



What was this course about?

1

- What are the big-picture **programming** concepts in this course?
- What is **object oriented programming (OOP)**?
- What are the **programming skills** you practiced in this course?



What did you learn about coding?

2

- What are the two fundamental categories of data types?
- List all of the data types we covered in this course
- List all forms of control flow covered in this course



What did you learn about OOP?

3

- What is a class? What is an object?
- What two kinds of things do objects have?
- What's the point of OOP?



What skills did you practice?

4

- What is fluent recall?
- Differentiate between "practical skills" and "knowledge skills" and give examples from this class.
- What is the point of mastering such skills? What's coming in future courses?

What did you learn about coding?

2

Data types

Primitive:

- Numbers: int, float, etc.
- booleans
- chars
- Strings
- null

Complex:

- Arrays and lists
- Objects

Control Flow

- Input / Output
- Variables
- Conditionals
- Loops
- Functions / Methods
- Exceptions

What was this course about?

1

• Programming Concepts:

- **Data:** Concepts related to creating or structuring the data processed by the computer.
- **Control:** Concepts related to controlling which line of code runs next, and which part of a line runs first.
- **Object oriented concepts:** A style of programming where you create new types of data to solve problems.
- **Skills:** reading, editing, writing, problem solving.

What skills did you practice?

4

Skills

- **Knowledge skills:** Fluent recall of coding concepts: e.g. What is a primitive type? What is an object? What is a method? (**Fluent recall:** Quick and effortless, without needing to look it up.)
- **Coding Skills: Understand:** read, trace, edit. **Write:** Translate "spec" into code.

Why? (Future classes)

- Write your own "spec":
- Algorithm design
 - Software engineering

What did you learn about OOP?

3

Basic OOP concepts

- A class is a definition of a new type of data (e.g. a Point). An object is a specific instantiation of a class: e.g. a specific Point like (2,2) or (5,3)
- Objects can have **instance variables** (data), and **methods** (chunks of code)

What's the point?

- Modeling (the world) with objects
- Solving problems with objects

2 ♠ What are ints/bools/vars/main?

5

- What is a **data type**? What makes a type **primitive** vs **complex**?
- What is an **integer**? What is an **int**?
- How are **ints** and **booleans** related?
- What is a **method**? What does it mean to "call a **method**"?
- What does it mean for a **method** to "return an **int**"?
- What's the relationship between two **int** variables with the same name in different **methods**?
- What is the "**driver**" pattern?

2 ♥ Syntax for ints/bools/variables?

6

Warm ups:

- Write the **syntax patterns** for **class** and **method**
- Write the syntax pattern for a method that returns an **int**. What about for a **boolean**?
- Write a table of **operators** for **ints** and what they do. Same for **booleans**
- Write the syntax pattern for **variable** assignment and update.

Writing Challenge: Define a class called **IntDemo** with two static methods, one that returns an **int** and one that returns a **boolean**. Use these **methods** to demonstrate the syntax for **variable assignment/update** and any interesting **operator** behavior that you know of.

2 ♦ Syntax for main method?

7

Warm ups:

- Write the syntax pattern for a **main method**
- Write the syntax pattern for a **instantiating a class** and calling a **method** on it.
- Write the syntax pattern for printing **output** to the console.

Writing Challenge: Define a class called **IntDemoRunner** with one static **main method**. Use this method to demonstrate the syntax for **instantiating** a new **IntDemo** object, calling its **methods**, and printing their **return values**.

2 ♣ Fluency with ints/bools/vars/main?

8

Warm ups:

- Run the code
- Watch the videos

Suggested activities:

- Use debugger to trace from driver to method calls
- Intentionally break code to see what is required and what isn't
- Add to **demoInt()** and **demoBool()** to explore additional operators and variable behavior.
- Make additional methods to investigate other primitive types.

Syntax for ints/bools/variables?

6

```
class IntDemo {
    public static int demoInt(){
        int a = 2; //Assignment to a
        int b = 6; //Assignment to b
        a = b * a; //Updating a
        return a;
    }
    public static boolean demoBool(){
        boolean a = true; //Assignment to a
        boolean b = false; //Assignment to b
        a = a && b; //Updating a
        return a;
    }
}
```

What are ints/bools/vars/main?

5

- **Data type:** values + allowed operations (primitive vs. complex)
- **int:** Java's primitive whole-number type; an **integer** is the general math concept it represents
- **boolean:** true/false; often the result of comparing ints
- **Method:** reusable named code block; "calling" runs it
- "Returns an **int**" = the call evaluates to an int
- Same-named **int** in different methods = independent
- **"Driver" pattern:** a class whose main uses other classes

Fluency with ints/bools/vars/main?

8

Prep for writing challenge:

- Practice answering all questions
- Practice all syntax patterns on paper
- Practice "compressing" code into your own words

Writing challenge:

- **Stage #1:** Delete and rewrite on your computer until you can do so with just the help of your compressed description
- **Stage #2:** On paper, practice writing code by hand using just the writing challenge prompts until you can do so without errors.
- **Stage #3:** Repeat stages #1 and #2 after a period of time has passed.

Syntax for main method?

7

```
class IntDemoRunner {
    public static void main(String[] args){
        IntDemo d = new IntDemo();
        d.demoInt(); //Does "nothing"...
        int x = d.demoInt(); //Can be observed!
        System.out.println(x); //Output x to console

        System.out.println(d.demoBool()); //More output
    }
}
```

3 ♠ What are strings/chars/drivers?

9

- What is a **char**? How is it different from a **String**?
- Is a **String** primitive or complex? Why?
- What does it mean to **concatenate** a **String**?
- What does it mean for a **driver** to "drive" **multiple** classes?
- What is a **main** method?
- Explain the parts of a **main** method's signature

3 ♥ Syntax for strings/chars?

10

Warm ups:

- Write the syntax pattern for a **char** variable and a **String** variable
- Write the syntax pattern for **concatenating** **Strings**
- List other **String** methods or operators
- List any "gotchas" with **String** concatenation that you know of

Writing Challenge: Define a class called **StringDemo** with a **static method** that returns nothing. Use this to demonstrate **String** concatenation, indexing, and any other **String** operators/methods you know.

3 ♦ Syntax for drivers?

11

Warm ups:

- Write the syntax pattern for a **Driver** class that drives 2 other classes.

Writing Challenge: Define a class called **MultiClassRunner** that creates an **IntDemo** and a **StringDemo** object, then calls methods on both.

3 ♣ Fluency with strings/chars/drivers?

12

Warm ups:

- Run the code
- Watch the videos

Suggested activities:

- Add to **greeting()** to explore additional string methods.
- What happens if you change `println` to `print`?
- What happens if you treat a `char` like a number (e.g. add or multiply a `char` by a number)?

Syntax for strings/chars?

10

```
class StringDemo {
    public static void greeting(){
        String s = "Hello!";
        System.out.println(s);
        System.out.println(s+s+s);

        char c1 = s.charAt(0);
        char c2 = s.charAt(5);
        System.out.println(c1+c2+c1);
    }
}
```

What are strings/chars/drivers?

9

- **char**: primitive type for a single character, e.g. 'a'
- **String**: complex type for text; a sequence of chars
- **Concatenate**: join strings/chars end-to-end with +
- A **driver** can create + use objects from several classes
- **Main method**: The method that is called when you run the program ("entry point")
 - "public" method can be called from other classes/files
 - "static" method is part of the class, not instances of the class
 - "void" method has no return value
 - "String[]" an array of strings containing command line arguments

Fluency with strings/chars/drivers?

12

Prep for writing challenge:

- Practice answering all questions
- Practice all syntax patterns on paper
- Practice "compressing" code into your own words

Writing challenge:

- **Stage #1**: Delete and rewrite on your computer until you can do so with just the help of your compressed description
- **Stage #2**: On paper, practice writing code by hand using just the writing challenge prompts until you can do so without errors.
- **Stage #3**: Repeat stages #1 and #2 after a period of time has passed.

Syntax for drivers?

11

```
class MultiClassRunner {
    public static void main(String[] args){
        IntDemo d = new IntDemo();
        StringDemo t = new StringDemo();

        System.out.println(d.demoInt());    //From IntDemo
        t.greeting();                       //From StringDemo
    }
}
```

4 ♠ What is a conditional?

13

- What is a **conditional**? What is **branching**?
- What is an **if statement**? What's the difference between **if** and **if-else**?
- List all **comparison operators** and say what they do.
- List all **logical operators** and say what they do.
- What is **short-circuit evaluation**? How does it differ between **&&** and **||**?
- What is the difference between an **expression** and a **statement**? Give an example of each. Compare and contrast a **ternary** and **if-statement**.
- Construct a truth table to demonstrate your understanding of boolean expression evaluation.

4 ♥ Syntax for conditionals (e.g. ifs)?

14

Warm ups:

- Write the syntax pattern for an **if statement**
- Write the syntax pattern for an **if-else statement**
- Write a **boolean expression** using a comparison and a logical operator
- Write a **short-circuit** expression using **&&** and using **||**

Writing Challenge: Define a class called **Checker** with a static method that demonstrates using a **conditional** to branch and return different **boolean** values.

4 ♦ Syntax for calls + conditionals?

15

Warm ups:

- Write the syntax pattern for calling a method and **branching** on its result
- Write the syntax pattern for an **if** with no else
- Write the syntax pattern for **nested** if statements

Writing Challenge: Define a class called **BranchRunner** whose **main** method calls **isPositive** and **prints a different message** depending on the returned value.

4 ♣ Fluency with conditionals?

16

Warm ups:

- Run the code
- Watch the videos

Suggested activities:

- Write code to explore short-circuit behavior
- Write code to explore **if** vs **if-else**
- Write code to explore all comparison operators
- Randomly generate boolean expression and complete its truth table

Syntax for conditionals (e.g. ifs)?

14

```
class Checker {
    public static boolean isPositive(){
        int n = 7;
        if(n > 0 || false == true){ //Will false == true be evaluated?
            return true;
        } else {
            return false;
        }
    }
}
```

What is a conditional?

13

- **Conditional:** any control structure that chooses between branches.
Branches: Different code paths that might execute at runtime
- **if:** runs a block when true; skips it otherwise
- **if-else:** runs one block if true, the other if false
- **Comparison operators:** == != < > <= >= → produce a boolean
- **Logical operators:** && (and), || (or), ! (not)
- **"Expression"** is evaluated to a value at runtime. **"Statement"** just executes an action at runtime. **Ternary** is a branching expression; **if-statement** is a branching statement.
- **Short-circuit:** && skips the right side once the left is false; || skips it once the left is true

Fluency with conditionals?

16

Prep for writing challenge:

- Practice answering all questions
- Practice all syntax patterns on paper
- Practice "compressing" code into your own words

Writing challenge:

- **Stage #1:** Delete and rewrite on your computer until you can do so with just the help of your compressed description
- **Stage #2:** On paper, practice writing code by hand using just the writing challenge prompts until you can do so without errors.
- **Stage #3:** Repeat stages #1 and #2 after a period of time has passed.

Syntax for calls + conditionals?

15

```
class BranchRunner {
    public static void main(String[] args){
        Checker c = new Checker();
        boolean result = c.isPositive();

        if(result){
            System.out.println("It's positive!");
        } else {
            System.out.println("It's negative!");
        }
    }
}
```

5 ♠ What is iteration (e.g. for loop)? 17

- What is a **loop**? Why use one instead of repeating code?
- What's the difference between a **for loop** and a **while loop**?
- What is a **loop counter**? What does it mean to **increment** it?
- What is an **infinite loop**? How do you avoid one?

5 ♥ Syntax for for-loops? 18

Warm ups:

- Write the syntax pattern for a **for loop** counting 0 to 9
- Write the syntax pattern for looping over each **char** in a **String**

Writing Challenge: Define a class called **Looper** with a static method that prints each **char** in a hardcoded word, using a **for loop**.

5 ♦ Syntax for while-loops? 19

Warm ups:

- Write the syntax pattern for an infinite **while** loop
- Write the syntax pattern for a **while loop** with the same behavior as a for-loop

Writing Challenge: Define a class called **LoopRunner** whose **main** method calls **Looper's** **countE()** method 99 times using a while loop.

5 ♣ Fluency with loops? 20

Warm ups:

- Run the code
- Watch the videos

Suggested activities:

- Write code to explore various loops
- Use internet to research **do-while** and **for each** loops.
- Write one program with 4 kinds of loops
- Write an infinite loop

Syntax for for-loops?

18

```
class Looper {
    public static void countE(){
        String word = "tree";
        for(int i = 0; i < word.length(); i++){
            char c = word.charAt(i);
            System.out.println(c);
        }
    }
}
```

What is iteration (e.g. for loop)?

17

- **Loop**: repeats code while a condition holds — avoids copy-pasting the same lines by hand
- **for loop**: best when you know how many times to repeat
- **while loop**: best when repeating until something changes
- **Loop counter**: tracks progress; **increment** = `i++`
- **Infinite loop**: condition never turns false; ensure condition becomes true to avoid this

Fluency with loops?

20

Prep for writing challenge:

- Practice answering all questions
- Practice all syntax patterns on paper
- Practice "compressing" code into your own words

Writing challenge:

- **Stage #1**: Delete and rewrite on your computer until you can do so with just the help of your compressed description
- **Stage #2**: On paper, practice writing code by hand using just the writing challenge prompts until you can do so without errors.
- **Stage #3**: Repeat stages #1 and #2 after a period of time has passed.

Syntax for while-loops?

19

```
class LoopRunner {
    public static void main(String[] args){
        Looper looper = new Looper();
        int count = 0;
        while(count < 99){
            System.out.println("**** " + count + " ****");
            looper.countE();
            count++;
        }
    }
}
```

6 ♠ What are arrays/fields?

21

- What is the **accumulator pattern**? What does the variable outside the loop do?
- What is an **array**? How is it different from a single variable?
- How do you access one element of an **array**? How do you find its **length**?
- What is a *field* or **instance variable**? What is the **getter/setter** pattern? What is a **constructor**?
- Distinguish between **static** vs **non-static** method and explain why **getters** and **setters** must be non-static.

6 ♥ Syntax for arrays?

22

Warm ups:

- Write the syntax pattern for declaring an **array** of ints with a few values
- Write the syntax pattern for looping over an **array** with a **for loop**
- Write the syntax pattern for **accumulating** values in an array

Writing Challenge: Define a class called **Summer** with a static method that returns the **sum** of a hardcoded **array** of ints.

6 ♦ Syntax for fields?

23

Warm ups:

- Write the syntax pattern for a class with a **instance variable (field)** and **constructor** that sets it.
- Write the syntax pattern for a **getter method** and a **setter method**.

Writing Challenge: Write a class called **Location** that encapsulates an x and a y value. Make appropriate getters and setters for each.

6 ♣ Fluency in arrays/fields?

24

Warm ups:

- Run the code
- Watch the videos

Suggested activities:

- Write a class **Person** and construct an instance that represents you
- Use **sumAll** to write **sumEven** which sums only even numbers in an array (Hint: Use **modulus** operator)

Syntax for arrays?

22

```
class Summer {
    public static int sumAll(){
        int[] nums = {2, 4, 6, 8};
        int total = 0;           //Accumulator variable
        for(int i = 0; i < nums.length; i++){
            total = total + nums[i]; //Update it
        }
        return total;           //Return it
    }
}
```

What are arrays/fields?

21

- **Accumulator pattern:** a variable declared before a loop, updated each pass to build a result
- **Array:** a fixed-size list of values of one type
- Access with `arr[i]`; length with `arr.length`
- **Instance variable:** Each **object** instance of the same class has its own copy of each instance variable. **field** is a synonym. **constructor:** special method that runs when class is instantiated.
- **static** method: called on the class itself; **non-static:** called on an object, can use its instance variables
- Getters and setters protect instance variables from direct access (which is considered "best practice")

Fluency in arrays/fields?

24

Prep for writing challenge:

- Practice answering all questions
- Practice all syntax patterns on paper
- Practice "compressing" code into your own words

Writing challenge:

- **Stage #1:** Delete and rewrite on your computer until you can do so with just the help of your compressed description
- **Stage #2:** On paper, practice writing code by hand using just the writing challenge prompts until you can do so without errors.
- **Stage #3:** Repeat stages #1 and #2 after a period of time has passed.

Syntax for fields?

23

```
class Location{
    int x;
    int y;

    public Location(int x, int y){
        this.x = x; this.y = y;
    }

    public int getX(){ return x; }
    public int getY(){ return y; }
    public void setX(int x){ this.x = x; }
    public void setY(int y){ this.y = y; }
}
```

7 ♠ What is guard/extend?

25

- What is a **guard statement**? Why check the "bad" case first and return early?
- What does **break** do inside a loop?
- What is an **abstract class**? What is **extending** a class?
- What does it mean for two classes to be **siblings** of the same parent?

7 ♥ Syntax for guard/break?

26

Warm ups:

- Write the syntax pattern for a **guard statement** (early return)
- Write the syntax pattern for a **break** inside a **for loop**

Writing Challenge: Define a class called **Finder** with a static method that returns **true** as soon as it finds a negative number in a hardcoded array, using a **guard statement** to break early when it does.

7 ♦ Syntax for extend/override?

27

Warm ups:

- Write the syntax pattern for an **abstract class** with an abstract method; show how it is overridden in another class.
- Write the syntax pattern for calling a static method and show how it differs from a non-static method call.

Writing Challenge: Define an **abstract class** called **Entity** that has a **Location** and a method **move**. Define **Treasure** as an extension of **Entity**.

7 ♣ Fluency in guard/extend?

28

Warm ups:

- Run the code
- Watch the videos

Suggested activities:

- Write a class **Animal** and extend it to create classes for **Cat** and **Fish**. Give appropriate methods to each.
- Write a loop with an early break wrapped in an **if**. Can you make the conditional so complicated that no one can tell if the break will execute or not?

Syntax for guard/break?

26

```
class Finder {
    public static boolean hasNegative(){
        int[] nums = {4, 7, -2, 9};
        if(nums.length == 0) return false; //Guard
        boolean found = false;
        for(int i = 0; i < nums.length; i++){
            if(nums[i] < 0){
                found = true; break;
            }
        }
        return found;
    }
}
```

What is guard/extend?

25

- **Guard statement:** an early if...return that handles a bad/edge case up front
- **break:** immediately exits the nearest loop
- **Abstract class:** a parent that can't be new'd (instantiated) directly — only its children can. Extending makes a new child class from the parent -- inheriting the methods and fields.
- **Siblings:** two subclasses of the same parent, each with their own override

Fluency in guard/extend?

28

Prep for writing challenge:

- Practice answering all questions
- Practice all syntax patterns on paper
- Practice "compressing" code into your own words

Writing challenge:

- **Stage #1:** Delete and rewrite on your computer until you can do so with just the help of your compressed description
- **Stage #2:** On paper, practice writing code by hand using just the writing challenge prompts until you can do so without errors.
- **Stage #3:** Repeat stages #1 and #2 after a period of time has passed.

Syntax for extend/override?

27

```
abstract class Entity {
    public Location location = new Location(0,0);
    public abstract void move(int x, int y);
    public boolean isAt(int x, int y){
        return this.location.getX() == x && this.location.getY() == y;
    }
}

class Treasure extends Entity {
    public void move(int x, int y){ } //Doesn't move
}
```

8 ♠

What are params/methods?

29

- What is a **parameter**? How is it different from a **local variable**?
- What's the difference between a **parameter** and an **argument**?
- Can a parameter be an **array**?
- When you **override** a method that has a parameter, what must match?
- What happens if the parameter **type** doesn't match the parent's?

8 ♥

Syntax for params/methods?

30

Warm ups:

- Write the syntax pattern for a method with **multiple primitive parameters**
- Write the syntax pattern for a method with **an array parameter**
- Write the syntax pattern for a method with **an object parameter**

Writing Challenge: Make a class called **Monster** that extends **Entity**. Write a **Room** class that has an array of **Monsters** with a corresponding setter.

8 ♦

Syntax for object params?

31

Warm ups:

- Write the syntax pattern for a method with **an object parameter**
- Write the syntax pattern for a **"wrapper" class**

Writing Challenge: Write **RoomWrapper** that has a **Room** variable and a method **monstersAt(Location l)** that returns the number of monsters at that **Location**.

8 ♣

Fluency with params?

32

Warm ups:

- Run the code
- Watch the videos

Suggested activities:

- Write a method with a parameter of one type. Call it with the wrong type. What compiler error is thrown?
- Can method have 10 parameters? 100? Is there a limit where an error is thrown?

Syntax for params/methods?

30

```
class Monster extends Entity {
    public void move(int x, int y){
        this.location.setX(this.location.getX() + x);
        this.location.setY(this.location.getY() + y);
    }
}

class Room {
    public int width, height;
    public Monster[] monsters;
    public void setMonsters(Monster[] monsters){
        this.monsters = monsters;
    }
}
```

What are params/methods?

29

- **Parameter:** a local variable a method declares to receive input, filled in from the caller's argument
- **Argument:** the actual value passed in at the call site
- A parameter can be any type, including an **array**
- An override must keep the same parameter **type**
- A mismatched type = a **new**, unrelated method, not an override

Fluency with params?

32

Prep for writing challenge:

- Practice answering all questions
- Practice all syntax patterns on paper
- Practice "compressing" code into your own words

Writing challenge:

- **Stage #1:** Delete and rewrite on your computer until you can do so with just the help of your compressed description
- **Stage #2:** On paper, practice writing code by hand using just the writing challenge prompts until you can do so without errors.
- **Stage #3:** Repeat stages #1 and #2 after a period of time has passed.

Syntax for object params?

31

```
class RoomWrapper {
    Room room;
    public RoomWrapper(Room room){
        this.room = room;
    }
    public int monstersAt(Location l){
        int total = 0;
        for(int m = 0; m < room.monsters.length; m++){
            if(room.monsters[m].isAt(l.getX(), l.getY())) total++;
        }
        return total;
    }
}
```

9 ♠ What are nested loops/factories? 33

- What are some use-cases for a nested for-loop?
- Explain the "factory" pattern

9 ♥ Syntax for nested loops? 34

Warm ups:

- Write the **syntax pattern** for a nested for-loop that prints a grid
- Improve/debug the code on the back of this card.

Writing Challenge: Give **RoomWrapper** a **print()** method that prints where all monsters are in the room. (Don't extend **RoomWrapper**; just add onto your existing one.)

9 ♦ Syntax for factories? 35

Warm ups:

- Improve/debug the code on the back of this card.

Writing Challenge: Make a class called **RandomFactory** with a static method **randomLocation(int maxX, int maxY)** to return a **Location** with x and y values between 0 and the given maximums. Also write a static method **randomMonsters(int num, int maxX, int maxY)** to return a **Monster[]** of **num** monsters at random locations.

9 ♣ Fluency in nested loops/factories? 36

Warm ups:

- Run the code
- Watch the videos

Suggested activities:

- Fix the compile errors in the code in this module (See my video for details)
- Write your own **Person** class. Then write a **PersonFactory** to construct interesting people: yourself, your favorite historical figure, etc.
- Write a **Pet** class. Give each **Person** an array of ****Pet[]***.
- Change your **PersonFactory** to construct pets for people returned.

Syntax for nested loops?

34

```
class RoomWrapper {  
    ... //Previous code here (not shown)  
  
    public void print(){  
        for(int i = 0; i < this.room.width; i++)  
            for(int j = 0; j < this.room.height; j++)  
                System.out.print(this.monstersAt(new Location(i,j)));  
                System.out.println();  
    }  
}
```

What are nested loops/factories?

33

- **Factory pattern:** a class (or static method) whose job is to construct other objects
- **Nested for-loop:** Useful for printing 2D grids, iterating over a 2D array (e.g. `int[][]`), or comparing every pair of items from two collections (e.g. arrays).

Fluency in nested loops/factories?

36

Prep for writing challenge:

- Practice answering all questions
- Practice all syntax patterns on paper
- Practice "compressing" code into your own words

Writing challenge:

- **Stage #1:** Delete and rewrite on your computer until you can do so with just the help of your compressed description
- **Stage #2:** On paper, practice writing code by hand using just the writing challenge prompts until you can do so without errors.
- **Stage #3:** Repeat stages #1 and #2 after a period of time has passed.

Syntax for factories?

35

```
import java.util.Random;  
  
class RandomFactory {  
    public static Location randomLocation(int maxX, int maxY){  
        Random rand = new Random();  
        return new Location(rand.nextInt(maxX), rand.nextInt(maxY));  
    }  
    public static Monster[] randomMonsters(int num, int maxX,  
                                           int maxY){  
        Monster[] monsters = new Monster[num];  
        for(int i = 0; i < num; i++)  
            monsters[i] = new Monster(randomLocation(maxX, maxY));  
    }  
}
```

10 ♠ What are 2d-arrays/modeling? 37

- What is a **multi-dimensional array**? What does `int[][]` represent?
- How do you access a value in a **2D array**?
- What is "Modeling" in the context of OOP?
- How are **instance variables** useful in modeling? How are **methods** useful?
- What might multi-dimensional arrays be useful for modeling?
- What is the value stored in an instance variable that has not been set?
- What happens a method is called on an instance variable that has not been set?

10 ♥ Syntax for 2d-arrays? 38

Warm ups:

- Write the syntax pattern for initializing a 2D array of **ints** (`int[][]`)
- Write the syntax pattern for initializing a local 2D array of **objects** (e.g. `Room[][]`) using nested loops.
- Write the pattern for the above when the 2D array is an instance variable.

Writing Challenge: Model a **Dungeon** storing the player's state as a **Monster** named "player", a grid of rooms (`Room[][]`), and a **Location** to track which room the player is currently in.

10 ♦ Syntax for 2d-arrays of objects? 39

Warm ups:

- Write the syntax pattern for accessing a 2D array cell using an instance of **Location**
- Write the syntax pattern for extracting the dimensions of a 2D array.

Writing Challenge: Add `nextRoom()` to **Dungeon** to move the player to a random room (hint use **RandomFactory**), a method to return the **Room** the player is in, and a method to print the monsters in that room. When a player moves to a new room, it should replace the first monster in the room.

10 ♣ Fluency in 2d-arrays? 40

Warm ups:

- Run the code
- Watch the videos

Suggested activities:

- Write your own **Person** class. Then write a **PersonFactory** to construct interesting people: yourself, your favorite historical figure, etc.
- Write a **Pet** class. Give each **Person** an array of `**Pet[]*`.
- Change your **PersonFactory** to construct pets for people returned.

Syntax for 2d-arrays?

38

```
class Dungeon{
    public Monster player = new Monster(new Location(0,0));
    public Location current_room = new Location(0,0);
    public Room[][] rooms = new Room[5][5];
    public Dungeon(){
        for(int i = 0; i < this.rooms.length; i++)
            for(int j = 0; j < this.rooms[0].length; j++) {
                rooms[i][j] = new Room(10,10);
                rooms[i][j].setMonsters(
                    RandomFactory.randomMonsters(10, 10, 10)
                );
            }
    }
}
```

What are 2d-arrays/modeling?

37

- **Multi-dimensional array**: an array of arrays, e.g. a grid
- Access with `arr[row][col]`
- **Modeling**: representing a real (or imagined) thing as **objects** -- using **instance variables** to model the state of things and **methods** to model behavior
- 2D arrays are useful for anything "grid like" -- e.g. a game board or a spreadsheet of data.
- Before a variable is assigned, if it is non-primitive, it's value is the **null** value. If a method is called on **null** an error will be thrown.

Fluency in 2d-arrays?

40

Prep for writing challenge:

- Practice answering all questions
- Practice all syntax patterns on paper
- Practice "compressing" code into your own words

Writing challenge:

- **Stage #1**: Delete and rewrite on your computer until you can do so with just the help of your compressed description
- **Stage #2**: On paper, practice writing code by hand using just the writing challenge prompts until you can do so without errors.
- **Stage #3**: Repeat stages #1 and #2 after a period of time has passed.

Syntax for 2d-arrays of objects?

39

```
class Dungeon { ... // Previous code here
    public void nextRoom(){
        this.getCurrentRoom().monsters[0] = new Monster();
        this.current_room = RandomFactory.randomLocation(
            this.rooms.length, this.rooms[0].length);
        this.getCurrentRoom().monsters[0] = this.player;
    }
    public Room getCurrentRoom(){
        return this.rooms[this.current_room.x][this.current_room.y];
    }
    public void printCurrentRoom(){
        new RoomWrapper(this.getCurrentRoom()).print();
    }
}
```



What are interfaces/casting?

41

- What is an **interface**? How is it different from a class?
- What does it mean for a class to **implement** an interface?
- Can one class implement **more than one** interface?
- Why might **implementing interfaces** be sometimes preferable to **extending classes**?
- What is **casting**?



Syntax for interfaces?

42

Warm ups:

- Write the syntax pattern for an **interface** with a method signature
- Write the syntax pattern for a class that **implements** an interface

Writing Challenge: Use an **interface** to mark **Monsters** and **Rooms** as **Attackable** (has a method **attacked(Attackable attacker)**). An attacked monster should take 10 points of damage. Attacking a **Room** should cause monsters in the room to take 1 point of damage.



Syntax for casting?

43

Warm ups:

- Write the syntax pattern for **casting** a reference to a specific type

Writing Challenge: Extend **Monster** with a new class **PlayableMonster**. Implement the **attacked(Attackable attacker)** method such that if the player is attacked by a **Monster**, the player and attacker both take 5 points of damage.



Fluency in interfaces/casting?

44

Warm ups:

- Run the code
- Watch the videos

Suggested activities:

- Write **Animal** and extend to **Cat** and **Fish**. Also make **Vehicle** and extend to **Car** and **Boat**. Make an interface **Floatable** that **Fish** and **Boat** can both implement
- Write an example of casting a **Floatable** to a **Fish** and calling one of its methods. What error is thrown if the **Floatable** was a **Boat** not a fish?

Syntax for interfaces?

42

```
public interface Attackable{ public void attacked(); }

public class Monster extends Entity implements Attackable {
    ... //Previous code
    int health = 10;
    public void attacked(Attackable attacker){ this.health -= 10; }
}

public class Room implements Attackable {
    ... //Previous code
    public void attacked(Attackable attacker){
        for(int i = 0; i < this.monsters.length; i++)
            this.monsters[i].health -= 1;
    }
}
```

What are interfaces/casting?

41

- **Interface:** a contract listing method signatures, with no bodies
- **implements:** a class promises to provide bodies for an interface's methods. Yes — a class can implement multiple interfaces
- **Interfaces vs. extending:** Java only allows single inheritance (extends one class), but a class can implement many interfaces — so interfaces let unrelated classes (like Treasure and Monster) share a capability (e.g. Interactable) without forcing them into one rigid "is-a" hierarchy
- **Casting:** A syntax for telling the JVM to treat a object as if it is a different (related) type.

Fluency in interfaces/casting?

44

Prep for writing challenge:

- Practice answering all questions
- Practice all syntax patterns on paper
- Practice "compressing" code into your own words

Writing challenge:

- **Stage #1:** Delete and rewrite on your computer until you can do so with just the help of your compressed description
- **Stage #2:** On paper, practice writing code by hand using just the writing challenge prompts until you can do so without errors.
- **Stage #3:** Repeat stages #1 and #2 after a period of time has passed.

Syntax for casting?

43

```
public class PlayableMonster extends Monster{
    public void attackable(Attackable attacker){
        if(attacker instanceof Monster){
            Monster m = (Monster) attacker;
            m.health -= 5;
            this.health -= 5;
        }
    }
}
```

Q ♠ What's IO/exceptions?

45

- What is a **Scanner**? What does `System.in` represent?
- How do you read a line of **user input** with a **Scanner**?
- What is **file I/O**? Why might a program need to read a **file**?
- What does `throws Exception` on a method do?
- What is a language "ecosystem" vs the "standard library"?

Q ♥ Syntax for IO?

46

Warm ups:

- Write the syntax pattern for using a **Scanner** to ask the user what to do next.

Writing Challenge: Give **PlayableMonster** a **nextAction()** method to get input from the user and move accordingly.

Q ♦ Syntax for importing/exceptions?

47

Warm ups:

- Write the syntax pattern for **importing** a class from another package
- Write the syntax pattern for `main` that `throws Exception`

Writing Challenge: Write a driver for **Dungeon** that lets the player move around inside a room and from room to room. Make it throw an **exception** if a wrong command is entered.

Q ♣ Fluency in IO/exceptions?

48

Warm ups:

- Run the code
- Watch the videos

Suggested activities:

- Add a new feature of your choice to the game
- Look online for an interesting Java library in the larger ecosystem. Are you able to include it into your project?

Syntax for IO?

46

```
import java.util.Scanner;
class PlayableMonster extends Monster {
    ... //Previous code
    public void nextAction(){
        System.out.println("north or south?");
        Scanner sc = new Scanner(System.in);
        String line = sc.nextLine();
        if(line.startsWith("north"))
            this.move(0,-1);
        else if(line.startsWith("south"))
            this.move(0,1);
    }
}
```

What's IO/exceptions?

45

- **Scanner**: an object that reads input piece by piece
- `System.in`: the keyboard / standard input stream
- Read a line with `sc.nextLine()`
- **File I/O**: reading/writing a file on disk — so data can outlive the program instead of being hardcoded
- `throws Exception`: lets the method skip handling errors (bypassing the safer default)
- **Standard library**: It's the core features of a language; comes when you install the language. **Ecosystem**: All the libraries written by others (huge); must be downloaded separately

Fluency in IO/exceptions?

48

Prep for writing challenge:

- Practice answering all questions
- Practice all syntax patterns on paper
- Practice "compressing" code into your own words

Writing challenge:

- **Stage #1**: Delete and rewrite on your computer until you can do so with just the help of your compressed description
- **Stage #2**: On paper, practice writing code by hand using just the writing challenge prompts until you can do so without errors.
- **Stage #3**: Repeat stages #1 and #2 after a period of time has passed.

Syntax for importing/exceptions?

47

```
class DungeonDriver {
    public static void main(String[] args) throws Exception {
        Dungeon d = new Dungeon();
        d.player = new PlayableMonster();
        while(true){
            System.out.println("Want to 'act' or 'change rooms'?");
            Scanner sc = new Scanner(System.in);
            String line = sc.nextLine();
            if(line.startsWith("act")) d.player.nextAction();
            else if(line.startsWith("change")) d.nextRoom();
            else throw new Exception("Game over. Please follow directions");
        }
    }
}
```



What is List/null/try/catch?

49

- What is a **List**? How is it different from an **array**?
- How do you **add** to a List? How do you get its **size**?
- What is **null**? What happens if you call a method on a **null** reference?
- What is the point of try/catch?



Syntax for Lists?

50

Warm ups:

- Write the syntax pattern for initializing and populating an **ArrayList**

Writing Challenge: Use an **ArrayList** instead of a **Monster[]** in **Room**.



Syntax for try/catch?

51

Warm ups:

- Write the syntax pattern for a **try/catch** block
- Write the syntax pattern for checking if a reference is **null**
- Write the syntax pattern for **throwing** an exception

Writing Challenge: Change **PlayableMonster**'s **nextAction()** method to give the player a way to move in all directions. Throw an exception if the direction is not possible.



Fluency in... Java?

52

Warm ups:

- Run the code
- Watch the videos

Suggested activities:

- Make another change and practice "let the errors guide you." Try making two changes at once. Does the strategy still work?
- Refactor your code: e.g. organize into folders, change the name of a class, move a method from one class to another. How well do the errors guide you?
- Research online and write code to explore additional primitives that we didn't cover: e.g. **long**, **float**, and **enum**

Syntax for Lists?

50

```
import java.util.ArrayList;

class Room implements Describable {
    //Change this line:
    public Monster[] monsters;
    //To this:
    public ArrayList<Monster> monsters;
}

//And then... use the errors to help you out... :)
```

What is List/null/try/catch?

49

- **List**: a resizable, growable collection (unlike a fixed-size array)
- `list.add(x)` adds; `list.size()` returns the count
- `null`: a reference pointing to "nothing"
- Calling a method on `null` throws a **NullPointerException**
- `try/catch` lets you handle code even if it would error and exit by design

Fluency in... Java?

52

Prep for writing challenge:

- Practice answering all questions
- Practice all syntax patterns on paper
- Practice "compressing" code into your own words

Writing challenge:

- **Stage #1**: Delete and rewrite on your computer until you can do so with just the help of your compressed description
- **Stage #2**: On paper, practice writing code by hand using just the writing challenge prompts until you can do so without errors.
- **Stage #3**: Repeat stages #1 and #2 after a period of time has passed.

Syntax for try/catch?

51

```
class PlayableMonster extends Monster {
    ... //Previous code

    public void nextAction(){
        ... //Previous code

        //Add new logic here
    }

    //Then... let the errors guide you... :)
}
```