

# CSE 11

# Accelerated Intro to Programming

## Lecture 4

Greg Miranda, Summer 2 2021

This lecture is being recorded

# Announcements

- PA0.5 due today @ 11:59pm
- PA1 due today @ 11:59pm
- PA2 due Sunday @ 11:59pm
- Discussion 2 today @ 2pm
  - Covers PA2
- Lecture Quiz 1 due Friday @ 11:59pm
- Lecture Quiz 2 due Sunday @ 11:59pm

```
class Example {  
    int average(int n, int m, int o) {  
        return (n + m + o) / 3;  
    }  
    String withDotAtTheEnd(int n) {  
        return n + ".";  
    }  
    String ans = this.withDotAtTheEnd(this.average(6, 3, 8));  
}
```

# JShell

- The Java Shell tool (JShell) is an interactive tool for learning the Java programming language and prototyping Java code.
- The way to think about the environment of JShell:
  - Sort of inside a class
    - Can start writing field definitions and trying things out
  - Good tool for experimentation
    - Can write one field definition or method definition at a time

- Make a String

- String h = "hello";
  - JShell immediately prints out the string
- String h2 = "he" + "llo";
  - Evaluates the expression, shows us the value

- Methods already defined by Java that we can use

- String – is built-in Java class (i.e. already defined in Java)
  - Defines many methods

- String myName = "Greg";



- int nameLen = myName.length();



- Note: these method calls are using something other than this

- We can call methods on many different kinds of values in Java
- When we define a method within a class and call that method from within the class
  - Then we use this to refer to methods within the class
- When call a method that's in another class
  - We use a particular value and then use that method
    - That method is going to be able to use information about that class to get its answer

- length() – does something different depending on which value it's called from

- Other String methods:
  - `String myFullName = "Gregory Joseph Miranda";`
  - `String middle = myFullName.substring(8, 14);`
    - What did the method `substring()` do?

- length() and substring()
  - 2 methods defined on Java's built-in String class
  - Can use them to do different types of calculations with String
- A bunch more String methods to come...
- Main point:
  - String value – can use these existing methods to do this calculations
- 2<sup>nd</sup> big lesson:
  - Indexes – indexing into Strings to access the characters
    - Something we will be working with as we go forward



- Another String method:
  - `String myWeirdName = myFullName.replace("e", "WEIRD");`
- What did `replace()` do?
- What's the value of myFullName after calling `replace()`?  
*↳ unchanged* *immutable*
- Keep track of the String methods you learned about in your own notes
  - These methods are all written down online
    - Java documentation – we would be able to see all these methods
      - Quick search: Java string documentation
        - Many String methods we could use
        - `repeat()`

- String helloTwice = h.repeat(2);
- String manyHello = h.repeat(20);
- What if we want to find if another String appears in a String, like a search?

→ int index = myFullName.indexOf("Joseph");

- What if the String is not in my name?
  - int anotherIndex = myFullName.indexOf("Orange");
  - What happened?

→ 0+ – index where we found the String

- -1 – didn't find the String

- Just a few more String methods

- Working with the idea that there is built-in stuff in Java that we are going to be able to use
  - This will help us write interesting programs that work with and manipulate text

- String example program

```
class StringExamples {  
}
```

- Write a method called firstHalf that:
  - Takes a String and returns a new String that has just the first half of the characters from the input String
- When writing a method:
  - Think about what some examples are and what we expect the results to be:
    - We can write these down as fields
    - Then we can easily check if we are right after running the program
  - Examples first – then build up into the implementation
    - Do on paper/whiteboard first – then type them in

- One of the first things to think about is:
  - What method (or methods) out of the methods we saw on strings is going to be useful here
    - We will be able to accomplish this only with methods we have seen so far

substring  
length

- This showed us how to implement a method from a word problem prompt
- We thought through some examples
  - Which helped us to refine our understanding
- We experimented a little bit
  - Figured out we are okay with this empty String result
- This is the process we should use when implementing methods
  - i.e. Programming Assignments

- ① method header
- ② write tests
- ③ implement method

# New Data Type

- Previous data types:
  - String
  - int
- Examples
  - `boolean b1 = 4 < 5;`
  - `boolean b2 = 5 < 4;`
- New data type:
  - Boolean
    - Uses different kinds of operators
      - Comparison operators

- String – many different types of strings, infinite # of strings
  - Only limited by how much memory is in our computer
- int – somewhat limited
  - -2,147,483,648 to 2,147,483,647
- Boolean – only two values
  - true / false
    - Represents the answers to yes or no questions
      - 4 < 5
        - Asking the question: is 4 less than 5?

- Many boolean operators

- `boolean b3 = 4 == 4;`    *//checks for equality*

- `boolean b4 = 4 == 5;`

- = is not the same as ==

- = is used to create or initialize a field definition

- Assignment operator

- `boolean b5 = 5 > 4;`

- `boolean b6 = 5 >= 4;`

- As well as <=

- All of these are ways to compare numbers

- Gives true/false (yes or no) answers

- What happens if we use it to compare Strings?

- `boolean stringComp = "a" < "b";`

*→ compiler error*



- Useful idea when learning a new feature
  - Ask if it works with other things you've worked with before
- Comparison operators like < and > do not type check
  - Only numeric types work with Java's type checking
- What about == on Strings?
  - boolean stringComp = "a" == "a";
    - == does produce an answer on Strings
  - boolean stringComp = "a" == "b";
  - Does produce an answer, but not recommended for Strings
  - We will talk more about comparing Strings for equality in future weeks
  - Only use == for numeric comparisons in this course

→ .equals()

- Main lesson:
  - 2 new values
    - true/false
  - With new data type boolean
  - New relational and comparison operators that work with booleans
    - < >
    - <= >=
    - ==
  - Another comparison operator
    - • !=
      - boolean b7 = 4 != 5;
      - boolean b8 = 5 != 5;
      - Opposite of the == answer

# Boolean Operations

- What can you do given a boolean?
  - What if we want to ask more than simple questions
    - Are two things true at the same time?
    - Is one of two things true?

- Combining booleans into another boolean

- boolean and1 = true && true; *true*
- boolean and2 = true && false; *false*
- boolean and3 = false && true;
- boolean and4 = false && false;

- boolean or1 = true || true; *true*
- boolean or2 = true || false;
- boolean or3 = false || true;
- boolean or4 = false || false; *false*

*&& and*



*intersection*

*|| or*

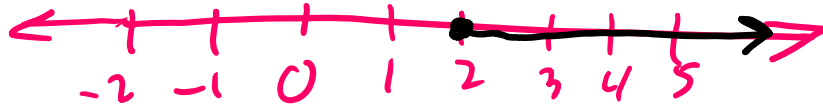


*union*

# Methods with Booleans

$$N \geq 2$$

- Number line 1



- Problem:

- Write a method that takes a number and returns true if it's in the region in our number line example

- Examples:

↓

2

true

5

true

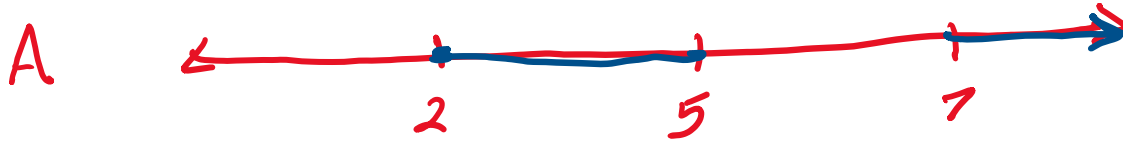
-1

false

```
boolean numberLine2(int number) {  
    return (number > 5) && (number < 7) || (number < 2);  
}
```

1st                      2nd

- What does this number line look like?



# More Complicated Questions with Methods

- Write the method to calculate absolute value that takes a number and returns the negation if it's less than 0, or that number otherwise
- Examples:
  - `int abs1 = this.absolute(-2);`     `//should produce 2`
  - `int abs2 = this.absolute(4);`     `//should produce 4`

```
int absolute(int number) {  
}
```

- Important comparison we need to do here
  - Is the number less than 0?
    - number < 0
  - Don't want to return true or false, we want to return the right number
- New Java syntax:
  - if statement

1



So far...

types → data types

- 3 different kinds of data

→ int

→ String → sequence of characters

→ boolean

- Very simple kinds of data

- Tons of information that can't be represented by only these 3 types of data.

- Applications keep track of much more than a single piece of informatino

→ Chess game

→ Microsoft Office

→ Google Documents

# Compound Data

- Central concepts in this course

- Methods

- We will keep coming back to this

- Compound Data

- Need to know how to combine simple pieces of data together into more complicated structures
    - Idea
      - Take multiple pieces of information
      - Package them together
      - Using that packaged together information

} encapsulation

- Going to learn more about what this thing called class is
  - We've been using this idea of this keyword **class**
    - Writing example classes with everything inside them
  - More about what's going on when we use a class
    - When we make a class
    - How we use them to package up data
- Examples of packing data together
  - Drawing / graphing

- Drawing / Graphing - common piece of compound data

- Tons of different fields use it
  - Art on paper, physics, engineering
- Points, coordinates, really matter

2D  
 $(x, y)$

3D  
 $(x, y, z)$

- Points are not defined by a single number

- Crucial to understanding points on a coordinate plane, or points anywhere
  - There are 2 numbers, packaged together that mean something
- This also gives us a visual, graphical representation of the data 4,5
  - We can see that there are 2 numbers involved in defining the point in the upper right

- What is it going to look like to specify the shape of a point in Java?

- Let's construct some points and do some computation

- At the top of the file, above ExamplesLec class - add class Point

```
class Point {  
    int x;  
    int y;  
}
```

- What's different about these field definitions than ones we've written before?

- What's different about these field definitions than field definitions we've written before?
  - No equals
- Every time we've written something before, we've had something on the right-hand side
  - A number or a calculation
    - `int examplesOfNum = 4 + 5;`
- We don't have that here. Why? ✓
  - This point class is going to describe all possible points
  - Can't just pick and say:
    - "I'm only going to represent the point where x is 4 and y is 5 with this class"

- Point is going to describe the shape of any points that are made up of 2 integers
  - Without specifying any particular point
- In a different part of the program:
  - We will specify particular points
- These are uninitialized field definitions
  - Field definitions without a value
    - ➔ No value given to these
  - These are the most common kinds of field definitions that we write

# Constructor

- Next thing we need to write something that looks like a method

```
Point (int x, int y) {  
    this.x = x;  
    this.y = y;  
}
```

- What makes this different than a method?



- In many way, this is not exactly like methods we've written before
  - The shape looks kind of like a method
  - Main thing different:
    - The only thing that appears before the parenthesis: **the name of the class**
    - That's what makes this really special
- This has a name:


### ➔ Constructor


- We write constructors by writing the name of the class before the open parenthesis
  - Instead of writing some return type and some name
- Special kinds of methods:
  - Used for creating new Points in this case
  - We use the constructor so we can create many different points using the same class


- For now, ignore the body of the constructor....

 • Treat this as cookie cutter code

- Always going to do the same pattern when we make a class

 • Name of the class

 • List of parameters that exactly match the fields

 • In the body, one line per parameter

- Says this.<name> = <name>;

- Future weeks we will get into more detail how constructors work

- Too many details to cover right now...

- For now, just use this pattern


# Creating Points

- Let's use the Point class to create several points
  - New syntax coming...
- `Point fourFivePoint = new Point(4, 5);`
- `Point negOneThreePoint = new Point(-1, -3);`
- What's different here?

- This is a way we can write a program that represents the two points that were drawn earlier
  - This gives us the ability to represent these as values inside Java
- Let's run it...
  - What is this going to print?

- It's useful to think about a picture representation of these Point values
  - Not only do we have the graph
  - But it's possible to draw these Points we created as pictures

# Objects

- Formal definition in Java:
  - Things that get created with new are called **objects** 
  - Each time we use the new keyword with the name of the class and values for each of the fields
    - We say we created a new object
- Something interesting happened
  - Wrote a program that used new
- What did it print out?

- Print out the same kind of things as the example classes we've used all along
  - Says new
  - The name of that class
  - Inside it lists the fields and their values
- The same process is happening for the Points we created with new as the ExamplesLec class
  - This tells us something about what's going on behind the scenes
    - Whenever we use run, something is happening
      - It's printing out a whole bunch of stuff

mac / <sup>terminal</sup> ./run Class Name  
windows | run Class Name

- Now we see exactly what is happening behind the scenes

- When we do **run**

- It's using new to create a single ExamplesLec object
    - Just makes one of them
    - Then it print it out

- It's as if it said:


→ **new** ExamplesLec ( )

- And then printed out all the fields inside ExamplesLec
  - There really is an ExamplesLec object that got created
  - Just like the two Point objects that were created




- So what is that going to look like in terms of this picture?
  - What should we draw to capture the idea that there is an ExamplesLec object that got created?

Box idea  
Reference idea

- Something we could do to see the difference
  - Make another field:
    - • Not going to create a Point here
      - Point fourFivePointAgain = this.fourFivePoint;
        - Use one that already exists
  - What is going to print out now?
  - What is the contents of ExamplesLec going to be now?
  - What are possible things this could look like next?

- Objects are created

- Whenever we have a field that is referring to an object

- It just stores a reference to the object 

- The object is created and just sits there

- And many fields can reference the same object

# Class Methods

- Previously we talked about methods
  - How to write them
  - How to call them
- How do we write methods that work with this compound data?
  - Like Points
- We should be able to write methods that do things with points

- Let's look at a simple method first
  - quadrant
    - takes no parameters
    - returns a string of which quadrant the points is in
  - What is the method header going to look like for this?
- Examples:
  - Make sure we understand what it should return for a few different cases
  - String quadA =
    - What should we write to call quadrant()?
    - What did we write before?
      - `this.quadrant();`
        - Does this still work?

- There is a rule based on how we call methods based on the classes of the objects that we are using
  - We have to use an object of the class that contains the method we want to call in order to call it
- To call quadrant()
  - We can't use this
    - **this** is referring to the ExamplesLec object
    - We have to use one of the Points
- The thing before the dot in the method call
  - Has to be a reference to an object of the class that contains the method



- Since `quadrant()` is defined inside the `Point` class
  - The object here has to be a reference to an object of the `Point` class
- `String quadA = this.fourFivePoint.quadrant();`
  - This is what we need to use to call the `quadrant` method on the `fourFivePoint` object
- • “We call the method on a reference to an object”
  - this.fourFivePoints - the reference to the object we are using
  - quadrant() - the method we are calling

- `String quadA = this.fourFivePoint.quadrant();`
  - What should this produce?
- Another example:
  - `String quadB = this.negOneThreePoint.quadrant();`
    - What does this produce?



- The same method, quadrant(), is called
  - But it changes its answer depending on which reference to an object we use to call it
- That means that whatever we write in the body of quadrant
  - Better depend on the values of the reference we used to call this method
    - Better depend on those values because we need to get different answers for these 2 lines
- The way this will work in the body of quadrant
  - We are going to use the keyword this again
    - The way we have used the keyword this before
- Introduce a new rule for this

↙ ↘ ↙ ↘  
→ if (this.x > 0 && this.y > 0) { ... }

- When we say this.x, the value that we get when we look up this.x
  - Is the value of x on the object we used to call the method
    - In the first case:
      - We used the object that has this.fourFivePoint to call the method
    - In the second case:
      - We used the object this.negOneThreePoint to call the method
  - In the first case, this.x will be 4
  - In the second case, this.x will be -1
  - In the first case, this.y will be 5
  - In the second case, this.y will be -3
  - The reference we get for this inside a method is always related to the reference that appeared before the method call.

↖  
this.fourFivePoint.quadrant();

- `if (this.x > 0 && this.y > 0) { ... }`
  - How do we finish this?

# Weekly Pay Problem

- `weeklyPay`: takes a number of hours worked and an hourly rate, and returns the pay with overtime (over 40 hours) counting as double the rate
- Examples: