TOPIC 3
INTRODUCTION TO PROGRAMMING

Notes adapted from Introduction to Computing and Programming with Java: A Multimedia Approach by M. Guzdial and B. Ericson, and instructor materials prepared by B. Ericson.

Outline

- To create **objects**
  - Using the new operator
- To declare **reference variables**
  - In order to refer to objects
- To learn about **object methods**
  - Send **messages to objects** to ask them to do something
- To create a **method** to perform a task
- To learn about **class methods**

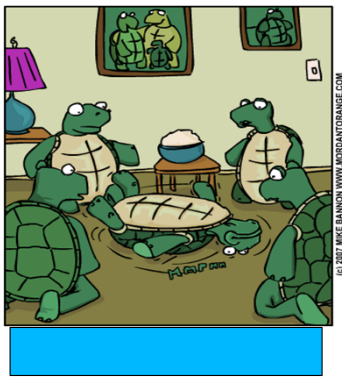
3

Creating objects

Making “things” to program with.

Turtles

4



A very intelligent turtle

Found programming UNIX a hurdle

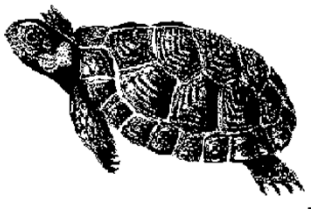
The system, you see,

Ran as slow as did he,

And that's not saying much for the turtle.

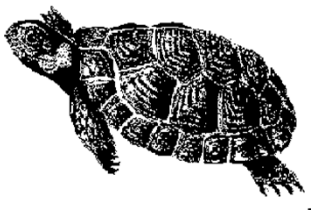
Turtles

- Goal: to take a closer look at classes and objects in Java
- How? We will work with **Turtle objects** that can be moved around the screen in their own world
  - They can be moved forward, backward, turned, …
  - Each turtle has a pen, which can leave a trail



Turtles in Java

- We need to define what we mean by a **Turtle**, to Java and to the computer (in other words, we need to **tell the computer** what we mean by Turtle)
- We have a Turtle class definition
  - Stored in a file called Turtle.java
  - Part of a number of classes created for use with the course text



Syntax vs Semantics

7

- Syntax means the rules for defining a program, the rules we must follow to make code that compiles
  - In English, syntax would mean putting periods at the end of sentences, not ending sentences with a proposition, starting a sentence with a capital, etc
- Semantics is the meaning behind what you wrote
  - For example, in English → "Dog, bread to chair!" is syntactically correct, but makes no sense! The semantics aren't right
- In computer programming, the syntax are the rules for how to type in code so it compiles – such as ending statements with a semi colon, or all the 'orders' of words we will see.

Creating objects in Java

8

- To create an object of a class, we use a special keyword **new** with the **syntax**

  new ClassName(value, value, …);
- Our turtles example:
  - Our Turtle objects live in a World object
  - So, we must first create a World object, so that our turtles have a place to live
  - How do we create a World object? …

Creating objects in Java

- Let's try typing

  new World();

- This will create a new World object, displayed as :

- But we will not have any way to refer to it again!
  - Recall: this is why we need variables, so that we can access values later
  - We need to store it in a variable

Reference variables

- Recall that all variables are declared by

  type name;

- But the type of a reference variable is the name of the class, so we **declare a reference variable** by:

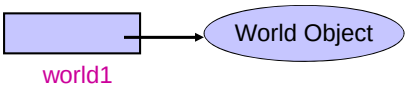
  Class name;

- We can declare a reference variable and have it refer to a new object in one statement:

  Class name = new Class(value, value, …);

Creating a new world

- Example: declare a reference variable world1 which refers to a new World object:



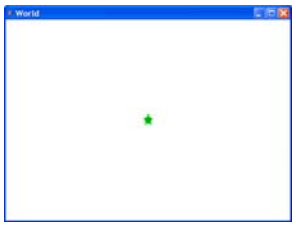

```
World world1;
world1 = new World();
```

- The world starts off with a size of 640 by 480, with no turtles

Creating a new turtle

- To create a Turtle object, we must specify the World object that it lives in:

```
Turtle turtle1 = new Turtle(world1);
```



- It starts off facing north and in the center of the world by default

Creating a new turtle

- Wait a sec… In that slide, we had:
  - `Turtle turtle1 = new Turtle(world1);`
- What is (world1)?
- We don't want the Turtle to have no where to live! So when we make the Turtle, we tell him to live in world1. If we wanted him in world8, we could first make world8, then make the Turtle live there
- Specifying things to an object when you make it is called using **parameters**
- There will be more on this later!

Creating several objects

- You can create several World objects:

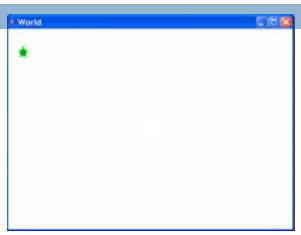
  `World world2 = new World();`

- You can also create several Turtle objects in one world:

  `Turtle turtle2 = new Turtle(world2);`

  `Turtle turtle3 = new Turtle(world2);`
  - Note that one turtle is on top of the other (why?)

Other ways to create turtles



- Turtles can also be created at different starting positions
- To start a turtle off at **position** (30,50) in a world created previously:

```
Turtle turtle4 = new Turtle(30, 50, world2);
```

Now we are telling the turtle to live in world2, and where to start off! Again, using **parameters**

- Turtle positions are given as **x and y values**
  - X starts at 0 on the left and increases horizontally to the right
  - Y starts at 0 at the top of the window and increases to the bottom

Turtle basics

- We can print out the status of our world and our turtles to see what's what:

```
System.out.println(world1);
System.out.println(turtle1);
```

Try this: what does it print?

- Being able to print out the "states" of objects can be very handy when debugging Java code

17

Object methods

How to do things with objects.

Actions are called "methods" in Java

- Now that we have created Turtle objects, we can perform actions on them
- An **action** performed on/by an **object** is a collection of Java statements called a **method**
  - More precisely, it is called an **object method** or **instance method**
  - We will see something called a class method later

Object methods

- A **method** is a **named** collection of statements that carry out a specific task
  - Example: a method called forward causes a turtle to move forward a specified distance
  - Example: a method called turnRight causes a turtle to turn to the right
- We can think of an object method as a **message sent to an object**, to do something
  - Example: send a message to a Turtle object to move forward

Defining methods in Java

- We **define** a method by writing the code that performs the action
  - Every method in Java must be defined inside a class
    - Remember, Java is **Object Oriented** and Objects are written in classes → if the method isn't in a class, it doesn't exist!
  - Example: the method forward is defined for the Turtle class
  - We will see what a method definition looks like later

Calling methods in Java

- We **call** (**invoke**) a method from a program when we want it to be **executed**
  - We can call methods written by others (e.g. the methods forward, turnRight, etc. for a turtle)
  - Or methods that we have written ourselves (later)
- An object method can only be executed on an object belonging to the class in which the method was defined
  - Example: our method forward can only be invoked on objects from the Turtle class

Calling methods in Java

- Object methods must be executed on an object, using the syntax

  objectReference.methodName()

- The object reference is typically the name of an object reference variable
  - Example: turtle1.forward();

Method parameters

- Methods may take input to act upon
- Input is **passed in** to the method in the form of a list of **parameters** in parentheses, given when the method is invoked, as in

  …methodName(parameter1, parameter2, …);

Remember, just like telling the turtle where to live!

- Example of a method call with a parameter :

  turtle1.forward(50);

Return values

- If a method produces a result, this result is **returned** by the method to wherever the method was invoked
  - We can think of the result as "replacing" the invocation of the method
  - Examples upcoming

Return Values vs Paramters

25

- An easy way to think of parameters and return values is to think of them like baseball
- When you call a method, you throw it parameters
- When its done, it pass you back the return value

Exercise: methods for strings

26

- Try the following String example:

```
String name = "Harry Potter";
String lowerName = name.toLowerCase();
System.out.println(lowerName);
String upperName = name.toUpperCase();
System.out.println(upperName);
System.out.println(name);
```

  - Notice that the value of name didn't change
  - Strings are called **immutable** objects: all String methods that modify a string do not change the original string, but return a new one.

27

Methods for turtles

What we can make turtles do!

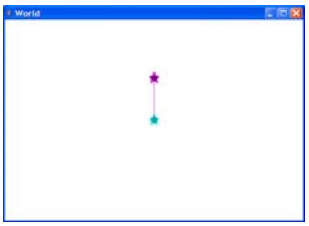
You'll need these for your first assignment

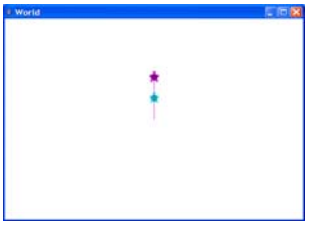
Moving a turtle

- Turtles can move forward:

  turtle3.forward();
- The default is to move by 100 steps (pixels)
- You can also tell the turtle how far to move (pass a **parameter**):

  turtle2.forward(50);
- There are corresponding backward() methods to move a turtle backward





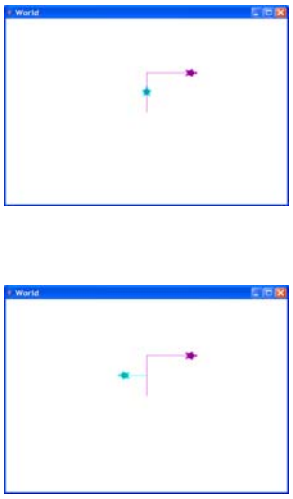
Turning a turtle

- Turtles can turn
  - Right

```
turtle3.turnRight();
turtle3.forward();
```

  - Left

```
turtle2.turnLeft();
turtle2.forward(50);
```



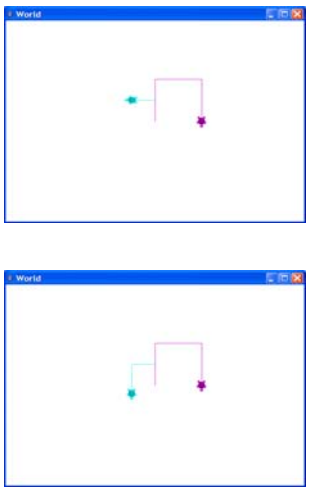
Turning a turtle

- Turtles can turn by a specified number of degrees
  - A positive number turns the turtle to the right

```
turtle3.turn(90);
turtle3.forward(100);
```

  - A negative number turns the turtle to the left

```
turtle2.turn(-90);
turtle2.forward(70);
```



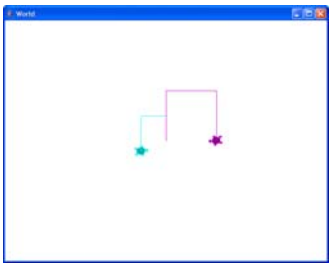
Turning a turtle

- Turtles can turn to face other turtles:

```
turtle2.turnToFace(turtle3);
turtle3.turnToFace(turtle2);
```

- Turtles can turn to face specific points:

```
turtle2.turnToFace(0,0);
turtle3.turnToFace(639,479);
```



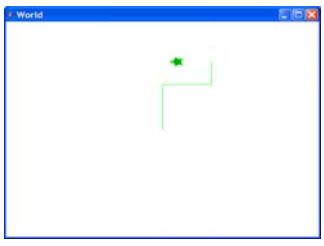
The pen

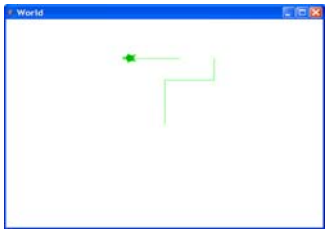
- Each turtle has a pen
- The default is to have the pen down to leave a trail
- You can pick the pen up:

```
turtle1.penUp();
turtle1.turn(-90);
turtle1.forward(70);
```

- You can put it down again:

```
turtle1.penDown();
turtle1.forward(100);
```



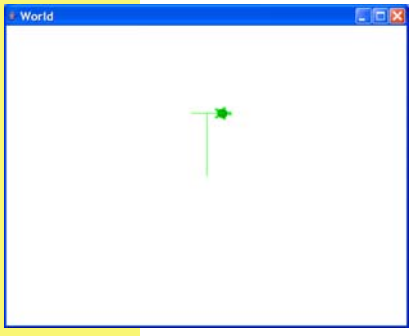


Drawing a **T**

- ***Algorithm*** (Steps in the process):
  - Create a World variable and a World object, and a Turtle variable and Turtle object
  - Ask the Turtle object to go forward 100
  - Ask the Turtle object to pick up the pen
  - Ask the Turtle object to turn left
  - Ask the Turtle object to go forward 25
  - Ask the Turtle object to turn 180 degrees
  - Ask the Turtle object to put down the pen
  - Ask the Turtle object to go forward 50

Drawing a T

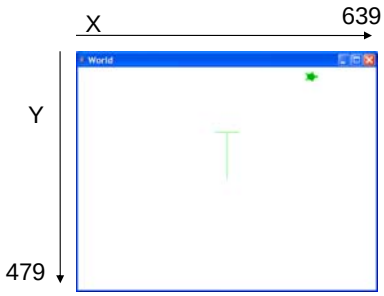

```
World world1 = new World();
Turtle turtle1 = new Turtle(world1);
turtle1.forward(100);
turtle1.penUp();
turtle1.turnLeft();
turtle1.forward(25);
turtle1.turn(180);
turtle1.penDown();
turtle1.forward(50);
```



Moving to a location

- A turtle can move to a particular location:

```
turtle1.penUp();
turtle1.moveTo(500,20);
```



Setting attributes

- An object method can work with the **properties** (**attributes**) of the object on which it was invoked
- Example: there are methods to **set** a turtle's width, height, name, etc.

```
turtle1.setWidth(50);
turtle1.setHeight(30);
turtle1.setName("Tiny");
```

Getting attributes

- There are methods to **get** a turtle's width, height, etc.

```
int width = turtle1.getWidth();
int height = turtle1.getHeight();
System.out.println("This turtle is " + width +
                   "x" + height);
```

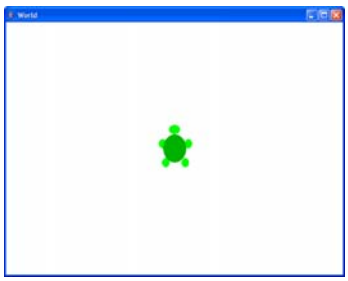
- These methods produce a **result**
- This value is **returned** by the method to wherever the method was invoked
- Another way to print the turtle's size:

```
System.out.println("This turtle is " + turtle1.getWidth()
                   +"x" + turtle1.getHeight());
```

Checking and changing size

- Tripling a turtle's size

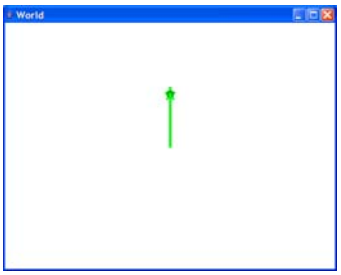
```
int width = turtle1.getWidth();
int height = turtle1.getHeight();
turtle1.setWidth(width * 3);
turtle1.setHeight(height * 3);
```



Changing pen width

- You can change the width of the trail:

```
World world1 = new World();
Turtle turtle1 = new Turtle(world1);
turtle1.setPenWidth(5);
turtle1.forward(100);
```



Changing pen color

- You can set the color of the pen:

```
turtle1.setPenColor(java.awt.Color.RED);
```

- There are predefined colors you can use:

```
java.awt.Color.RED
```

- Classes defined as part of the Java language are documented in the Java Application Programming Interface (Java API) at http://java.sun.com/j2se/1.5.0/docs/api
  - Find the package java.awt
    - A package is a group of related Java classes
  - Find the class Color

Using colors

- It is much easier to specify colors by using the **import statement**

  import java.awt.Color;
- Then you can just use the class name Color without needing the name of the package java.awt as well
- Example:

  turtle1.setPenColor(Color.RED);
- In a Java program, import statements go at the very beginning of the source file

Misc. coloring methods

- You can change the turtle color:

  turtle1.setColor(Color.BLUE);
- You can change the turtle's body color:

  turtle1.setBodyColor(Color.CYAN);
- You can change the turtle's shell color:

  turtle1.setShellColor(Color.RED);

- These **set** methods have corresponding **get** methods to retrieve colors too

Other things to do

- You can have a turtle hide and then later show itself by using

```
turtle1.hide();
turtle1.show();
```

- You can get a turtle's position by using

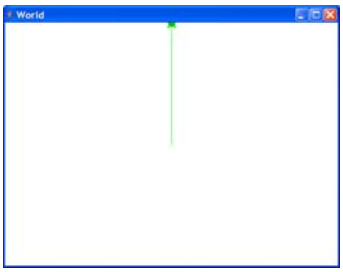
```
int xPos = turtle1.getXPos();
int yPos = turtle1.getYPos();
System.out.println("This turtle is at " + xPos + "," + yPos);
```

Objects control their state

- In our turtles world, for example, turtles won't move out of the boundaries of the world
- Try:

```
World world2 = new World();
Turtle turtle2 = new Turtle(world2);
turtle2.forward(600);
```

- Note that the turtle stopped at the edge of the screen and did not go any further



45

Writing your own object methods

How to customize code to make it do what YOU want

Creating methods

- We are not restricted to just using methods provided by Java or by other programmers
- We can write our own methods (so cool!)
- Remember:
  - A method is a collection of Java statements that performs some task
  - A method must be defined within a class

Defining a method

- The syntax for ***defining a method*** is

```
visibility returnType name(parameterList)
  {
    body of method (statements)
  }
```

  - Visibility, returnType and name… lets look into these a little closer!: determines access to the method
    - Usually "public" (all access) or "private" (just within this class)
  - returnType: is the type of thing returned
    - If nothing is returned, use the keyword "void"
  - name: start with a lowercase word and uppercasing the first letter of each additional word

Visibility

- Determines who (and what) can use your method
- In other words, determines **access** to the method
- Usually its either "public" (anywhere in the program can use it) or "private" only other methods in that class can use it

Visibility – Lets practice

Class Cell

Pubilc void grow(){}

Private void eat(){}

Class Body

Public void move(){}

Public void standUp(){}

Private void talk(){}

Can the body tell the cell to grow? To eat?

Can the cell tell the body to talk? To move?

Return Type

- Here we say the type of what will be "thrown" back at us
- For instance, if you ask the turtle what size its pen is, it will return an int to → the size of the pen is an integer, so the type of the thing it returns is "int"
- **If nothing is being returned, the return type is void**

Name

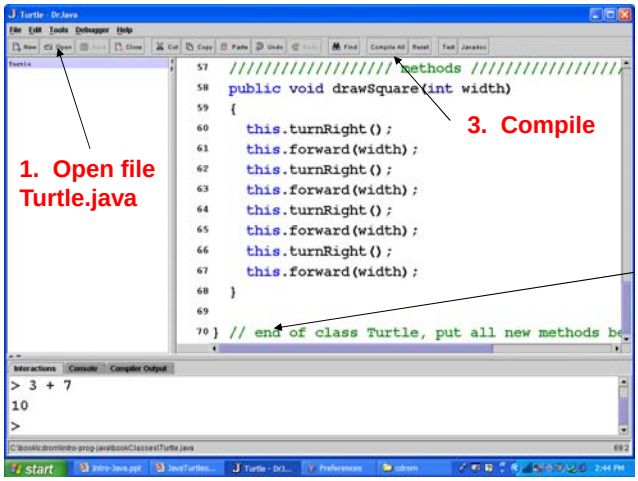
- This is the name of the method you are using, what you want to call it
- For example, if you are writing a method for a turtle to go **forward**, you might name it "**moveForward**", but you probably **wouldn't** name it "watermellon"
- YOU get to pick the name, it is NOT a keyword
- There are some rules though: start with a lowercase word and uppercasing the first letter of each additional word

Example: drawing a square


```
public void drawSquare()
 {
   this.turnRight();
   this.forward(30);
   this.turnRight();
   this.forward(30);
   this.turnRight();
   this.forward(30);
   this.turnRight();
   this.forward(30);
 }
```

- The visibility is public
- void: this method doesn't return a value
- The method name is drawSquare
- There are no parameters
- Notice that the parentheses are still required
- The keyword "this" refers to the object this method is invoked on

Adding a method to a class



Trying the method

- Compiling resets the Interactions pane, so you will need to create a world and turtle again:

```
World world1 = new World();
Turtle turtle1 = new Turtle(world1);
turtle1.forward(50);
turtle1.drawSquare();
turtle1.turn(30);
turtle1.drawSquare();
```

  - This has the turtle draw two squares using the new drawSquare() method we added to the Turtle class
  - What if we want to draw a square that is not 30 by 30?

Adding a parameter


```
public void drawSquare(int width)
 {
   this.turnRight();
   this.forward(width);
   this.turnRight();
   this.forward(width);
   this.turnRight();
   this.forward(width);
   this.turnRight();
   this.forward(width);
 }
```

- **Defining a parameter list**
  - specifies the values passed in to the method
  - for each parameter, give its type and the variable name used for it within the method
- There is only one parameter for this method
  - Its type is int
  - Its name is width

Trying the better drawSquare

- Type the following in the Interactions pane:

```
World world1 = new World();
Turtle turtle1 = new Turtle(world1);
turtle1.forward(50);
turtle1.drawSquare(30);
turtle1.turn(30);
turtle1.drawSquare(50);
```

- What values are **passed to** the drawSquare method here?
- When we invoke a method, the parameters passed to the method are called **actual parameters**

How does it work?

- What happens when you ask turtle1 to drawSquare(30) by

  turtle1.drawSquare(30);

- Java will check the Turtle class to see if it has a method drawSquare that has an int parameter
  - The actual parameter 30 will be copied to the parameter variable width
  - Java will start executing the code in drawSquare
  - The this in the method's code refers to turtle1 (the object the method was invoked on)

How does it work?

- Now consider

  turtle1.drawSquare(50);

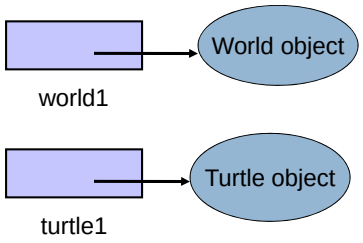
  When the drawSquare method is executed,
  - What will be the value of the parameter width ?
  - What will this refer to?
- Now add this to the Interactions pane:

  Turtle turtle2 = new Turtle(world1);
  turtle2.drawSquare(40);

  When the drawSquare method is executed,
  - What will be the value of the parameter width ?
  - What will this refer to?

Tracing with pictures

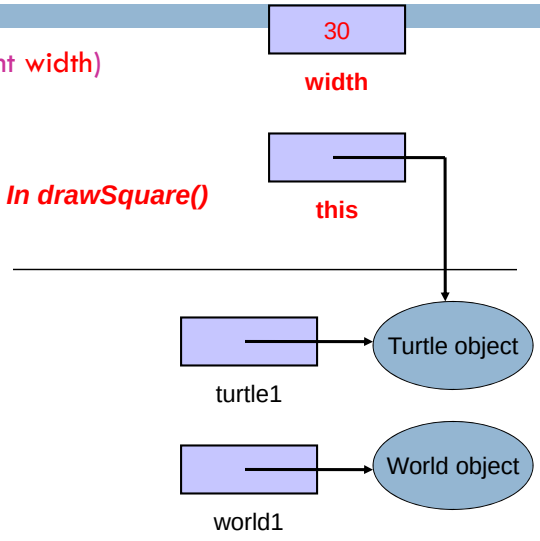

```
World world1 = new World();
Turtle turtle1 = new Turtle(world1);
```




```
turtle1.drawSquare(30);
```

In the drawSquare method


```
public void drawSquare(int width)
 {
   this.turnRight();
   this.forward(width);
   this.turnRight();
   this.forward(width);
   this.turnRight();
   this.forward(width);
   this.turnRight();
   this.forward(width);
 }
```



Comments in Java Code

- To make code more easily understandable, Java allows you to put ***comments*** in your code
  - Comments are ignored by the Java compiler
  - But are useful to people reading Java code
- Commenting code is good programming practice!
- Java allows commenting in two ways:

```
/* Everything between these symbols
 is a comment */
// Everything on the line following the double
// slash is a comment
```

Example


```
/* Method to draw a square of a specified width:
 * turtle starts at top left corner of square
 * and ends where it started, facing the same way */
public void drawSquare(int width)  {
    this.turnRight();
    this.forward(width);
    this.turnRight();
    this.forward(width);
    this.turnRight();
    this.forward(width);
    this.turnRight();
    this.forward(width);
 }
```

Sample problems

- Create a method for drawing a rectangle
  - Pass in the width and height
- Create a method for drawing an equilateral triangle
  - All sides have the same length
  - Pass in the length
- Create a method for drawing a diamond
- Create a method for drawing a house
  - Using the other methods

Class methods

Send a message to a class, you do NOT need an object!

Class methods in Java

- **Class method** (**static method**)
  - Can be thought of as a **message** sent to a **class**, to do something
  - Does **not** pertain to a particular object belonging to the class

Calling class methods

- Recall that object methods must be executed on an object, using the syntax

  objectReference.methodName()

- Class methods are invoked by giving the class name, using the syntax

  ClassName.methodName()

Example

- Java has a predefined class called Math
  - Find the Java API documentation for the class Math at http://java.sun.com/j2se/1.5.0/docs/api
  - It contains methods for performing basic numeric operations such as square root, trigonometric functions, rounding, etc.
- Math.round(2.95)
  - The round method rounds a floating point number to the nearest integer
  - It takes a float **parameter** and **returns** the int value that is the closest integer

Return values

- If a method produces a result, this result is **returned** by the method to wherever the method was invoked
  - Example: the sqrt method of class Math returns the positive square root of a double value
  - Example: what will be printed by System.out.println(Math.sqrt(9.0));
  - We can think of the result as "replacing" the invocation of the method

Sample problem

- For the class Math, find the documentation for the methods min, max, random
- Try the following method calls in the Interactions pane:

```
Math.max(10, 100)
Math.min(10,100)
Math.random()
```

Class methods vs. object methods

- In the Java API documentation, how can you tell which are class methods and which are object methods?
  - Look for the keyword **static** on the method
    - If it has the keyword static, then it is a class method
      - No object needs to be created in order to call the method
    - If there is no keyword static, then it is an object method
      - You need an object before you can invoke it

The main method

- In Java there is a special method called the ***main method***
  - Every program that you want to run must have a `main` method
  - Execution of a Java program always starts with the main method
  - The main method must be defined within some class

The main method

- The ***main method definition*** starts with
  public static void main(String[] args)
    - static because it is not invoked on any object
    - void because it returns nothing
    - (It takes one String array parameter which we will not use)

The main method

- Woah woah woah… What is this:
  - String[] args

Remember how we can send a parameter? The main method also gets a parameter!

It gets some string parameters. The square brackets "[]" are a signal to the program that it is getting an array of strings, not just one. We will learn what an array is in the **next lecture**.

You should always include this in your main method!

The main method

- Recall that in DrJava
  - Interactions pane: used to try out (practice) individual expressions and statements
  - Definitions pane: used to type in a complete Java program that will be compiled and run
    - There must be a main method in one of the classes
    - Execution starts at the beginning of the main method

Examples

- You have actually used main methods in the programs you typed in, in the labs
  - Lab 1: programs TryDrJava and MyName
  - Lab 2: program CentigradeToFahrenheit

Summary: Java Concepts

- Creating objects
- Declaring reference variables
- Object methods
- Invoking methods
- Passing parameters
- Writing methods
- Comments
- Class methods
- The main method

Key Notes

- Make sure you know how to:
  - Make a new object (slide 8)
  - Call a method on an object (slide 22)
  - Define a method (slide 45)
- Object methods must be in the object class
- Static methods do not need to be called on an object