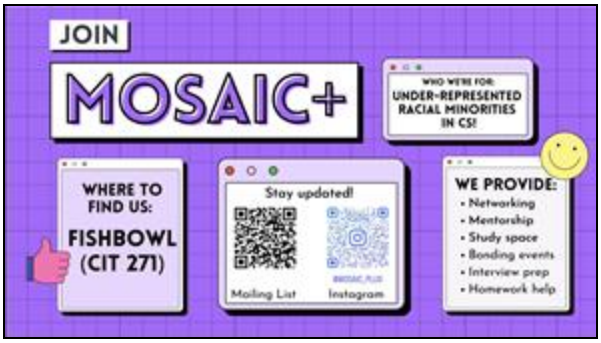
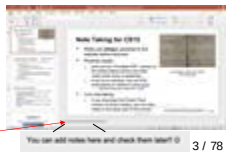


Note Taking for CS15

- 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 I'm speaking!
  - if you're on campus, you can find instructions on how to Live note-taking print here!
- If you download the PowerPoint version of slides, you can take notes in the lower part of the screen

Lucy Reyes' notes from CS15! (HTA in 2019)



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

Object Oriented Programming

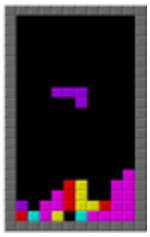
- Models the "application world" as system of collaborating objects
- In OOP, objects are "smart" in their specialty
  - have properties and behaviors (things they know how to do)
- Objects collaborate by sending each other messages
- Objects typically composed of other component objects

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/association 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
    - # of tiles
  - board:
    - size
    - rows
    - columns

Example: Tetris (3/3)

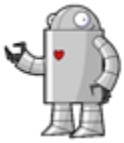
- Capabilities: What do those objects know how to do?
  - piece:
    - be created
    - fall
    - rotate
    - stop at collision
  - board:
    - be created
    - remove rows
    - check for end of game

Outline

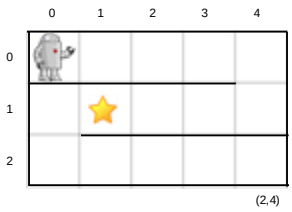
- Calling methods
- Declaring and defining a class
- Instances of a class
- Defining methods
- The `this` keyword

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 90º
- We will learn how to communicate with `samBot` using Java



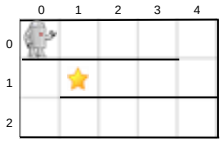
samBot's World



- 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)

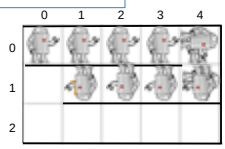
- **Goal**: move `samBot` from starting position to destination by giving her a list of instructions
- `samBot` only knows how to "move forward *n* steps" and "turn right"
- What instructions should be given?



Giving Instructions (2/3)

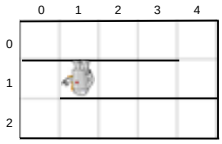
Note: samBot moves in the direction her outstretched arm is pointing.
Yes, she can move sideways and upside down in this 2D world!

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



Giving Instructions (3/3)

- Instructions must be given in a language `samBot` knows
- That's where Java comes in!
- In Java, give instructions to an object by **giving it commands**
  - we use "sending a message" and "giving a command" as synonyms!

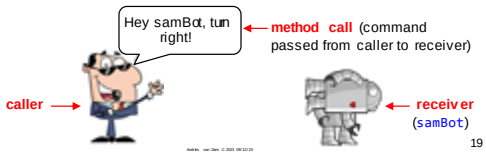


"Calling Methods": Giving Commands in Java (1/2)

- `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 must respond by utilizing a predefined 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

Method names don't have spaces! Our style guide has capitalization conventions, e.g., "camelCase"

- `samBot`'s "turn right" method is called `turnRight`
- To call methods on `samBot` in Java, you need to address her by name!
- To call the `turnRight` method on `samBot`:

```
samBot.turnRight();
```

- Every command to `samBot` takes the form:

```
samBot.<method name(…)>;
```

You can substitute any method inside < >!

; ends Java statement

- 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:

```
samBot.moveForward(<number of steps>);
```

- This means that if we want her to move forward 2 steps, we say:

```
samBot.moveForward(2);
```

Calling Methods: Important Points

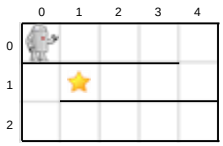
- Method calls in Java have parentheses after the method's name
- In the definition (body) of the method, extra pieces of information to be taken in by the method are called **parameters**; in the call to the method, the actual values taken in are called **arguments**
  - e.g., in defining f(x), x is 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, 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);`

"pseudocode"

An informal notation of English, math, and Java-like commands



Java code

Hand Simulation

- Simulating lines of code **by hand** checks that each line produces correct action
- 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

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

TopHat Question Logistics

Join Code: 553500

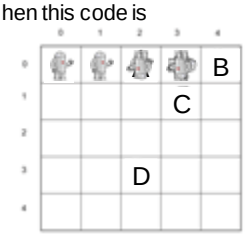
- To make lectures less passive, improve engagement, and
- To gauge how well you are following a lecture, stop lecture and let you answer simple questions through **TopHat**
  - sign up here if you haven't done so already
- Question will be released when a "TopHat Question" slide comes up
- Approximately 1-minute window to answer the question
- We will collect results real-time and discuss the answers during lecture
- 5% of total grade– another good reason to attend!
- Drop lowest 4 scores

TopHat Question

Join Code: 553500

Where will `samBot` end up when this code is executed?

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



Choose one of the positions or
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

```
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);
    }
}
```


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

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

- Before, we've talked about objects that handle messages with "methods"
- Introducing a new concept… **classes**!

```
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 corresponding 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`) in a new class called `RobotMover`
- Need to tell Java compiler about `RobotMover` before we can use it

```
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

declaration of the RobotMover class

```
public class RobotMover {

    /* additional code elided */

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

Note: public and class are Java "reserved words" aka "keywords" and have pre-defined meanings in Java; use Java keywords a lot

Declaring and Defining a Class (2/3)

- **Class definition** (aka "body") defines properties and capabilities of class
  - 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`
  - each method has a declaration followed by its definition (also enclosed in {…} braces)
- A class's properties are defined by its **instance variables** – more on this next week

```
public class RobotMover {

    /* additional code elided */

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

declaration of moveRobot method

definition of moveRobot method

definition of RobotMover class

Declaring and Defining a Class (3/3)

- **General form for a class:**

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

declaration

definition

```
public class RobotMover {
    /* additional code elided */
    public void moveRobot(Robot myRobot) {
        /* method body */
    }
}
```

- 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)

Note: Normally, support code is a "black box" that you can't examine

```
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!*/
}
```

in-line comment

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

Methods of the TA's `Robot` class

```
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!*/
}
```

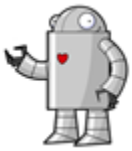
Note that when we call `moveForward`, we have to pass an argument of type `int` or the Java compiler will throw an error

- `public void turnRight()` and `public void moveForward(int numberOfSteps)` each **declare a method**
  - more on `void` later!
- `moveForward` needs to know how many steps to move, so the parameter is `int numberOfSteps` within parentheses
  - `int` tells compiler this parameter is an "integer" ("`moveForward` takes a single parameter called `numberOfSteps` of type `int`")

Outline

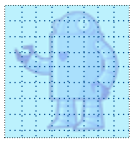
- Calling methods
- Declaring and defining a class
- Instances of a class
- Defining methods
- The `this` keyword

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 `chloeBot`)
- All `Robots` (all **instances** of the class `Robot`) are restricted to **the exact same capabilities**: the methods defined in the `Robot` class. What one `Robot` **instance** can do, all instances can do since they are made with the same blueprint!
- All `Robots` also have **the exact same properties** (i.e., every `Robot` has a `Color` and a `Size`)
  - they all have these properties (e.g. `Size`), but the values of these properties may differ between instances (e.g., a big `samBot` and small `chloeBot`)

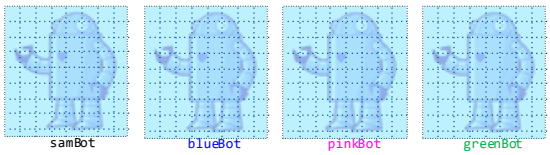
Classes and Instances (2/4)



The `Robot` class is like a blueprint

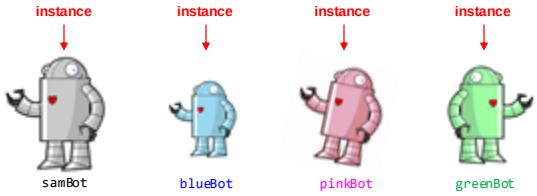
Classes and Instances (3/4)

- We can use the `Robot` class to build actual `Robot`s - **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

Join Code: 553500

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

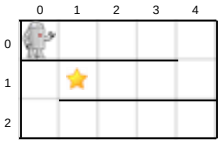
B. The call `blueBot.chaChaSlide();` will make `blueBot` do the cha-cha slide

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

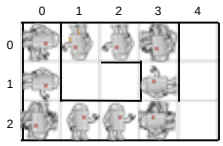
```
public class RobotMover {
    /* additional code elided */

    public void moveRobot(Robot myRobot) {
        // Your code goes here!
        // …
        // …
    }
}
```


Declaring vs. Defining Methods

- **Declaring** a method means the class knows how to do a new task, e.g., any instance of class `Robot` can `chaChaSlide()`
- **Defining** a method actually explains how all instances of the class execute this task (i.e., what sequence of commands it specifies)
  - `chaChaSlide()` could include stepping backwards, alternating feet, stepping forward
- For now, you will need to both **declare** and **define** your methods

A Variation on moveRobot (1/2)




```
public class RobotMover {
    /* additional code elided */

    public void newMoveRobot(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
- Any Robot instance like samBot only knows how to turn right, so must 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 new method in Robot called turnLeft

```
public class RobotMover {
    /* additional code elided */

    public void newMoveRobot(Robot myRobot) {
        myRobot.turnRight();
        myRobot.moveForward(2);
        myRobot.turnRight();
        myRobot.turnRight();   "turn left"
        myRobot.turnRight();
        myRobot.moveForward(3);
        myRobot.turnRight();
        myRobot.turnRight();   "turn left"
        myRobot.turnRight();
        myRobot.moveForward(2);
        myRobot.turnRight();
        myRobot.turnRight();   "turn left"
        myRobot.turnRight();
        myRobot.moveForward(2);
    }
}
```


Defining a Method (1/2)

- Almost all methods take on this general form:

explanation in later lecture

```
<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

```
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 (1/3)

- 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();`"?
- Recall the syntax for calling methods: <instance>.<method>();

```
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 (2/3)

- The `this` keyword allows an instance (like `samBot`) to call one of its own methods on itself
- `this` is short for "this same instance" or "defined in this method"
- Use `this` to call an existing method of `Robot` class (`turnRight()`) within a new method of `Robot` class (`turnLeft()`)

```
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 (3/3)

- 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 not required for code to work, but it is good style and CS15 expects it

```
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` class has `turnLeft()`, we can call `turnLeft()` on any instance of `Robot`
- We'll see how we can use `turnLeft()` to simplify our code in a few slides

```
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();
    }
}
```

TopHat Question

Join Code: 553500

```
public class Robot {

    /* additional code elided */

    public void turnLeft() {
        this.turnRight();
        this.turnRight();
        this.turnRight();
    }
}
```

Given the `turnLeft` 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

Summary

Class declaration

```
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();
    }
}
```

Class definition

Method declaration

Method definition

Simplifying our code using `turnLeft`

```
public class RobotMover {
    public void newMoveRobot(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);
    }
}
```

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

We've saved a lot of lines of code by using turnLeft!

This is good! More lines of code make your program harder to read, debug, and maintain

turnAround (1/3)

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

```
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!
    // …
    // …
    // …
}
```


turnAround (2/3)

- 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 can work as well

```
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();
    }
}
```


turnAround (3/3)

- Instead of calling turnRight, could call our newly created method, turnLeft
- Both solutions will lead to the same end goal, in that they will turn the robot around 180º
- How do they differ? When we try each of these implementations with samBot, what will we see in each case? Is one way better than the other?

```
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();
    }
}
```

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();`

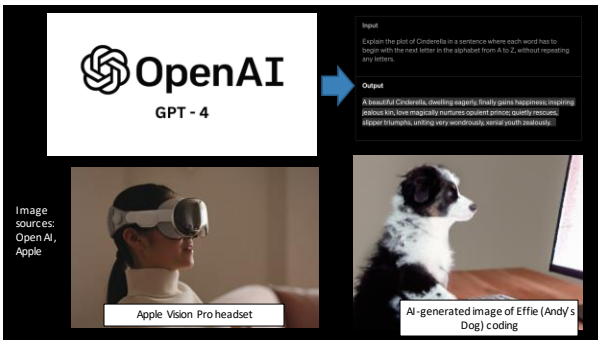
Announcements

- Lab 0 Linux and Terminal out today
  - If you did not sign up for section or have not received an email about your section, please email the HTAs
  - Review GitHub/IntelliJ setup before lab!
- Rattytouille out tomorrow!
  - Due Saturday 09/16
  - No Early or Late Hand-in
- RISD students: please 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

Intro to Socially Responsible Computing

CS15 Fall 2023

65 / 78



Technology

Lina Khan Is Coming for Amazon, Armed With an FTC Antitrust Suit

- FTC is expected to focus on Amazon's online marketplace
- Agency chief deemed unlikely to accept tweaks to the business

PRESS RELEASE

Justice Department Sues Google for Monopolizing Digital Advertising Technologies

Actors vs. AI: Strike brings focus to emerging use of advanced tech

SAG-AFTRA has joined the Writer's Guild of America in demanding a contract that explicitly demands AI regulations to protect writers and the works they create.

BuzzFeed Is Quietly Publishing Whole AI-Generated Articles, Not Just Quizzes

These read like a proof of concept for replacing human writers.

Image sources: NBC News, Futurism, Bloomberg, Justice Department

Headlines From 2022 Shown in this Lecture

AI Is Not Going To Replace Writers Anytime Soon – But The Future Might Be Closer Than You Think

Anticipating the emergence of AI-assisted academic misconduct

Recent Headlines

AS ACTORS STRIKE FOR AI PROTECTIONS, NETFLIX LISTS $900,000 AI JOB

A.I. in Barrington schools: We need a policy

Artificial intelligence will be part of academic integrity policy

Ban or Embrace? Colleges Wrestle With A.I.-Generated Admissions Essays.

68 / 78

What is Socially Responsible Computing?

- SRC @ Brown started in 2019 (in its 5th year)
- Brown CS: implemented across 18 CS courses
  - 3 Fall 2023 SRC-focused courses:
    - CSCI 1805 -- Computers, Freedom and Privacy
    - CSCI 1860 – Cybersecurity Law and Policy
    - CSCI 1870 – Cybersecurity Ethics
- Similar initiatives: embedded ethiCS at Harvard, Stanford, …
- Focus in CS15: get exposed to a broad range of topics that you can explore later

69 / 78

TECH DOES NOT EXIST IN A VACUUM

"**Technology is neither good, nor bad.**

Nor is it neutral…

… technology can have quite different results when introduced into different contexts or under different circumstances"

- Melvin Kranzberg, 1986

SRC is NOT about…

- Hating on technology
- Hating on capitalism
- Guilting you about your internship
- Telling you what to believe

71 / 78

Our approach

- *Cautious*, *pragmatic* techno-optimism
- Empowering you to **come to your own conclusions**
- Allowing you to (start to) understand…

How tech is shaped → How tech works → How tech shapes society

72 / 78

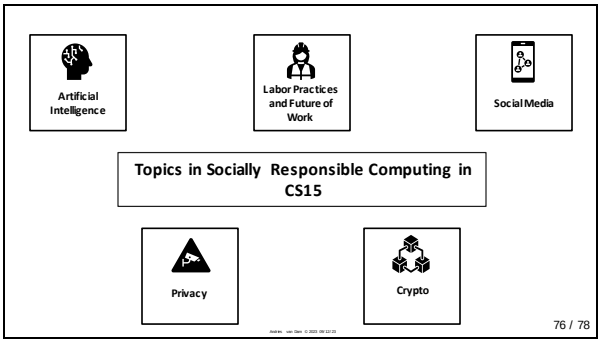
What does it mean for **you**?

Multidisciplinary solutions to today's challenges

What does it mean for **you**?

What does it mean for **you**?

Academic What courses might complement CS to get a more holistic understanding of tech and society?	**Career** What non-technical business decisions shape today's technology?
Individual	
Political What policies could enhance the benefits and mitigate the harms of tech?	**Technical** What can be done by developers to ensure that their products have good social impacts?



SRC in CS15

Mode of Delivery

- Mini-lectures (this Thurs: A.I.)
- Lab and section activities
- Extra credit discussion sections

77 / 78

Technology alone won't solve our problems

Image source: xkcd Here To Help

78 / 78