Lecture 2
Calling and Defining Methods in Java

Outline
• Calling methods
• Declaring and defining a class
• Instances of a class
• Defining methods
• The `this` keyword

Meet samBot (kudos to former head TA Sam Squires)

• `samBot` is a robot who lives in a 2D grid world
• She knows how to do two things:
  - move forward any number of steps
  - turn right 90°
• We will learn how to communicate with `samBot` using Java

samBot's World

• This is `samBot`'s world
• `samBot` starts in the square at (0,0)
• She wants to get to the square at (1,1)
• Thick black lines are walls that `samBot` can't pass through

Giving Instructions (1/3)

• Goal: move `samBot` from her starting position to her destination by giving her a list of instructions
• `samBot` only knows how to "move forward n steps" and "turn right"
• What instructions should be given?
Giving Instructions (2/3)

- "Move forward 4 steps."
- "Turn right."
- "Move forward 1 step."
- "Turn right."
- "Move forward 3 steps."

Note: samBot moves in the direction her outstretched arm is pointing; yes, she can move sideways and upside down.

Giving Instructions (3/3)

- Instructions have to be given in a language samBot knows.
- That's where Java comes in!
- In Java, give instructions to an object by giving commands.

“Calling Methods”: Giving Commands in Java (1/2)

- samBot can only handle commands that she knows how to respond to.
- These responses are called methods.
- "method" is short for "method for responding to a command." Objects cooperate by giving each other commands.
  - object giving command is the caller.
  - object receiving commands is the receiver.

“Calling Methods”: Giving Commands in Java (2/2)

- samBot already has one method for "move forward 4 steps" and another method for "turn right."
- When we send a command to samBot to "move forward" or "turn right" in Java, we are calling a method on samBot.

Moving samBot forward

- Remember: when telling samBot to move forward, need to tell her how many steps to move.
- samBot’s "move forward" method is named `moveForward`.
- To call this method in Java:
  ```java
  samBot.moveForward(<number of steps>);
  ```

  - This means that if we want her to move forward 2 steps, say:
  ```java
  samBot.moveForward(2);
  ```

Turning samBot right

- samBot’s "turn right" method is called `turnRight`.
- To call the `turnRight` method on samBot:
  ```java
  samBot.turnRight();
  ```

  - To call methods on samBot in Java, need to address her by name.
  - Every command to samBot takes the form:
    ```java
    samBot.<method name>(<parameters>);
    ```

  - What are those parentheses at the end of the method for?
Calling Methods: Important Points

- Method calls in Java have parentheses after the method's name.
- Extra pieces of information passed to a method are called parameters; the actual values passed in are called arguments.
  - e.g.: in defining f(x), x is the parameter; in using f(2), 2 is the argument.
  - More on this next lecture!
- If the method needs any information, include it between parentheses (e.g., samBot.moveForward(2));
- If no extra information is needed, just leave the parentheses empty (e.g., samBot.turnRight());

Guiding samBot in Java

- Tell samBot to move forward 4 steps — samBot.moveForward(4);
- Tell samBot to turn right — samBot.turnRight();
- Tell samBot to move forward 1 step — samBot.moveForward(1);
- Tell samBot to turn right — samBot.turnRight();
- Tell samBot to move forward 3 steps — samBot.moveForward(3);

Hand Simulation of This Code

samBot.moveForward(4);
samBot.turnRight();
samBot.moveForward(1);
samBot.turnRight();
samBot.moveForward(3);

Clicker Question

Where will samBot end up when this code is

```
samBot.moveForward(3);
samBot.turnRight();
samBot.turnRight();
samBot.moveForward(1);
```

Putting Code Fragment in a Real Program (1/2)

```java
public class RobotMover {
    /* additional code elided */
    public void moveRobot(Robot samBot) {
        samBot.moveForward(4);
        samBot.turnRight();
        samBot.moveForward(1);
        samBot.turnRight();
        samBot.moveForward(3);
    }
}
```

Putting Code Fragments in a Real Program (2/2)

```java
public class RobotMover {
    /* additional code elided */
    public void moveRobot(Robot samBot) {
        samBot.moveForward(4);
        samBot.turnRight();
        samBot.moveForward(1);
        samBot.turnRight();
        samBot.moveForward(3);
    }
}
```
What is a class?
- A class is a blueprint for a certain type of object
- An object's class defines its properties and capabilities (methods)
- More on this in a few slides!

So far, we've been working within the class RobotMover
We need to tell Java about our RobotMover

Declaring and Defining a Class (1/3)
- As with dictionary entry, first declare term, then provide definition
- First line declares RobotMover class
- Breaking it down:
  - public indicates that anyone can use this class
  - class indicates to Java that we're about to define a new class
  - RobotMover is the name that we have chosen for our class

Note: public and class are Java "reserved words" aka "keywords" with pre-defined meanings in Java; we'll be using Java keywords a lot in the future

Declaring and Defining a Class (2/3)
- Class definition (aka "body") defines properties and capabilities of class
  - It is contained within curly braces that follow the class declaration
  - A class's capabilities ("what it knows how to do") are defined by its methods
  - RobotMover class only knows the very specific novelRobot method
  - A class's properties are defined by its instance variables—more on this next week!

Declaring and Defining a Class (3/3)
- General form for a class:
  ```java
  public class RobotMover {
      // additional code elided
      public void moveForward() {
          // code that moves robot forward
      }
      public void turnRight() {
          // code that turns robot right
      }
  }
  ```
- Each class goes in its own file, where name of file matches name of class
  - RobotMover class is contained in file "RobotMover.java"

Methods of the TA's Robot class
- ```java
  public class Robot {
      public void moveForward() {
          // code that moves robot forward
      }
      public void turnRight() {
          // code that turns robot right
      }
  }
  ```
- More on void later!
- Since moveForward needs to know how many steps to move, public int numberOfSteps as first parameter
  - Java lets you pass in parameters to methods
  - "moveForward takes a single parameter called numberOfSteps (type integer)"

The Robot class (defined by the TAs)
- Note: Normally, support code is in a "black box" that you can't examine
  ```java
  public class Robot {
      public void moveForward() {
          // code that moves robot forward
      }
      public void turnRight() {
          // code that turns robot right
      }
  }
  ```
- A class's properties are defined by its instance variables—more on this next week!
Classes and Instances (1/4)
- We've been saying `samBot` is a Robot
- We'll now refer to her as an instance of class `Robot`.
- All Robot's (all instances of the class `Robot`) have the exact same capabilities: the methods defined in the `Robot` class.
- All Robots also share the same properties (i.e. every Robot has a `Color` and a `Size`).

Classes and Instances (2/4)
- We can use the `Robot` class to build actual Robots - instances of the class `Robot`, whose properties may vary (next lecture).

Classes and Instances (3/4)
- The Robot class is like a blueprint.

Classes and Instances (4/4)
- Method calls are done on instances of the class.

Clicker Question
- You know that `blueBot` and `pinkBot` are instances of the same class. Let's say that the call `pinkBot.chaChaSlide()` makes `pinkBot` do the cha-cha slide. Which of the following is true?
  - A. The call `blueBot.chaChaSlide()` will make `blueBot` do the cha-cha slide.
  - B. The call `blueBot.chaChaSlide()` might make `blueBot` do the cha-cha slide or another popular line dance instead.
  - C. You have no guarantee that `blueBot` has the method `chaChaSlide()`.

Defining Methods
- Now that we know about calling methods, classes and instances, let's talk about defining methods.
- To understand this, let's use a variation of our previous example.
Declaring vs. Defining Methods

• Declaring a method says the class knows how to do some task.

• Defining a method actually explains how the class completes this task (what command it gives).

• Usually you will need to both define and declare your methods.

Defining a Method (1/2)

public class Robot {
   public void moveForward(int numberOfSteps) {
   // code that moves robot forward
   }
   public void turnRight() {
   // code that turns robot right
   }
   public void turnLeft() {
   // code that turns robot left
   }
}

Defining a Method (2/2)

public class Robot {
   public void moveForward(int numberOfSteps) {
   // code that moves robot forward
   }
   public void turnRight() {
   // code that turns robot right
   }
   public void turnLeft() {
   // code that turns robot left
   }
   public void moveForward(int numberOfSteps) {
   // code that moves robot forward
   }
   public void turnRight() {
   // code that turns robot right
   }
   public void turnLeft() {
   // code that turns robot left
   }
}

A variation

public class Room {
   public void enter() {
   // code to enter the room
   }
   public void leave() {
   // code to leave the room
   }
}

public class Robot {
   public void moveForward(int numberOfSteps) {
   // code that moves robot forward
   }
   public void turnRight() {
   // code that turns robot right
   }
   public void turnLeft() {
   // code that turns robot left
   }
   public void moveForward(int numberOfSteps) {
   // code that moves robot forward
   }
   public void turnRight() {
   // code that turns robot right
   }
   public void turnLeft() {
   // code that turns robot left
   }
}

The this keyword (1/2)

public class Robot {
   public void moveForward(int numberOfSteps) {
   // code that moves robot forward
   }
   public void turnRight() {
   // code that turns robot right
   }
   public void turnLeft() {
   // code that turns robot left
   }
   public void moveForward(int numberOfSteps) {
   // code that moves robot forward
   }
   public void turnRight() {
   // code that turns robot right
   }
   public void turnLeft() {
   // code that turns robot left
   }
}

A variation

public class Robot {
   public void moveForward(int numberOfSteps) {
   // code that moves robot forward
   }
   public void turnRight() {
   // code that turns robot right
   }
   public void turnLeft() {
   // code that turns robot left
   }
   public void moveForward(int numberOfSteps) {
   // code that moves robot forward
   }
   public void turnRight() {
   // code that turns robot right
   }
   public void turnLeft() {
   // code that turns robot left
   }
}

The this keyword (2/2)

public class Robot {
   public void moveForward(int numberOfSteps) {
   // code that moves robot forward
   }
   public void turnRight() {
   // code that turns robot right
   }
   public void turnLeft() {
   // code that turns robot left
   }
   public void moveForward(int numberOfSteps) {
   // code that moves robot forward
   }
   public void turnRight() {
   // code that turns robot right
   }
   public void turnLeft() {
   // code that turns robot left
   }
}

• Lots of code for a simple problem...

• When calling turnRight or moveForward on an instance of the Robot class, all code within the method's curly braces is executed.

• To call a method on an instance of a class, use the this keyword.

• The this keyword is used to refer to the current instance of the class.

• When working with RobotMove, we are asking to call this.moveForward().
public class Robot {
  public void turnLeft()
  public void moveForward(int numberOfSteps)
  public void turnRight()
}

• The this keyword is how an instance (like samBot) calls a method on itself
• Use this to call a method of Robot class from within another method of Robot class
• When samBot is told by, say, RobotMover to turnLeft, it responds by telling herself turnRight three times
• this.turnRight(); means "hey me, turn right!"
• this is optional, but CS15 expects

We're done!

• Have now seen our first method definition!
• Now that Robot has turnLeft, can call turnLeft() on any instance of Robot
• We've saved a lot of lines of code by using turnLeft!

Clicker Question

Give this method, what can we say about this.turnRight();?
A. Other objects cannot call the turnRight() method on instances of the Robot class
B. The current instance of the Robot class is calling turnRight() on another instance of Robot
C. The current instance of Robot class is calling the turnRight() method on itself
D. The call this.turnRight(); will not appear anywhere else in the Robot class definition

Summary

• The this keyword is how an instance (like samBot) calls a method on itself
• Use this to call a method of Robot class from within another method of Robot class
• When samBot is told by, say, RobotMover to turnLeft, it responds by telling herself turnRight three times
• this.turnRight(); means "hey me, turn right!"
• this is optional, but CS15 expects

Simplifying our code using turnLeft

• Have now seen our first method definition!
• Now that Robot has turnLeft, can call turnLeft() on any instance of Robot
• We've saved a lot of lines of code by using turnLeft!

turnAround

• The TAs could also define a method that turns the Robot around 180°
• See if you can declare and define the method turnAround

Clicker Question

Give this method, what can we say about this.turnRight();?
A. Other objects cannot call the turnRight() method on instances of the Robot class
B. The current instance of the Robot class is calling turnRight() on another instance of Robot
C. The current instance of Robot class is calling the turnRight() method on itself
D. The call this.turnRight(); will not appear anywhere else in the Robot class definition

The this keyword (2/2)

Class declaration

• public class Me {
  public void turnRight();
    // code that moves right
    // Class definition appears anywhere else in the file
  public void turnLeft();
    // code that moves left
    // Class declaration
  public void moveForward(int numberOfSteps);
    // code that moves forward
  public void turnRight();
    // code that turns right
  public void turnLeft();
    // code that turns left
}

Clicker Question

Give this method, what can we say about this.turnRight();?
A. Other objects cannot call the turnRight() method on instances of the Robot class
B. The current instance of the Robot class is calling turnRight() on another instance of Robot
C. The current instance of Robot class is calling the turnRight() method on itself
D. The call this.turnRight(); will not appear anywhere else in the Robot class definition

We're done!

• Have now seen our first method definition!
• Now that Robot has turnLeft, can call turnLeft() on any instance of Robot
• We've saved a lot of lines of code by using turnLeft!
Now that the `Robot` class has the method `turnAround`, we can call the method on any `Robot`

There are other ways of implementing this method that are just as correct

• Instead of calling `turnRight`, could call our newly created method, `turnLeft`

• Both of these solutions are equally correct in that they will turn the robot around 180°

• How do they differ? When we try each of these implementations with a `Robot`, what do we see in each case?

### Summary (1/2)

- Classes
  - a class is a blueprint for a certain type of object
- Instances
  - an instance of a class is a particular member of that class, on which we can call methods.
  - example: `samBot` is an instance of `Robot`

### Summary (2/2)

- Calling methods
  - an object sends a message to another object
  
  - general form: `instance.method (parameters)`
  
  - example: `samBot.turnRight()`

- Defining methods
  - how we describe a capability of a class
  
  - general form: `visibility (type) <name> (parameters)`
  
  - example: `public void turnLeft() { … }`

- The `this` keyword
  - how an instance calls a method on itself
  
  - example: `this.turnRight()`

### Announcements (1/2)

- Hack@Brown has 2 information sessions
  - Tuesday, 9/13 at 8pm in BERT 130 and Wednesday, 9/14 at 6:30pm in RISD’s Graphic Design Gallery
  - Learn more about different teams that help organize Hack@Brown
  - Outreach, Design, Dev, Sponsorship, Food & Logistics, Hardware, Media, Workshops, Experience

- Hours start today at 4:30 – see website for full schedule

- Labs start today
  - you should have a lab section by now – if not, email the Head TA ASAP
  - if you try to attend a lab section you aren’t signed up for, you will not get checked in

### Announcements (2/2)

- If you haven’t created a REEF account, please do so by Thursday before class. Instructions were sent out in an email

- For the best email response time: email the cs015ta's next best email cs015headta ASAP
  - slow response email an individual TA – don’t do it!