new ParallaxPing(IntelliBrain.getDigitalIO(4));
```

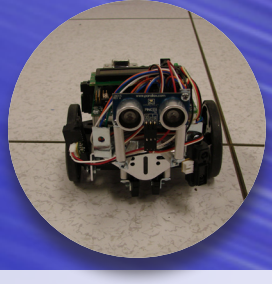


MazeSolver.java (Cnt'd) (Motion Methods not shown in this document)

```
RangeFinder rearFinder = new SharpGP2D12(IntelliBrain.getAnalogInput(1), null);
RangeFinder frontFinder = new SharpGP2D12(IntelliBrain.getAnalogInput(2), null);
AnalogInput leftLineSensor = IntelliBrain.getAnalogInput(6);
AnalogInput rightLineSensor = IntelliBrain.getAnalogInput(7);
PushButton startButton = IntelliBrain.getStartButton();

//Setup All the variables we will need to track distances, colors, etc.

//Sonar
float maxDistance=4.5f;
float distance=0.0f;
//IR Range
float rearDistance=0.0f;
float frontDistance=0.0f;
float maxFrontDistance=0.0f;
float maxRearDistance=0.0f;
//Photo Reflector
int leftLineReading=0;
int rightLineReading=0;
int maxReading=0;
int reading=0;

//First thing we ask the user to show us the finish zone
display.print(0,"MOVE BOT TO");
display.print(1,"FINISH ZONE");
Thread.sleep(4000);
display.print(0,"THEN PRESS");
display.print(1,"START");
startButton.waitPressed();
display.print(0, "MazeSolver");
display.print(1, "Obtaining Distance");
```

Day Five

LINE BY LINE

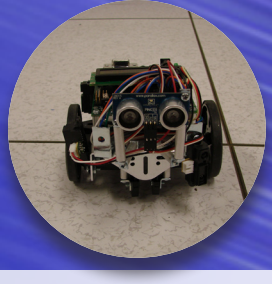


MazeSolver.java (Cnt'd) (Motion Methods not shown in this document)

```
vaV   leftLineReading=leftLineSensor.sample();
      rightLineReading=rightLineSensor.sample();
      maxReading=(leftLineReading+rightLineReading)/2;

      //Now Ask User to move bot to startin position
      display.print(0,"MOVE BOT TO");
      display.print(1,"START POSITION");
      Thread.sleep(4000);
      display.print(0,"THEN PRESS");
      display.print(1,"START");
      startButton.waitPressed();
      display.print(0, "MazeSolver");
      display.print(1, "Obtaining Distance");

      //Now we read this in for our reading
       leftLineReading=leftLineSensor.sample();
       rightLineReading=rightLineSensor.sample();
       reading=(leftLineReading+rightLineReading)/2;

      //Now we capture the current sonar distance
      ahead.ping();
      Thread.sleep(100);
      distance=ahead.getDistanceInches();

      //Now we begin capturing the IR range distances
      //NOTE: IF the program stalls on the Obtaining Distance Screen too long, you may
      //want to nudge the bot gently it is usually the IR range sensors not reading. A simple
      //nudge usually corrects the problem.
      //Now obtain the current rear distance, until we get a good reading.
      while(rearDistance<=0)
      {
      rearFinder.ping();
```

Day Five

LINE BY LINE

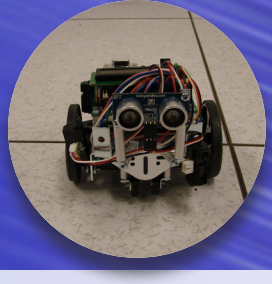


MazeSolver.java (Cnt'd) (Motion Methods not shown in this document)

```
        rearDistance=rearFinder.getDistanceInches();
        }

                //The max distance is the reading +5.5
                maxRearDistance=rearDistance+5.5f;

                //NOW FRONT DISTANCE, do the same steps
                while(frontDistance<=0)
                {
                frontFinder.ping();
                frontDistance=frontFinder.getDistanceInches();
                }
                maxFrontDistance=rearDistance+5.5f;

                //Create a behavior of each type
                Behavior driveStraight = new DriveStraight(leftWheel,rightWheel,true);
                Behavior turnLeft = new TurnLeft(leftWheel, rightWheel, ahead, maxDistance,
distance, 625, true);
                Behavior turnRight = new TurnRight(leftWheel, rightWheel, frontFinder,
rearFinder, maxFrontDistance, maxRearDistance, 625, true);
                Behavior stopBot = new StopBot(leftWheel,rightWheel,rightLineSensor, leftLi-
neSensor, reading, maxReading, true);

//Create an array and put all of your behaviors in order of highest priority to lowest
        Behavior behaviors[] = new Behavior[] { stopBot,turnLeft, turnRight, driveStraight };
//Create an Arbiter and give it the array of behaviors and a poll time in this case .5 seconds
        BehaviorArbiter control = new BehaviorArbiter(behaviors, 500);
//Give the arbiter a priority
        control.setPriority(Thread.MAX_PRIORITY);
//Start the process
        control.start();
   }catch(Throwable t){ t.printStackTrace();}}
//Motion Methods not included in this document
}
```

Day Five

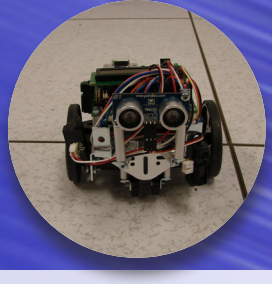
LINE BY LINE



StopBot.java (Motion Methods not shown in this document)

Source And Description:

```
import com.ridgesoft.robotics.behaviors.AbstractBehavior2;
import com.ridgesoft.robotics.behaviors.*;
import com.ridgesoft.robotics.*;
import com.ridgesoft.intellibrain.IntelliBrain;
import com.ridgesoft.robotics.Servo;

public class StopBot implements Behavior
{
        //All the stuff we need to access
        private Servo leftWheel;
        private Servo rightWheel;
        private boolean active;
        private AnalogInput leftLineSensor;
        private AnalogInput rightLineSensor;
        private int reading;
        private int max;

        //Set up a constructor to take in the data
        public StopBot(Servo l, Servo r, AnalogInput ls, AnalogInput rs, int re, int m, boolean a)
        {
                leftWheel=l;
                rightWheel=r;
                leftLineSensor=ls;
                rightLineSensor=rs;
                reading=re;
                max=m;
                active=a;
        }
```

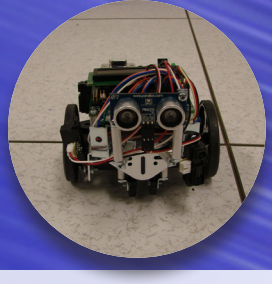


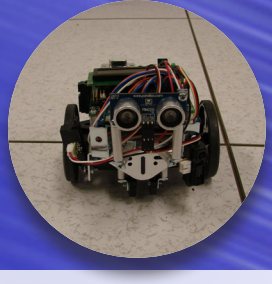
StopBot.java (Cnt'd) (Motion Methods not shown in this document)

```
//The poll method which will decide if we control the program.
//NOTE: we always want control of the program, becuase we have the lowest priority
//              we know nothing else will be denied control because of us.

public boolean poll()
{
        reading=(leftLineSensor.sample() + rightLineSensor.sample())/2;
        if(reading>=(max-70))
        {
        leftWheel.off();
        rightWheel.off();
        System.exit(1);
        return true;
        }
        else
        {
        return false;
        }

}

public void setActive(boolean b)
{
        active=b;
}

}
```

Day Five

NOTES





/*

Created for:

Lamar University
Computer Science Department
WIRED/INSPIRE
Summer 2007

*/