



Introduction to **Robots and the Mind** *- Programming Basics -*

Bert Wachsmuth & Michael Vigorito
Seton Hall University

Programming Groups

Albrecht	Brittney	&	Lino	Jennifer
Bie	Jennifer	&	Guida	David
		&		
De Clerico	Mario	&	Tigol	Angelo
Lambraia	Jonathan	&	Torres	Chelsea
		&		
Mahan	Kelsey	&	Gleason	Paige
McCaskill	Lauren	&	Perry	Thomas
Naslonski	Paulina	&	Wager	Tara
		&		
			Adams-	
Oliver	Keenan	&	Martinez	Shomari
		&		
Ralston	Kristen	&	Loughrey	Dana
		&		
			Brutus-	
Schindler	Kimberly	&	Foulkes	Rezina
Singh	Diljeet	&	Rubenstein	Kimberly
Tietchen	Shannon	&	Nivar	Aileen
		&		





Programming

“Creating a sequence of instructions to enable the computer/robot to do something”

[http://wordnetweb.princeton.edu/perl/webwn?s=computer programming](http://wordnetweb.princeton.edu/perl/webwn?s=computer%20programming)

- 1. Create** the program , i.e. the sequence of instructions.
Most spoken languages are full of ambiguities, so we use a *special* language instead, such as Java (or C++, Perl, Scheme, Python, or ...)
- 2. Translate** the program into instructions that the computer processor can understand
- 3. Execute** the instructions and test the program



Creating a Program

- ◆ Need to learn the grammar and vocabulary of our special language of choice (Java)
- ◆ Need a special editor, preferably with a build-in spell-checker for our language
- ◆ Need a mechanism to translate and execute our program



Basic Grammar of Java

- ◆ A (Java) program is a sequence of statements, one per line
- ◆ Java is case-sensitive, i.e. the word “LCD” and “Lcd” are considered different.
- ◆ A valid Java statement must end with a semi-colon ; unless it starts a group.
- ◆ Java uses three sets of parenthesis/brackets:
 - curly brackets “{ ... }” to group statements together
 - regular parenthesis “(...)” to denote inputs to functions and for math expressions
 - square brackets “[...]” to denote what’s called arrays

Basic Grammar of Java

Every (almost) Java program has a unique **program name** and includes as a minimum **the following lines**, known as the **standard framework**:

```
public class ProgramName
{
    // One-line comment describing the program in English
    public static void main(String[] args)
    {
        /*
            describes any necessary details using
            multi-line comments
        */
    }
}
```

Java Programs: *easy to read* ...

```

public class MysteryProgram
{
    public static void main(String args[])
    {
        LCD.drawString("Welcome", 0, 0);

        Motor.B.rotate(720);
        UltrasonicSensor sensor =
            new UltrasonicSensor(SensorPort.S3);
        if (sensor.getDistance() < 10)
        {
            Sound.playTone(440, 5);
        }
    }
}

```



Java Programs: difficult to create..

- ◆ Create a program that:
 - (a) Plays an “intro” tune
 - (b) Rotates a motor
 - (c) Shows a string on the screen
 - (d) Plays an “exit” tune





Creating a “correct” Program

- ◆ **Create source code** according to the Java grammar
- ◆ **Compile** the code into machine language
- ◆ **Execute** and test the program

Repeat until your program correctly solves the task.
Sometimes it helps to first solve simpler tasks ...

Creating a “correct” Program

- ◆ Create a program that (a) plays “intro” tune

... first ...

- ◆ Create a program that plays a *single* note



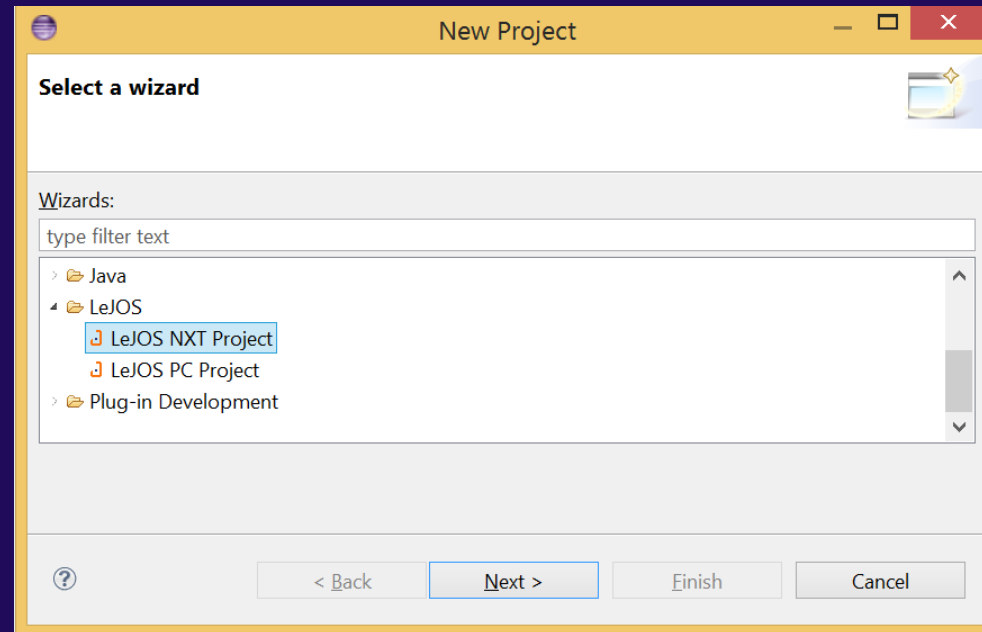


Gathering the Ingredients

1. **Create a new project**
2. **Create a new class containing our “standard framework”**
3. **Learn how to play notes and add the corresponding code to the framework**
4. **Execute the program and test it**
5. **Expand the program to solve original task**
6. **Test and refine if possible**

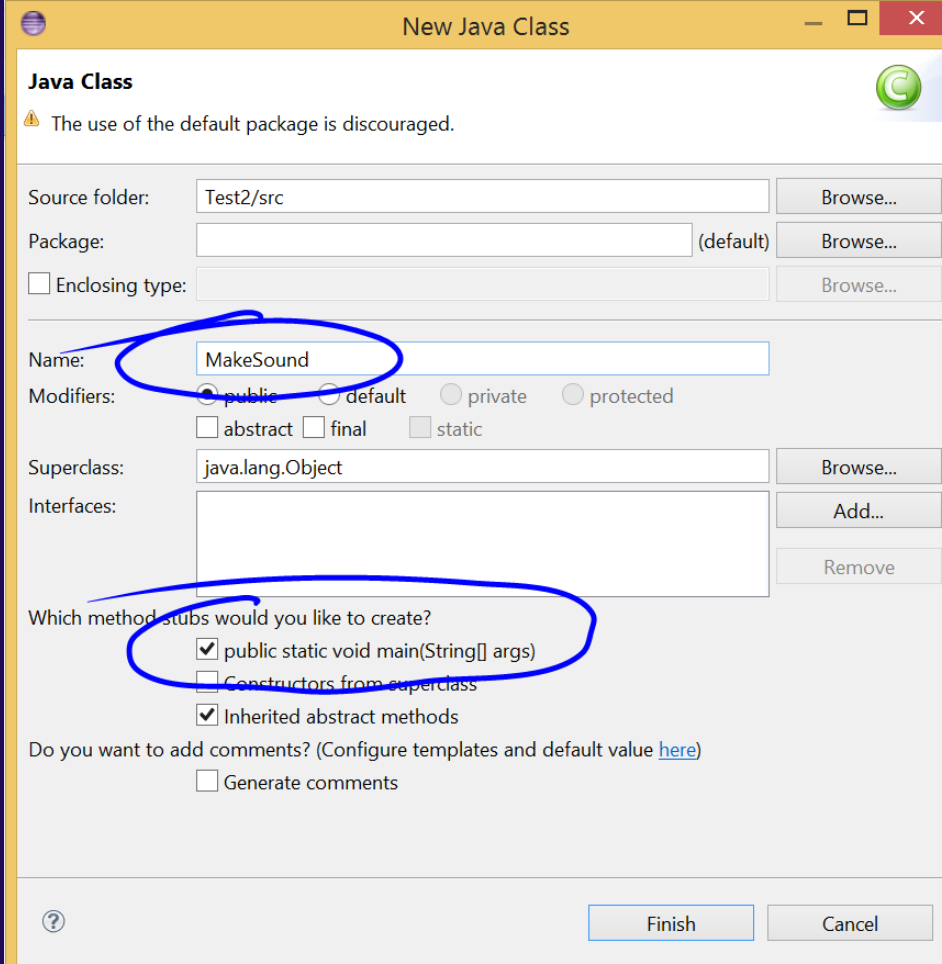
Create a new Project

- ◆ Click on “File | New Project”
- ◆ Expand “LeJOS”, highlight “LeJOS NXT Project” and click “Next”
- ◆ Enter a name for your project (no spaces or special characters), then hit “Finish”



Create Class with “Standard Framework”

- ◆ Highlight the new project in the “Project Explorer”
- ◆ Click on “File | New” and pick “Class”
- ◆ Enter a name for your class, such as “MakeSound” (remember, no spaces!)
- ◆ Check to create the method “public static void main”
- ◆ **Note** that for now you can think of “class” as a “program”



Java Class

The use of the default package is discouraged.

Source folder: Test2/src Browse...

Package: (default) Browse...

Enclosing type: Browse...

Name: MakeSound

Modifiers: public default private protected
 abstract final static

Superclass: java.lang.Object Browse...

Interfaces: Add...
Remove


Which method stubs would you like to create?

public static void main(String[] args)
 Constructors from superclass
 Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))
 Generate comments

? Finish Cancel

A Complete Robot Program



```
public class MakeSound {  
  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
    }  
}
```

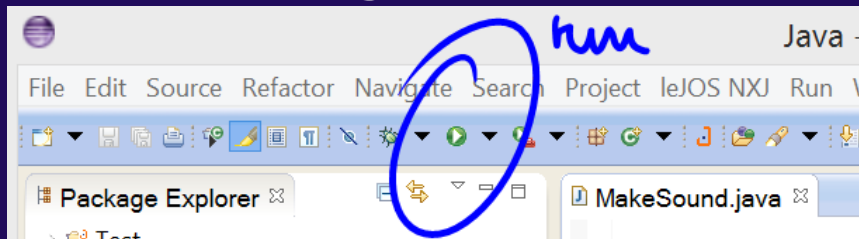
A Complete Robot Program

```
public class MakeSound
{
    public static void main(String[] args)
    {
        // TODO Auto-generated method stub
    }
}
```

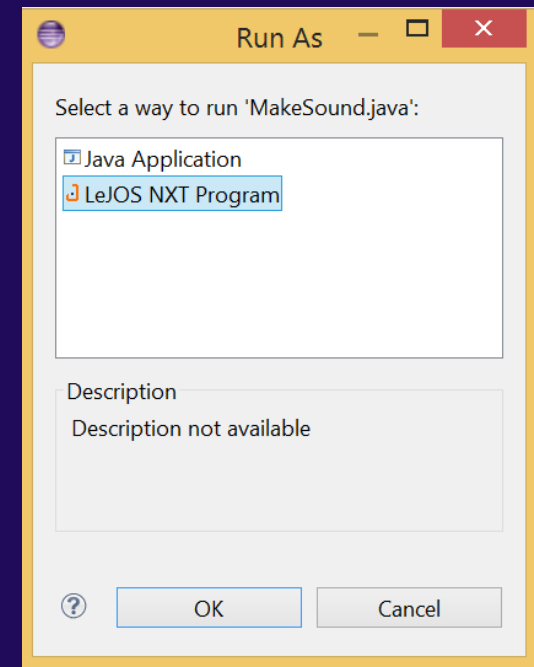


Executing the Program

- ◆ Plug-in the NXT brick and **turn it on**
- ◆ Click the green “run” button in the tool bar



- ◆ Select “LeJOS NXT Program” and click “OK”



The program will now be *linked*,
downloaded to the brick, and *executes* (runs) – of course it currently does nothing but you should not see any error.

Fixed NXT Components

- ◆ The NXT brick includes many *named components* such as LCD, Sound, Motor, etc.
- ◆ Some have fixed properties; programming those is easy: use them by **name** and call on their built-in **functions** using the syntax

Component.function(optional input)

- ◆ *Note: in proper Java lingo such functions are called **static methods***



The “Sound” Component

- ◆ The **Sound** component supports the following static methods to generate music:
 - **Sound.beep()**
 - **Sound.beepSequence()**
 - **Sound.beepSequenceUp()**
 - **Sound.buzz();**
 - **Sound.pause(milliseconds)**
 - **Sound.playTone(freq, duration)**

