Join our Email List:  
"WE STRIVE TO MAKE COMPUTER SCIENCE AT BROWN UNIVERSITY A MORE RECEPTIVE AND INCLUSIVE SPACE FOR UNDERREPRESENTED RACIAL MINORITIES."

Join our Slack Channel:  
Join our Email List:

Contact Us:  
mosaic-plus@brown.edu  
@mosaic_plus on Instagram

**MOSAIC+**

**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!
Add yourself to the listserv:  
Follow us at: @BrownUWiCS

**Class of 2025**

**WiCS Office**

**Hours:**
Tuesday, September 14
6:00-8:00PM

**LGBTQ+ Mentorship Program**

An initiative to build LGBTQ+ community in CS

**Mentorship**
1-on-1 or peer group mentoring for LGBTQ+ students - TGNC folks, femmes, and POC to especially welcome!

**Sign up now!**

**Community Events**
- LGBTQ+ Peer
- Connections
- Industry Panelists
- Game Nights
- + Submit your own ideas!

**More to Come**
Join our LGBTQ+ student group later this fall!

**form closes Sept 21st**

*we welcome closeted & questioning folks - participation is confidential!*

Sign up now!
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 and take notes while Andy is speaking!
  - if you’re on campus, you can find instructions on where to print 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.

Lecture Questions will be conducted through TopHat
- sign up here if you haven’t done so already

Synchronous Students:
- question will be released when a “TopHat Question” slide comes up
- approximately 1-minute window to answer the question
- we will collect results real-time and discuss the answers during lecture
- 5% of total grade and graded on completion
- drop the lowest 4 quiz scores

Asynchronous Students:
- encouraged to attempt the questions before proceeding
- 5% of total grade will be distributed accordingly to other portions of the grade

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

Review: Object Oriented Programming

- We model the “application world” as a system of collaborating objects
- In OOP, objects are “smart” in their specialty
  - have properties and behaviors (things they know how to do)
- Objects collaborate by sending each other messages
- Objects typically composed of component objects

Review: OOP as Modeling

- Write programs by modeling the problem as system of collaborating components
  - you determine what the building blocks are
  - put them together so they cooperate properly
  - like building with smart Legos, some of which are pre-defined, some of which you design!
  - containment diagrams, like the one shown here, are a great way to help model your program!
Example: Tetris (1/3)

- What are the game’s objects?
- What properties do they have?
- What do those objects know how to do?

Example: Tetris (2/3)

- What are the game’s objects?
  - piece, board
- Properties: What attributes and components do they have?
  - piece: orientation, position, shape, color, tiles
  - board: size, rows

Example: Tetris (3/3)

- Capabilities: What do those objects know how to do?
  - piece: be created, fall, rotate, stop at collision
  - board: be created, remove rows, check for end of game
Outline

- Calling methods
- Declaring and Defining a class
- Instances of a class
- Defining methods
- The this keyword

Meet samBot (kudos to former HTA 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 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?

![Diagram of a grid with a starting position marked and a destination marked by a star]

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

![Diagram showing the movement of *samBot* in a 2D grid]

9/14/21
“Calling Methods”: Giving Commands in Java (1/2)

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

Method names don’t have spaced Style guide has capitalization conventions e.g., “camelCase”
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.
  - As we did in slide 10 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

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

About TopHat Questions

- Increase engagement during lecture!
- Allow you to gauge your understanding of important concepts throughout lecture.
- Give you participation points for paying attention during lecture.

* TopHat questions are worth 5% of total grade for synchronous students! (See course syllabus.)
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

- A
- B
- C
- D
- E: None of the above

---

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`, a parameter of the `moveRobot` method, executes instructions
  - part of stencil code written for you by the TAs, which also includes any robot’s capability to respond to `moveForward` and `turnRight` - more on this later

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

---

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

- 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.moveForward(4);
        myRobot.turnRight();
        myRobot.moveForward(1);
        myRobot.turnRight();
        myRobot.moveForward(3);
    }
}
```
Outline

- Calling methods
- Declaring and defining a class
- Instances of a class
- Defining methods
- The this keyword

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 sailBot (or any other Robot) 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.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

---

Declaring and Defining a Class (3/3)

- **General form for a class:**
  ```java
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 **must match** name of class
  - RobotMover class is contained in file "RobotMover.java"

---

The Robot class (defined by the TAs)

- **public class Robot**
  ```java
  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*/
  }
  ```
- **public class Robot declares a class called Robot**
- Information about the properties and capabilities of Robot (the class definition) goes within the red curly braces
Methods of the TA's Robot class

```java
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 */
    }

    Note that when we call moveForward, we have to pass an argument of type int or the Java compiler will throw an error.
}
```

Outline

- Calling methods
- Declaring and defining a class
- Instances of a class
- Defining methods
- The this keyword

Classes and Instances (1/4)

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

Jeff, host of Survivor: Rhode Island
The Robot class is like a blueprint.

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

• 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(); might make blueBot do the cha-cha slide or another popular line dance instead
B. The call blueBot.chaChaSlide(); will make blueBot do the cha-cha slide
C. You have no guarantee that blueBot has the method chaChaSlide();

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

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 means the class knows how to do some task, like `pinkBot` can `chaChaSlide()`
- **Defining** a method actually explains how the class executes this task (i.e. what sequence of commands it specifies)
  - `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)

```java
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`
Defining a Method (1/2)

- Almost all methods take on this general form:
  public class Robot {
    // 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.

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

Defining a Method (2/2)

- We're going to define a new method `turnLeft`
- To make a `Robot` turn left, tell it to turn right three times

```
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();
  }
}
```

Outline

- Calling methods
- Declaring and defining a class
- Instances of a class
- Defining methods
- The `this` keyword
The **this** keyword (1/2)

- When working with the class `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();"`?

```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();
    }
}
```

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

- The **this** keyword allows an instance (like `samBot`) to call a method on itself (**this** is short for "this instance" or "defined in this method").
- 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.

```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();
    }
}
```

We're done!

- Now that `Robot` has `turnLeft()`, can call `turnLeft()` on any instance of `Robot`.
- We will see how we can use `turnLeft()` to simplify our code in a few slides.

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

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

Summary

```
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();
    }
}
```

Simplifying our code using `turnLeft`

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

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

- Lab 0 Linux and Terminal out today
  - If you did not sign up for section or have not received an email from your section TA, please email the HTAs
  - Review GitHub/IntelliJ setup before lab!

- Andy’s Kitchen out tomorrow!
  - Due Saturday 09/18
  - No Early or Late Hand-in

- RISD students email the HTAs after class so we can make sure we have your emails

- Newly registered RISD students come up to speak with Andy after class
“With great power comes great responsibility”

- Tech moves faster than policy → engineers often set ethical precedents
- Understanding the social implications of technology important to being an effective computer scientist
Framing

- Kranzberg's First Law: “Tech is neither good nor bad, nor is it neutral.” (1986)
- Goals:
  - provide a framework for considering impact of tech industry
  - re-evaluate how we define “progress”
  - consider new stakeholders and unintended consequences of tech

What SRC is not

- Excessive faith in tech
  - techno-solutionism: technology will solve any problem, including problems created by other technology
  - techno-chauvinism: technology is better at problem-solving than people
- Excessive hatred of tech/techno-pessimism: technology creates as many problems as it solves → we should decrease reliance on technology
- Telling you what to believe/presenting “the correct opinion”
- Guilting yourself or students about your internship/job
- “Be ruthless with systems, be kind with people.” (Michael Brooks)

What SRC is

- Engagement with the ethical implications of our work
- Critical, pragmatic techno-optimism
  - we can build a better world with technology, if we understand the systems the technology shapes and is shaped by
  - understand technology’s potential harms to build it better
  - we present case studies / information in the news
  - come to your own conclusions!
What is “tech for good?”

- Ben Green "Good Isn’t Good Enough" (2019)
  - got Best Paper Award at NeurIPS (the ML conference)
  - non-technical paper winning award—watershed event, wouldn’t have happened 10 yrs ago!
  - AML has focused on this longer than other fields because the work is likely to have disproportionate societal impact
  - “social good” can be interpreted differently by everyone
  - computer science lacks a good working definition; we need a more rigorous methodology
  - we hope to provide context / a more logical framework!

- SRC at Brown started in 2019! 3rd year
  - extremely new / bleeding edge field in computer science
  - you can help define this
  - SRC in 15: preview of topics you will learn in more depth in future courses

"Those who do not know history’s mistakes are doomed to repeat them."
- Life of Reason: The Phases of Human Progress Vol. I, George Santayana

- Tech too often claims to be outside of history/politics
- Many “revolutionary” / “disruptive” innovations presented as new, replicate old power dynamics / sociopolitical phenomena
- Well-intentioned technologists can cause immense harm because they are not aware of how what they build will be used
- We can do better than harm mitigation! Knowing tech’s history allows us to build truly great things

More reading that may be of interest!
- The Paradox of Socially Responsible Computing: The limits and potential of teaching tech ethics — Jessica Dai, The Indy (Brown publication!)
- Brown Socially Responsible Computing TA Program
- “Good” isn’t good enough — Ben Green, NeurIPS
- Disruption: A Manifesto — The Editors, Logic Magazine
- Reboot — About
- What Tech Calls Thinking — Adrian Daub, Logic x FSG Originals