Hack@Brown 2020

Come to our info session: https://tinyurl.com/info2020

And apply here: https://tinyurl.com/hackatbrown2020
Due 9/13

WiCS
Women In Computer Science
Dedicated to improving diversity and inclusion across gender identity in CS.

Join our listserv to get updates and hear opportunities!
Email: wics@lists.cs.brown.edu
Add yourself to the listserv: https://tinyurl.com/brownwics

WELCOME BACK EVENT ft. Kabob & Curry
Thursday, 9/12/19
CIT 368

CS15 Mixer
• This Wednesday, September 11 in CIT 3rd floor atrium, from 5-6PM
• Get to know the 15 TA staff!
• Drinks and light snacks will be provided
Note Taking for CS15

- Slides are always uploaded to the website before lectures!
- Physical copies
  - print out the “Printable PDF” version of the slides before lecture in one of Brown’s printing centers and take notes while Andy is speaking!
  - printing center locations can be found here!
- Live note-taking
  - If you download the Power Point version of Andy’s slides, you can take notes in the lower part of the screen.

How to Install TopHat

- Computer: Go to https://tophat.com → Click Signup in upper right corner → select Student → join with course code or Search by School → input info under Account → enter your Banner ID under Grading → Add your phone number to submit responses in class via text under Phone
- IOS/Android: Download Top Hat Lecture App → click Create Student Account and follow instructions to complete
- Link with Detailed Instructions: https://tinyurl.com/y6ythebb
- CS15 Course Code: 783865

Review

- We model the “application world” as a system of collaborating objects
- Objects collaborate by sending each other messages
- Objects have properties and behaviors (things they know how to do)
- Objects are typically composed of component objects
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:
  o move forward any number of steps
  o 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 samBot can't pass through

Giving Instructions (1/3)

- **Goal**: move samBot from starting position to 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 in this 2D world!
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 it commands

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

- samBot can only handle commands she knows how to respond to
- These responses are called methods!
  - “method” is short for “method for responding to a command”. Therefore, whenever samBot gets a command, she can respond by utilizing a method.
- Objects cooperate by giving each other commands
  - caller is the object giving the command
  - receiver is the object receiving the command

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

- samBot already has one method for “move forward n 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.
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(…)>
  ```
- What are those parentheses at the end of the method for?

Moving samBot forward

- Remember: when telling `samBot` to move forward, you 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, we say:
  ```java
  samBot.moveForward(2);
  ```

Calling Methods: Important Points

- Method calls in Java have parentheses after the method’s name
- In the definition of the method, extra pieces of information to be passed into the method are called parameters; in the call to the method, the actual values passed in are called arguments
  - e.g. in `defining f(x)`, x is the parameter; in `calling f(2)`, 2 is the argument
  - more on parameters and arguments next lecture!
- If the method needs any information, include it between the 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

- Simulating lines of code by hand checks that each line produces correct action
  - we did this in slide 7 for pseudocode
- In hand simulation, you play the role of the computer
  - lines of code are “instructions” for the computer
  - try to follow “instructions” and see if you get desired result
  - if result is incorrect:
    - one or more instructions or the order of instructions may be incorrect

Hand Simulation of This Code

```java
samBot.moveForward(4);
samBot.turnRight();
samBot.moveForward(1);
samBot.turnRight();
samBot.moveForward(3);
```
About TopHat Questions

- Increase engagement during lecture!
- We encourage working with a neighbor and discussing concepts on all TopHat questions
- Can use app, website

• If you need an device to access TopHat, you can borrow a laptop from the IT Service Center on 5th floor of Page-Robinson Hall.

TopHat Question

Where will samBot end up when this code is executed?

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

Choose one of the positions or E: None of the above

```
          0  1  2  3  4
   0  A   B   C   D
   1   H   E   F   G
   2   I   J   K   L
   3   N   M   L   K
   4   O   P   Q   R
```

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

- Let’s demonstrate this code for real
- First, put it inside real Java program
- Grayed-out code specifies context in which an arbitrary robot named myRobot executes instructions
  - It is part of the stencil code written for you by the TAs, which also includes samBot’s or any other robot’s capability to respond to moveForward and turnRight - more on this later

```java
public class RobotMover {
    /* additional stencil code elided*/
    public void moveRobot(Robot myRobot) {
        myRobot.moveForward(4);
        myRobot.turnRight();
        myRobot.moveForward(1);
        myRobot.turnRight();
        myRobot.moveForward(3);
    }
}
```
Before, we’ve talked about objects that handle messages with “methods”

Introducing a new concept…

**classes**!

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

We’re about to explain this part of the code!

### 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!
- Let’s embed the `moveRobot` code fragment (method) that moves `samBot` (or any other `Robot` instance) in a new class called `RobotMover`
- Need to tell Java compiler about `RobotMover` before we can use it

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

### Declaring and Defining a Class (1/3)

- Like a dictionary entry, first **declare** term, then provide **definition**
- First line **declares** `RobotMover` class
- Breaking it down:
  - **public** indicates any other object can use instances of this class
  - **class** indicates to Java compiler that we are about to define a new class
  - `RobotMover` is the name we have chosen for our class

Note: **public** and **class** are Java “reserved words” aka “keywords” and have pre-defined meanings in Java, use 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 thus far only shows one specific method, moveRobot
  - A method is a declaration followed by its body (also enclosed in {...} braces)
- A class's properties are defined by its *instance variables* — more on this next week

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

Declaring and Defining a Class (3/3)

- General form for a class:

  `<visibility> class <name> {
      <code (properties and capabilities) that defines class>
  }

• To make code more compact, typically put opening brace on same line as declaration — Java compiler doesn’t care
• Each class goes in its own file, where name of file matches name of class
  - RobotMover class is contained in file "RobotMover.java"

The Robot class (defined by the TAs)

```java
public class Robot {
    // code that moves and turns
    public void moveForward(int numberOfSteps) {
        // code that moves forward
    }
    public void turnRight() {
        // code that turns right
    }
    /* other code elided — if you're curious, check out Robot.java in the stencil code! */
}
```

- **public class Robot declares a class called Robot**
- Information about the properties and capabilities of Robots (the class definition) goes within the red curly braces

Note: Normally, support code is a "black box" that you can’t examine
Methods of the TA's \textit{Robot} class

\begin{verbatim}
public class Robot {
  public void turnRight() {
    // code that turns robot right
  }

  public void moveForward(int numberOfSteps) {
    // code that moves robot forward
  }

  /* other code elided -- if you're curious, check out Robot.java in the stencil code! */
}
\end{verbatim}

Classes and Instances (1/4)

\begin{itemize}
  \item \texttt{samBot} is an instance of class \texttt{Robot}
    \begin{itemize}
    \item this means \texttt{samBot} is a particular \texttt{Robot} that was built using the \texttt{Robot} class as a blueprint (another instance could be \texttt{dwightBot})
    \end{itemize}
  \item All \texttt{Robots} (all instances of the class \texttt{Robot}) have the \textit{exact same capabilities}: the methods defined in the \texttt{Robot} class. What one \texttt{Robot} instance can do, they all can do since they are made with the same blueprint!
  \item All \texttt{Robots} also have the \textit{exact same properties} (i.e., every \texttt{Robot} has a \texttt{Color} and a \texttt{Size})
    \begin{itemize}
    \item they all have these properties but the values of these properties may differ between instances (e.g., a big \texttt{samBot} and small \texttt{dwightBot})
    \end{itemize}
\end{itemize}

Classes and Instances (2/4)

The \texttt{Robot} class is like a blueprint
Classes and Instances (3/4)

We can use the Robot class to build actual Robot instances of the class Robot, whose properties (like their color in this case) may vary (next lecture).

Classes and Instances (4/4)

Method calls are done on instances of the class. These are four instances of the same class (blueprint).

TopHat 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

- We have already learned about defining classes, let's now talk about defining methods.
- Let's use a variation of our previous example:

```java
public class RobotMover {
    /* additional code elided */
    public void moveRobot(Robot myRobot) {
        // Your code goes here!
    }
}
```

Declaring vs. Defining Methods

- Declaring a method says the class knows how to do some task like `pinkBot can chaChaSlide()`
- Defining a method actually explains how the class completes this task (what command it gives); `chaChaSlide()` could include:
  stepping backwards, alternating feet, stepping forward
- Usually you will need to both define and declare your methods

A Variation on moveRobot (1/2)

```
public class RobotMover {
    /* additional code elided */
    public void moveRobot(Robot myRobot) {
        myRobot.turnRight();
        myRobot.moveForward(2);
        myRobot.turnRight();
        myRobot.turnRight();
        myRobot.turnRight();
        myRobot.moveForward(3);
        myRobot.turnRight();
        myRobot.turnRight();
        myRobot.turnRight();
        myRobot.moveForward(2);
        myRobot.turnRight();
        myRobot.turnRight();
        myRobot.turnRight();
        myRobot.moveForward(2);
    }
    ...
}
```
A Variation on moveRobot (2/2)

- Lots of code for a simple problem.
- samBot only knows how to turn right, so have to call turnRight three times to make her turn left.
- If she understood how to `turn left`, would be much less code!
- We can ask the TAs to modify samBot to turn left by declaring and defining a method called turnLeft.

```java
public class RobotMover {
    /* additional code elided */
    public void moveRobot(Robot myRobot) {
        myRobot.turnRight();
        myRobot.moveForward(2);
        myRobot.turnRight();
        myRobot.turnRight();
        myRobot.moveForward(2);
        myRobot.turnRight();
        myRobot.turnRight();
        myRobot.moveForward(3);
        myRobot.turnRight();
        myRobot.turnRight();
        myRobot.turnRight();
        myRobot.moveForward(2);
        myRobot.turnRight();
        myRobot.turnRight();
        myRobot.moveForward(2);
    }
}
```

Defining a Method (1/2)

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

- Almost all methods take on this general form:

```java
<visibility> <type> <name> (<parameters>) {
    <list of statements within method>
}
```

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

Defining a Method (2/2)

```java
public class Robot {
    public void turnRight() {
        // code that turns robot right
    }
    public void moveForward(int numberOfSteps) {
        // code that moves robot forward
    }
    public void turnLeft() {
        // TA's code goes here!!
        // Here you'll have the method definition!
    }
}
```

- We're going to define a new method: turnLeft.
- To make a Robot turn left, tell her to turn right three times.
The **this** keyword (1/2)

```java
public class Robot {
    public void turnRight() {
        // code that turns robot right
    }
    public void moveForward(int numberOfSteps) {
        // code that moves robot forward
    }
    public void turnLeft() {
        this.turnRight();
        this.turnRight();
        this.turnRight();
    }
}
```

- When working with `RobotMover`, we were talking to `samBot`, an instance of class `Robot`.
- To tell her to turn right, we said `"samBot.turnRight();"`.
- Why do the TAs now write `"this.turnRight();"`?

The **this** keyword (2/2)

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

We’re done!

```java
public class Robot {
    public void turnRight() {
        // code that turns robot right
    }
    public void moveForward(int numberOfSteps) {
        // code that moves robot forward
    }
    public void turnLeft() {
        this.turnRight();
        this.turnRight();
        this.turnRight();
    }
}
```

- Have now seen our first method definition!
- Now that `Robot` has `turnLeft`, can call `turnLeft` on any instance of `Robot`.
TopHat Question

Given 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 the `Robot` class is calling the `turnRight()` method on itself
D. The call `this.turnRight()` will not appear anywhere else in the `Robot`'s class definition

```
public class Robot {
    public void turnRight() {
        // code that turns robot right
    }
}
```

Summary

```
public class Robot {
    /* additional code elided */
    public void turnLeft() {
        this.turnRight();
        this.turnRight();
        this.turnRight();
    }
}
```

```
public class RobotMover {
    public void moveRobot(Robot myRobot) {
        myRobot.turnRight();
        myRobot.moveForward(2);
        myRobot.turnRight();
        myRobot.turnRight();
        myRobot.moveForward(3);
        myRobot.turnRight();
        myRobot.turnRight();
        myRobot.moveForward(2);
        myRobot.turnRight();
        myRobot.turnRight();
        myRobot.moveForward(2);
    }
}
```

We've saved a lot of lines of code by using `turnLeft`!
This is good! More lines of code makes your program harder to read and more difficult to debug and maintain.
The TAs could also define a method that turns the Robot around 180°.

See if you can declare and define the method `turnAround`:

```java
public class Robot {
    public void turnRight() {
        // code that turns robot right
    }
    public void moveForward(int numberOfSteps) {
        // code that moves robot forward
    }
    public void turnLeft() {
        this.turnRight();
        this.turnRight();
        this.turnRight();
    }
    // your code goes here!
    // ...
    // ...
    // ...
}
```

Now that the `Robot` class has the method `turnAround`, we can call the method on any instance of the class `Robot`.

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

```java
public class Robot {
    public void turnRight() {
        // code that turns robot right
    }
    public void moveForward(int numberOfSteps) {
        // code that moves robot forward
    }
    public void turnLeft() {
        this.turnRight();
        this.turnRight();
        this.turnRight();
    }
    public void turnAround() {
        this.turnRight();
        this.turnRight();
    }
    // your code goes here!
    // ...
    // ...
    // ...
}
```

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 `samBot`, what will we see in each case?
Summary (1/2)

- **Classes**
  - A class is a blueprint for a certain type of object.
  - Example: `Robot` is a class.

- **Instances**
  - An instance of a class is a particular member of that class whose methods we can call.
  - Example: `samBot` is an instance of `Robot`.

Summary (2/2)

- Calling methods
  - An instance can call on the methods defined by its class.
  - General form: `instance.method name(<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 within its class definition.
  - Example: `this.turnRight()`.

Announcements

- HW1 is out!
- Mixer tomorrow in CIT 3rd floor atrium, 5-6PM!
- Sign up on Piazza! Link on website!
- Sections start today: you should have a section by now – if not, email the Head TAs ASAP (cs0150headtas@lists.brown.edu).
  - If you try to attend a section you aren’t signed up for, you will not get checked off.
  - Find assigned room in the same link that you used to sign up.
- For the best email response time: email the TA listserv! (cs0150tas@lists.brown.edu)
  - Next best: email cs0150headtas.
  - Slow response: email individual TA – don’t do it!