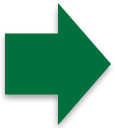
CS 10:
Problem solving via Object Oriented Programming

Introduction

Agenda



1. You, me, and this course
2. Why Object-Oriented Programming (OOP)
3. Java intro
4. Variables
5. Arrays

Let's start with our backgrounds

Your background

How did you satisfy the pre-reqs?

- CS 1
- ENGS 20
- AP/department exam
- Other

If you never take another CS class, I hope you'll be a better consumer of computer scientist's/data scientist's work products

Your plans?

CS majors? Minors? Not sure? Other?

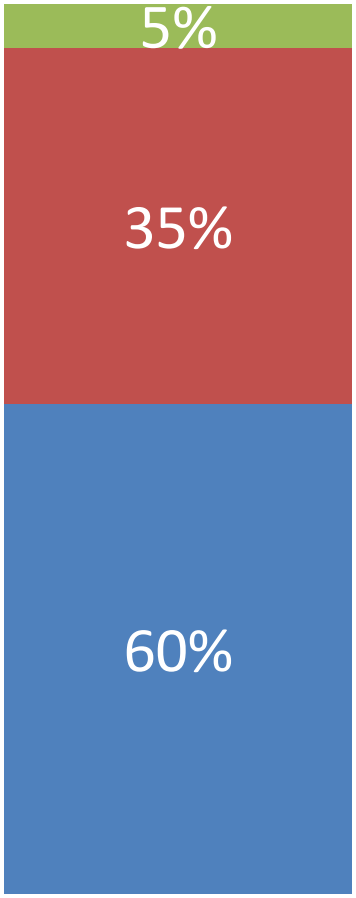
My background

This course is about solving problems with OOP, not simply how to program in Java

- Focus will be on solving problems with Object Oriented Programming (OOP), and you'll learn some Java along the way
- OOP is not the only way to solve problems, but it can be useful
- The course has three main components that overlap somewhat:
  1. Object Oriented Programming concepts and Java basics
  2. Abstract Data Types (ADTs) such as queues, stacks, trees, and graphs that form building blocks for solving problems (you'll see these ADTs again and again in CS)
  3. Solving wide range of real problems (graphics manipulation, characterize social networks, play Kevin Bacon game, compress files, analyze text…)
- You will learn far more by actually implementing things than you will by simply reading the material or only attending lectures

Material will be covered in lecture, section meetings, homework, and exams

Syllabus: http://www.cs.dartmouth.edu/~tjp/cs10



Section (Recitation) meetings

Homework

- Short assignments (SA): 10%
- Problem sets (PS): 25%

Exams

- Midterm1: 20%
- Midterm2: 20%
- Final: 20%

Textbook:

Data Structures & Algorithms in Java , 6th ed, by Goodrich, Tamassia, and Goldwasser

Lectures

- Stay home if sick
- **Show up on time**

LLMs

See course web site

We will also be using Canvas and Slack for announcements and help

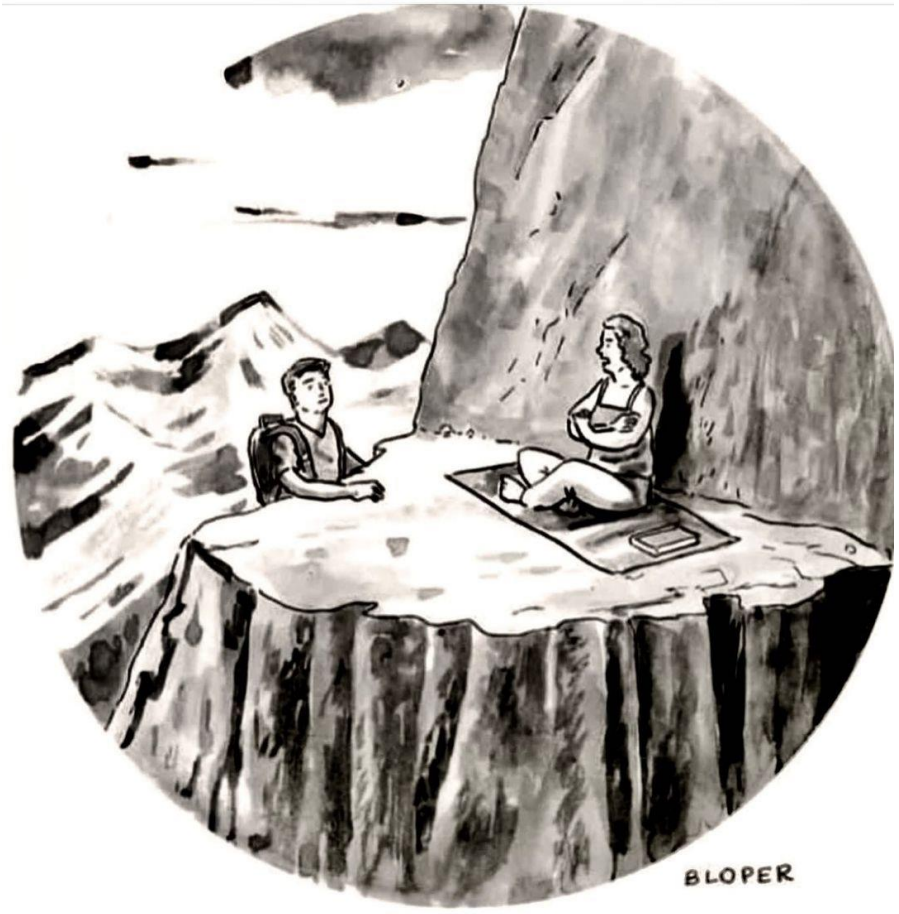
Canvas

- Course announcements and homework submissions
- Section assignments

Slack (access via Canvas)

- Q&A forum
- Ask questions, get answers
- **Don’t post code!**

Let me know if you don’t have access



"The answers you seek can be found in the syllabus."

Short Assignment 0 (SA-0) is out, complete survey before 8:00am tomorrow

SA-0

Find assignment on Canvas

1. Take course survey to understand your background and assign you to a section
2. Set up development environment
   - Instructions and screen shots provided on website
   - We will use IntelliJ IDEA for this course
3. Create your first Java class
4. Read and acknowledge course policies and honor code

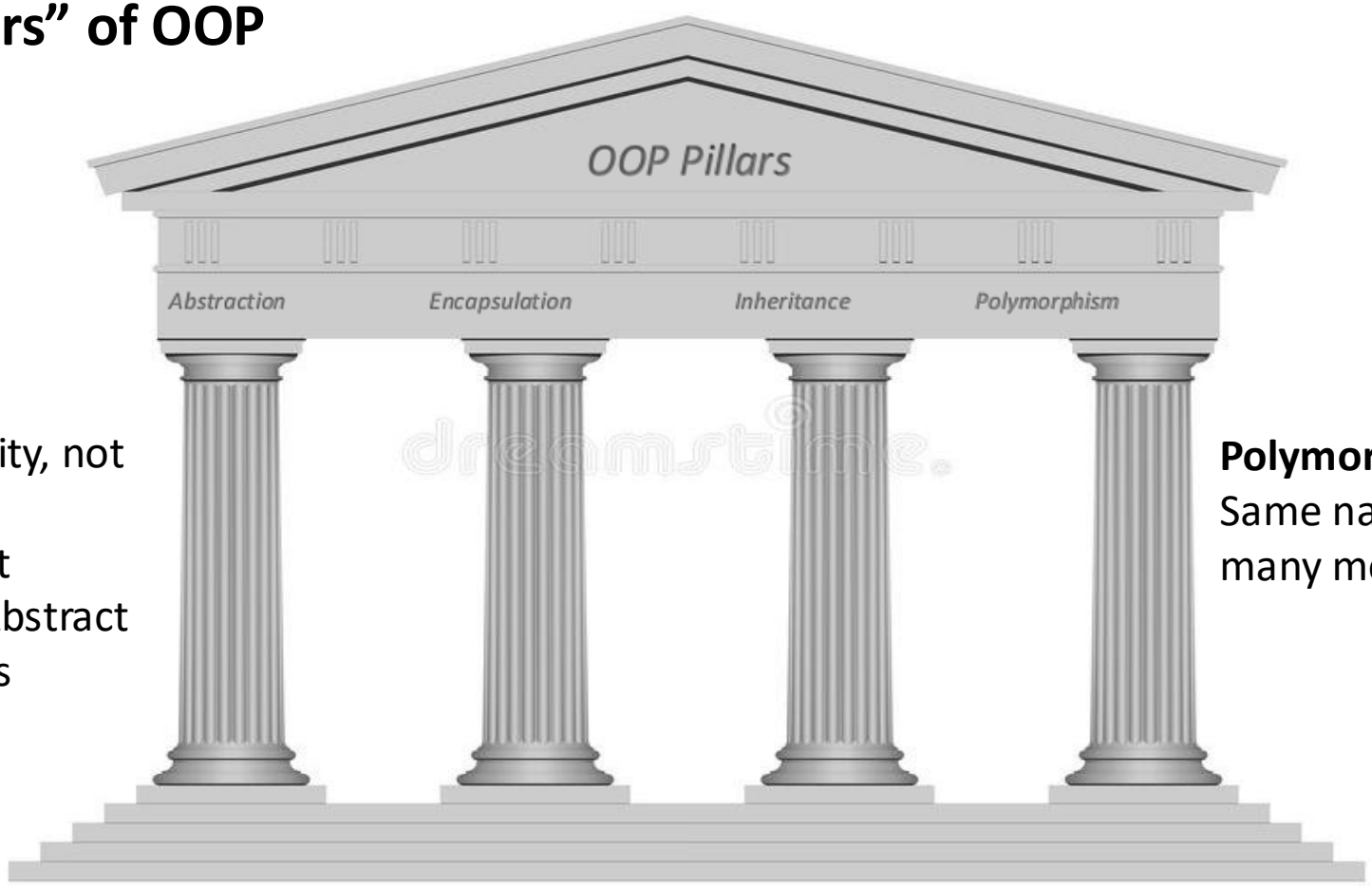
Complete survey **before 8:00am tomorrow** (or risk getting assigned to inconvenient section time!)

Agenda

1. You, me, and this course
2. Why Object-Oriented Programming (OOP)
3. Java intro
4. Variables
5. Arrays

OOP relies on four main pillars to create robust, adaptable, and reusable code

Four "pillars" of OOP



Abstraction

- Name functionality, not how to implement
- Leads to Abstract Data Types (ADTs)

Polymorphism

Same name, many meanings

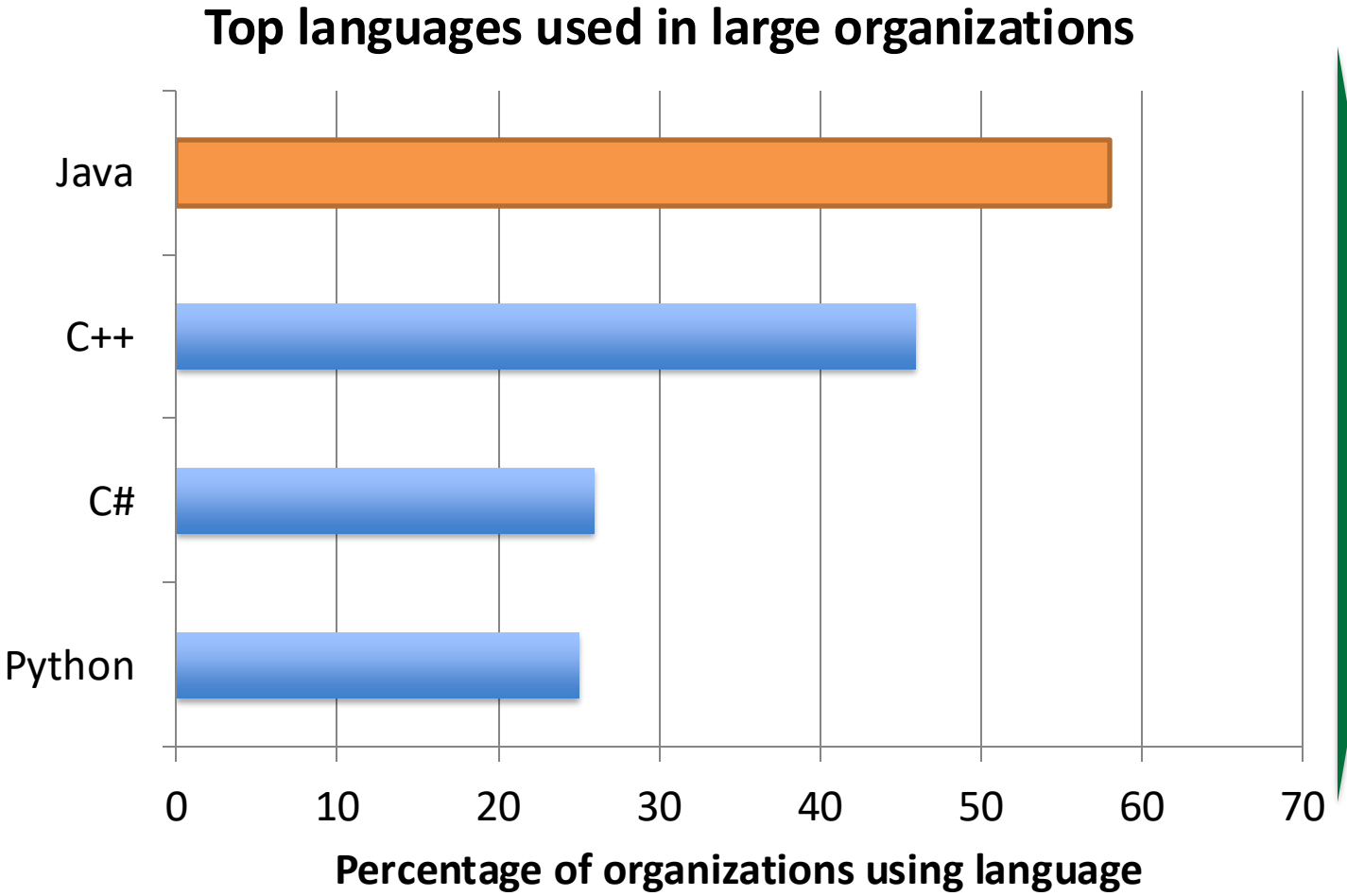
Encapsulation

- Bind code and data into one thing called an object
- Code called methods in OOP (not functions)

Inheritance

- Create specialty versions that "inherit" functionality of parent
- Reduces code

OOP is popular, especially in large organizations



- Each of the most common languages is object oriented
- Java is particularly popular in large organizations

Source: https://www.cloudfoundry.org/wp-content/uploads/Developer-Language-Report_FINAL.pdf (Javascript omitted)

Why is OOP in general, and Java in particular, so popular?

Approved answer: because it makes solving many types of problems easy (or perhaps easier)

Paul Graham's answer: it keeps mediocre programmers from doing too much damage

- In the real world, on a single project you may have dozens (or hundreds) of programmers working with thousands of objects – no one knows them all
- People come and go during the course of a non-trivial project – maintaining corporate knowledge is difficult
- We will see that objects can help prevent well-meaning people from making costly mistakes

Agenda

1. You, me, and this course
2. Why Object-Oriented Programming (OOP)
3. Java intro

Key point:

- **Java is a compiled, strongly typed language**

4. Variables
5. Arrays

We will be using Java, these things may blow your mind

Depending on your background, this may be weird:

- Must compile a program before it runs (so everything must be syntactically correct ahead of run time)
- Declare variable and give them a type
- White space/brackets
- For-each loops

Onward to OOP glory!

Image: https://www.askideas.com/man-riding-giant-chicken-funny-picture/

In keeping with tradition, we'll start with "Hello world"

HelloWorld.java

1. Start IntelliJ, create "cs10" Java Project (only need to do this one time)
2. Create "day1" Source folder to logically group your source code (e.g., "PS1" Source folder holds all the source code for Problem Set 1)
3. Create new "HelloWorld" class in "day1" source folder
   - File on disk is "HelloWorld.java"
   - Class Name is "HelloWorld"
   - IntelliJ "stubs" out "main" method (where program execution starts)

Other items of note:

Javadoc

- Java documentation feature
- Enter description for Class or method
- Starts with "/**", ends with "*/"
- Can add tags such as "@author" or "@param"

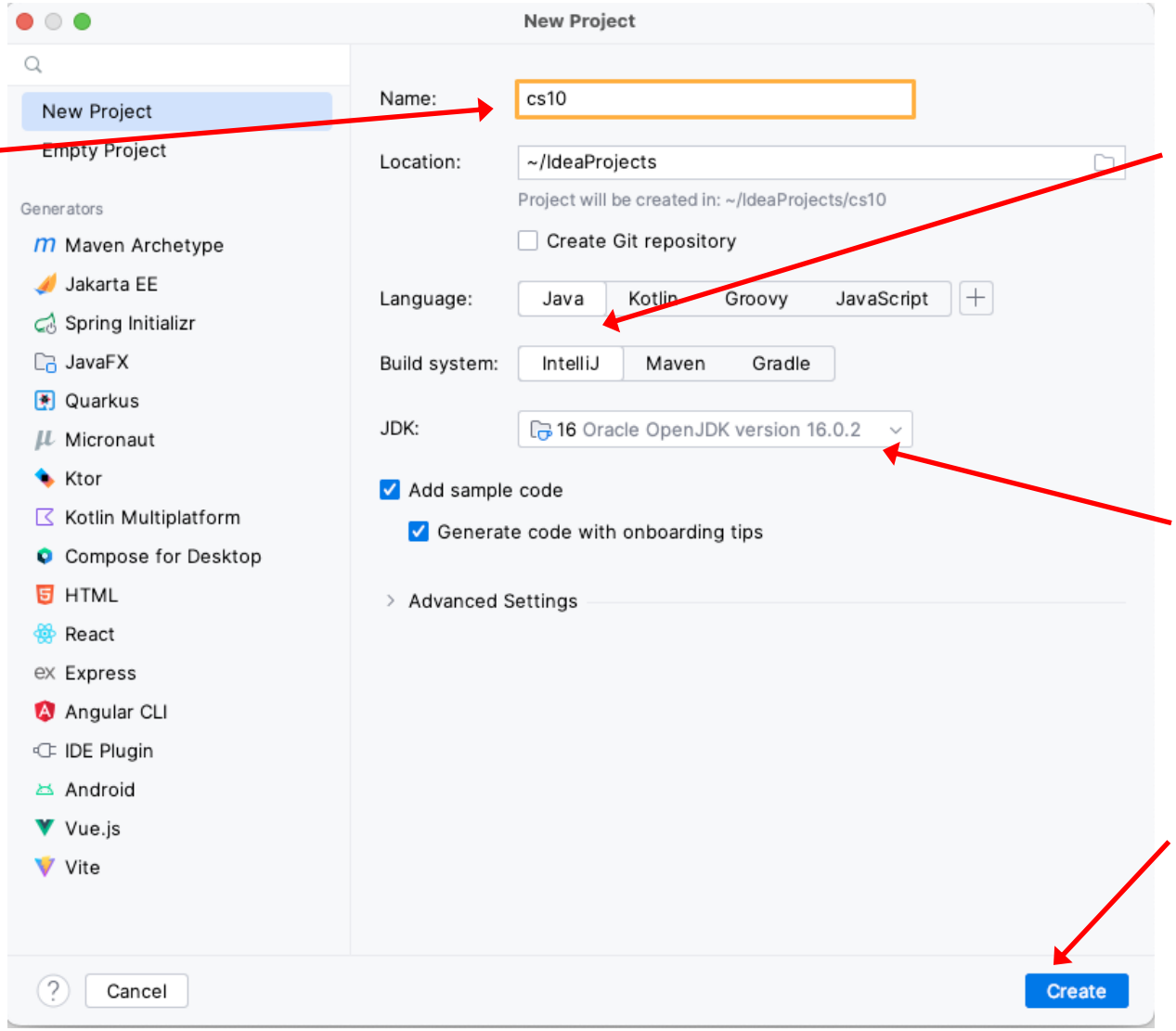
main() is where action starts

Add System.out.println("Hello World") to output to the console

Right click on code and choose "Run <class name>.main()" button to run

1. Create “cs10” Project to hold source code (only need to do this one time)

Start IntelliJ, then select “Create new project” or click File->New->Project



1) Enter project name cs10

2) Choose Java and IntelliJ

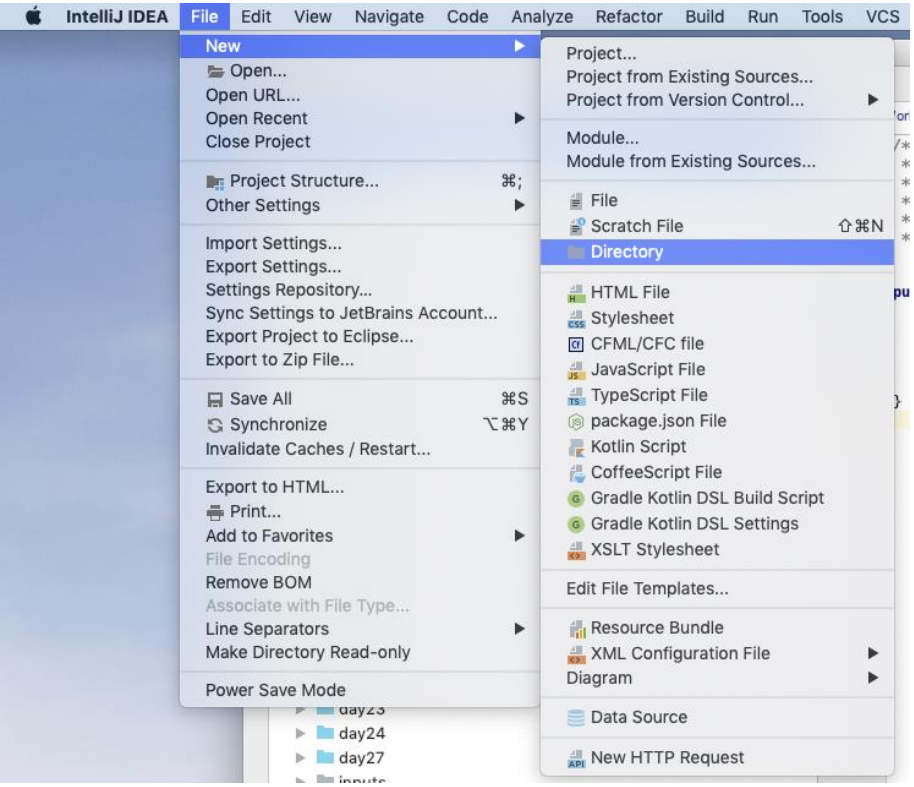
3) Choose Java 16.02

4) Click Create

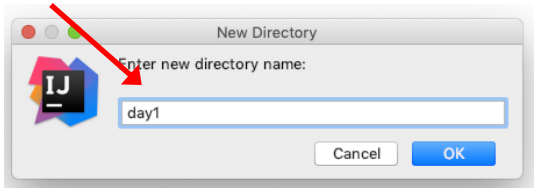
2. Create Source folder to hold your source code for day one of class

Click File->New->Directory to create directory for related code (e.g., “day1” or “PS1”)

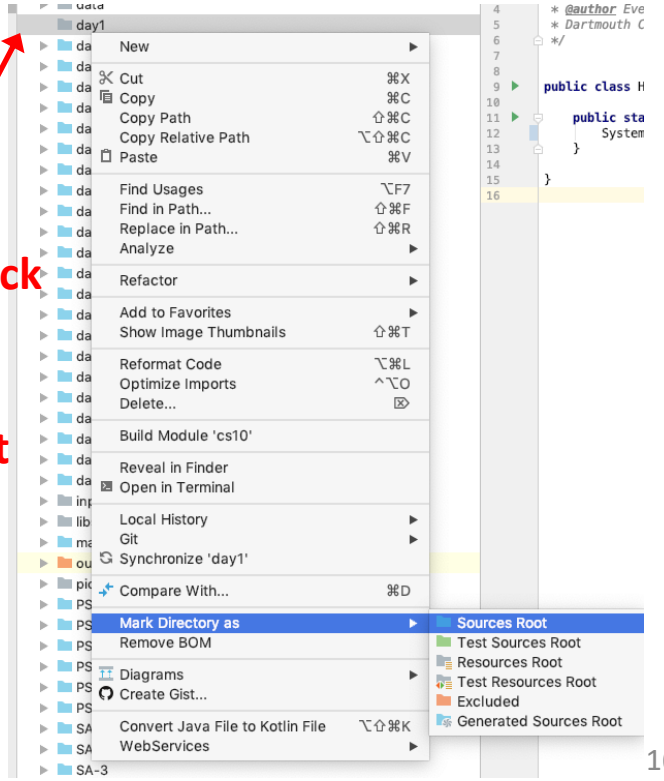
1) Click File->New->Directory



2) Give directory a name



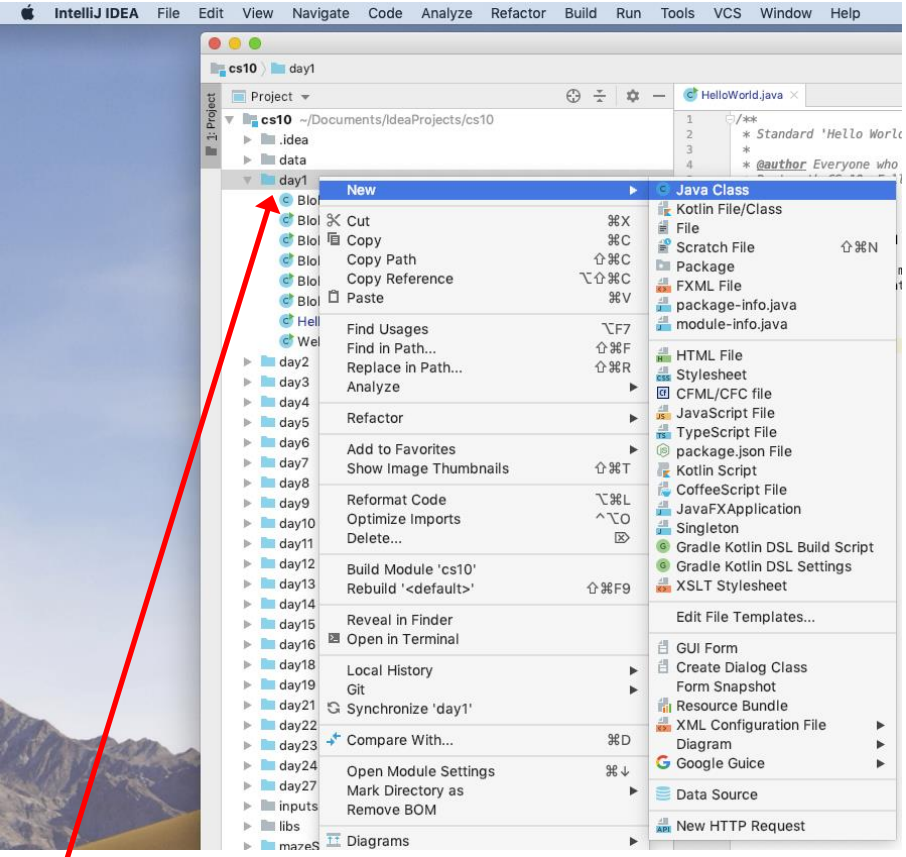
3) Right click on new directory then select “Mark Directory as” and “Sources Root”



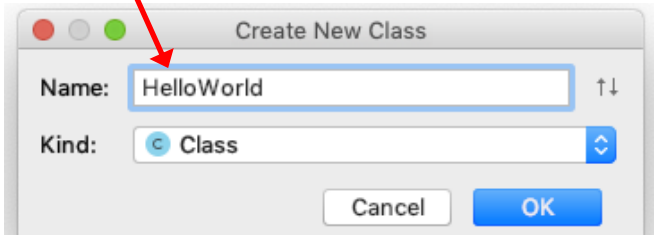
Source folders are a useful way to organize your code (ex. PS1 Source folder contains all code for Problem Set 1)

3. Create new “HelloWorld” class in “day1” source folder

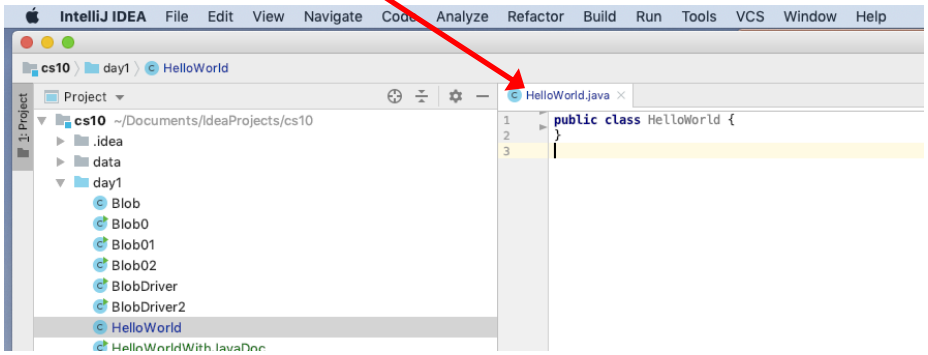
Right click on Source folder and select New->Java Class



2) Give class a name (starting with capital letter)

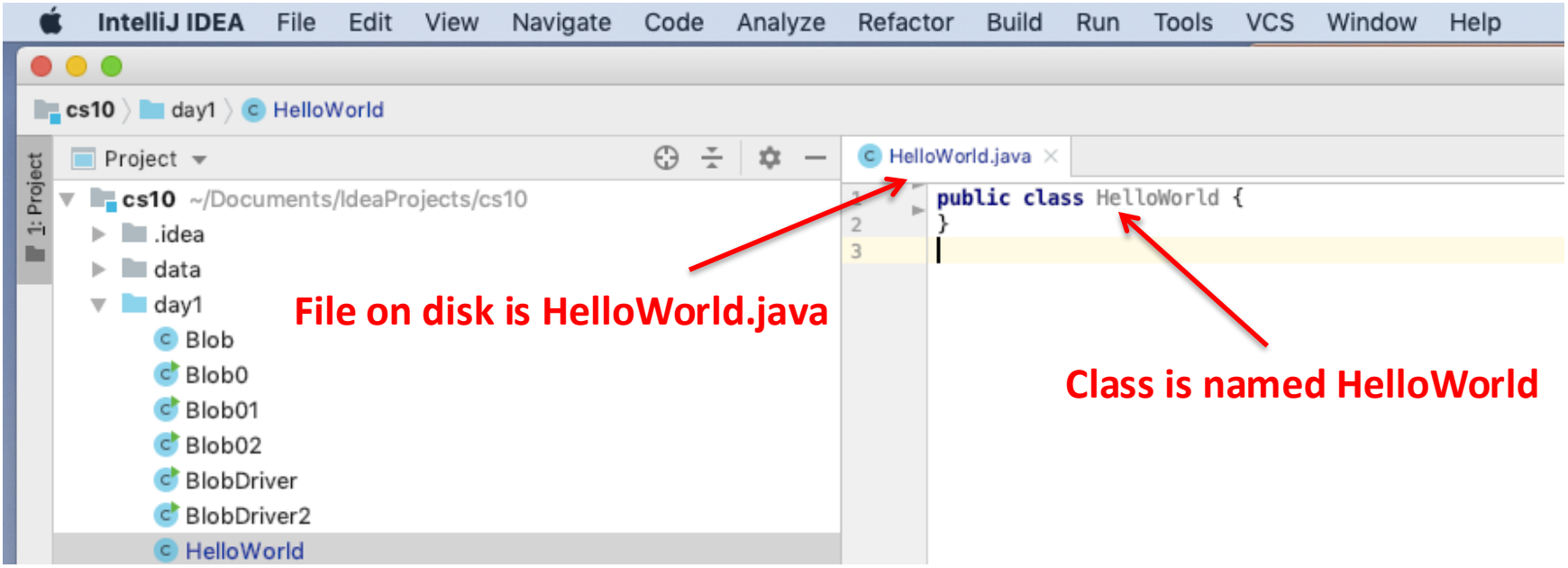


3) IntelliJ creates file on disk (e.g., “HelloWorld.java”) and sets up your new class



1) Right click on Source folder (e.g. “day1”), then select New->Java Class

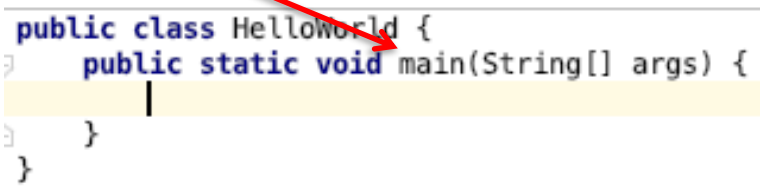
IntelliJ creates HelloWorld.java "boilerplate" code



We can flesh out the boilerplate code to print “Hello World!” to the console

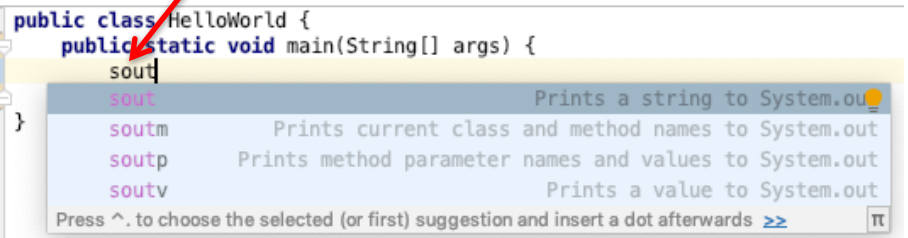
Execution begins at main() method

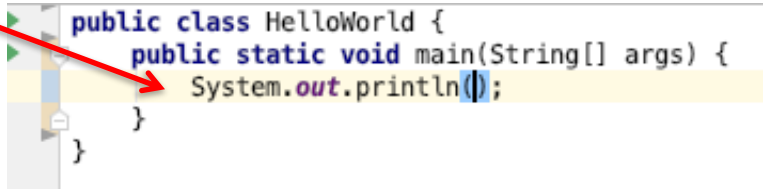
Type “main” then enter and IntelliJ expands to include the main method declaration



In Java a print statement is System.out.println(“text you want to print goes here”);

Type “sout” then enter to have IntelliJ fill out print statement for you (saves a lot of typing!)





We can flesh out the boilerplate code to print “Hello World!” to the console

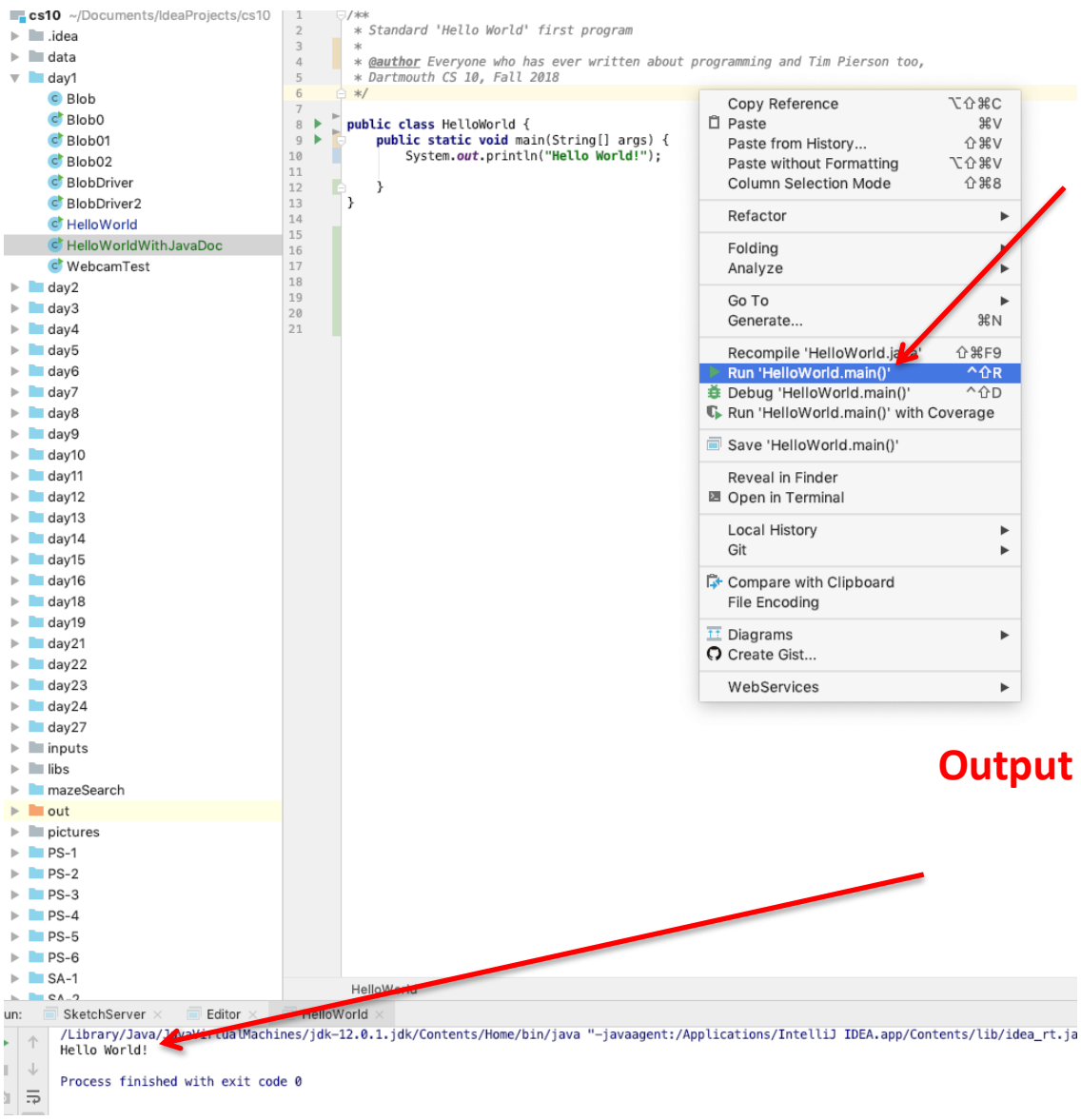
```
/**
 * Standard 'Hello World' first program
 *
 * @author Everyone who has ever written about programming and Tim Pierson too,
 * Dartmouth CS 10, Winter 2025
 */

public class HelloWorld {
   public static void main(String[] args) {
      System.out.println("Hello World!");

   }
}
```

Javadoc

- **Describes program (or method)**
- **Begins with “/**” ends with “*/”**

Add tags such as “@author” or “@param”

Running the program prints “Hello World!” to console

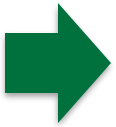


Run program by right clicking on program text and selecting “Run <class name>.main()”

Output appears in console below

Agenda

1. You, me, and this course
2. Why Object-Oriented Programming (OOP)
3. Java intro
4. Variables
5. Arrays



Key points:

1. **In Java we declare each variable and give it a data type**
2. **Data types cannot be changed**

In Python we declare variables but do not say what type of data they hold

Python example **python_variables0.py**

Code

```
print(x)
```

Variable x is not defined, Python has no idea what to print and gives an error message

Output

```
$ python3 python_variables0.py

Traceback (most recent call last):
  File "PythonVariables.py", line 2, in <module>
    print(x)
NameError: name ‘x' is not defined
```

In Python we declare variables but do not say what type of data they hold

Python example **python_variables01.py**

Code

```
x = 5
print(x)
```

Give a value to *x* and Python prints is value

Note: you didn't tell Python what type of data *x* holds, just its value

Python guesses *x* is an integer based on the value assigned (called dynamic or duck typing)

Output

```
$ python3 python_variables01.py
5
```

Python's type function tells us what kind of data the variable holds

Python example **python_variables02.py**

Code

```
x = 5
print(x)
print(type(x))
```

Confirm Python thinks variable *x* is an integer by printing its data type

Confirmed, Python thinks *x* is an integer

Output

```
$ python3 python_variables02.py
5
<class 'int'>
```

In Python a variable's data type can change

Python example **python_variables03.py**

Code

```
x = 5
print(x)
print(type(x))
x = "Hello World"
print(x)
print(type(x))
```

Python allows the type of a variable to change

Still guesses variable type based on value assigned

Now Python thinks *x* is a String

Output

```
$ python3 python_variables03.py
5
<class 'int'>
Hello World
<class 'str'>
```

In Python a variable's data type can change

Python example **python_variables03.py**

Code

```
x = 5
print(x)
print(type(x))
x = "Hello World"
print(x)
print(type(x))
```

Output

```
$ python3 python_variables03.py
5
<class 'int'>
Hello World
<class 'str'>
```

Python allows the type of a variable to change

Still guesses variable type based on value assigned

Now Python thinks *x* is a String

Unlike Python we will tell Java specifically what kind of data a variable holds

Once we give a variable a type, we can't change it to a different type later (e.g., an integer variable can't become a String variable in Java)

In Java, we explicitly say what type of data a variable holds (and can't change it later!)

Common primitive types

Type	Description	Size	Examples
int	Integer values (no decimal component)	32 bits (4 bytes)	-104,…1,2,3…107,…5032
double	Double precision floating point (has decimal component)	64 bits (8 bytes)	-123.45, 1.6
boolean	true or false	1 bit	true, false
char	Characters	16 bits (2 bytes for Unicode)	'a','b',…'z'

Note: String are objects, not primitives
We will discuss objects next class

In Java, we explicitly say what type of data a variable holds (and can't change it!)

JavaVariables0.java

Code

```
public class JavaVariables0 {
   public static void main(String[] args) {
      int x;
      System.out.println("x = "+x);
   }
}
```

Output

Java knows *x* is an integer because we declare it as an integer

We say Java is "strongly typed" because we tell Java what *type* of data a variable holds

When a variable is declared Java allocates memory for it

Here Java allocates memory for one integer (4 bytes)

Java does not initialize local variables

JavaVariables0.java

Code

```
public class JavaVariables0 {
   public static void main(String[] args) {
      int x;
      System.out.println("x = "+x);
   }
}
```

This code looks like it should run, but fails at compile time

Why?

***x* is not given an initial value**

It was also an error in Python when we didn't give *x* a value

Output

```
$ javac JavaVariables0.java
JavaVariables0.java:4: error: variable x might not have been initialized
      System.out.println("x = "+x);
                         ^
1 error
```

Java tells us where to find errors, pay attention to these hints when debugging!

JavaVariables0.java

Code

```
public class JavaVariables0 {
   public static void main(String[] args) {
      int x;
      System.out.println("x = "+x);
   }
}
```

Note: Java tells us what file contained the error

And also tells us what line number Thanks Java!

Output

```
$ javac JavaVariables0.java
JavaVariables0.java:4: error: variable x might not have been initialized
      System.out.println("x = "+x);
                         ^
1 error
```

We must initialize local variables ourselves

JavaVariables01.java

Code

```
public class JavaVariables01 {
   public static void main(String[] args) {
      int x = 5;
      System.out.println("x = "+x);
   }
}
```

Initialize *x* with an integer value

Note: javac from the command line compiles file name provided

Creates a file with a .class extension with the byte code (JavaVariables01.class here)

Output

```
$ javac JavaVariables01.java
$ java JavaVariables01
x = 5
```

java command runs the byte code (no need to provide the .class file extension)

Initialization can happen after a variable is declared

JavaVariables02.java

Code

```
public class JavaVariables02 {
   public static void main(String[] args) {
      int x;
      x = 5;
      System.out.println("x = "+x);
   }
}
```

Not necessary to give local variables a value when declared

Just give the variable a value before using it

Output

```
$ javac JavaVariables02.java
$ java JavaVariables02
x = 5
```

Variables can only hold the type of data they were declared to hold

JavaVariables03.java

Code

```
public class JavaVariables03 {
   public static void main(String[] args) {
      int x;
      x = "Hello world";
      System.out.println("x = "+x);
   }
}
```

Variables must hold the type of data they were declared to hold

Here we can't store a String in an integer variable!

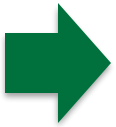
Java tells us where to find the error (file name: line number)

Output

```
$ javac JavaVariables03.java
JavaVariables03.java:4: error: incompatible types: String cannot be converted to int
      x = "Hello world";
          ^
1 error
```

Agenda

1. You, me, and this course
2. Why Object-Oriented Programming (OOP)
3. Java intro
4. Variables
5. Arrays



Key points:

1. **Arrays are just a contiguous block memory, (that's all they are!)**
2. **Arrays are different from Java's ArrayLists and Python's lists! We will soon see how they are different**

We can use multiple variables to store multiple values

MulitpleVariables.java

Code

```
public class MultipleVariables {
   public static void main(String[] args) {
      int score1 = 5, score2 = 7;
      System.out.println(”score1 = "+ score1 + ", score2 = " + score2);
   }
}
```

Say we wanted to track multiple quiz scores

Can declare multiple variables on one line

Here both score1 and score2 are integers, initialized with different values

This approach becomes cumbersome if we want to track many values

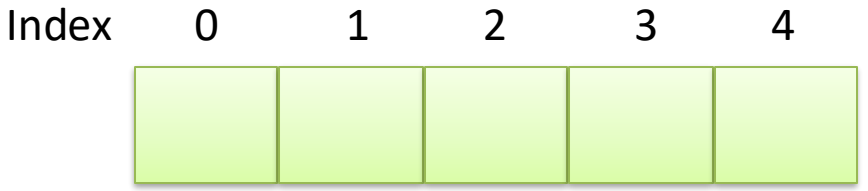
Output

```
$ javac MultipleVariables.java
$ java MultipleVariables
score1 = 5, score2 = 7
```

Arrays provide a better way to store many values in a contiguous block of memory

Code

```
public class MultipleVariablesArray {
   public static void main(String[] args) {
      double[] scores = new double[5]; //store quiz scores
      scores[0] = 10;  //zero indexed in Java
      scores[1] = 3.2;
      scores[2] = 6.5;
      scores[3] = 7.8;
      scores[4] = 8.8;  //valid indices are 0..4
      scores[5] = 9;    //error, index out of bounds!
      System.out.println(scores);
   }
}
```



Use an array to store multiple quiz scores

OS allocates a contiguous block of memory

Here enough room to hold 5 doubles (5 doubles * 8 bytes/double = 40 bytes)

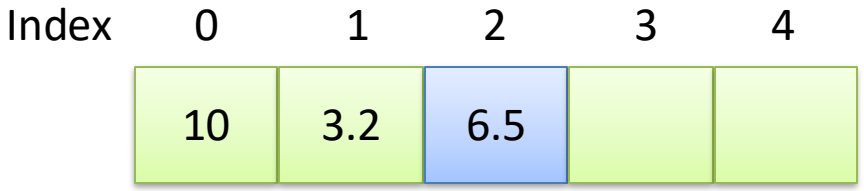
Arrays are zero-indexed in Java (unlike Matlab)

Keyword *new* allocates memory for array (we will see soon this is an object)

Finding an index in an array is two math operations: 1 addition and 1 multiplication

Code

```
public class MultipleVariablesArray {
   public static void main(String[] args) {
      double[] scores = new double[5]; //store quiz scores
      scores[0] = 10;  //zero indexed in Java
      scores[1] = 3.2;
      scores[2] = 6.5;
      scores[3] = 7.8;
      scores[4] = 8.8;  //valid indices are 0..4
      scores[5] = 9;    //error, index out of bounds!
      System.out.println(scores);
   }
}
```



Set values in the array with square brackets

First item is at index 0

Where is the last item?

At size-1 (index 4 here because array size is 5)

Java throws an exception if try to access memory outside the contiguous block

Code

```
public class MultipleVariablesArray {
   public static void main(String[] args) {
      double[] scores = new double[5]; //store quiz scores
      scores[0] = 10;  //zero indexed in Java
      scores[1] = 3.2;
      scores[2] = 6.5;
      scores[3] = 7.8;
      scores[4] = 8.8;  //valid indices are 0..4
      scores[5] = 9;  //error, index out of bounds!
      System.out.println(scores);
   }
}
```

Index	0	1	2	3	4
	10	3.2	6.5	7.8	8.8

Java throws an exception if you try to access an element before or after the array's block of memory

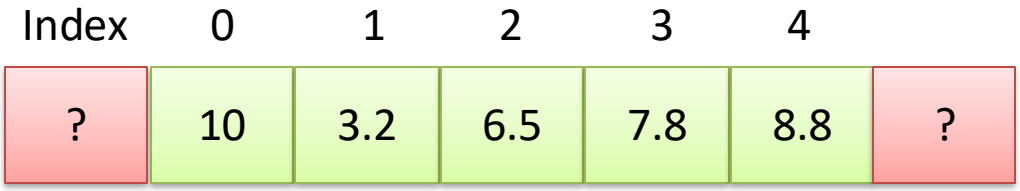
Output

```
$ javac MultipleVariablesArray.java
S java MultipleVariablesArray
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 5
at MultipleVariablesArray.main(MultipleVariablesArray.java:9)
```

Memory outside the contiguous block may be used for other purposes

Code

```
public class MultipleVariablesArray {
   public static void main(String[] args) {
      double[] scores = new double[5]; //store quiz scores
      scores[0] = 10;  //zero indexed in Java
      scores[1] = 3.2;
      scores[2] = 6.5;
      scores[3] = 7.8;
      scores[4] = 8.8;  //valid indices are 0..4
      scores[5] = 9;    //error, index out of bounds!
      System.out.println(scores);
   }
}
```



Can you assume the memory before or after the allocated block is available for your use?

NO! The OS allocated the block of memory for the array and may be using the memory before or after for other purposes!

C programmers can access memory before or after, this often causes bugs!

Output

```
$ javac MultipleVariablesArray.java
S java MultipleVariablesArray
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 5
at MultipleVariablesArray.main(MultipleVariablesArray.java:9)
```

Printing an array prints the starting memory address

Code

```
public class MultipleVariablesArray {
   public static void main(String[] args) {
      double[] scores = new double[5]; //store quiz scores
      scores[0] = 10;  //zero indexed in Java
      scores[1] = 3.2;
      scores[2] = 6.5;
      scores[3] = 7.8;
      scores[4] = 8.8;  //valid indices are 0..4
      //scores[5] = 9;    //error, index out of bounds!
      System.out.println(scores);
   }
}
```

Index	0	1	2	3	4
	10	3.2	6.5	7.8	8.8

By default, printing an array prints a value based on the array's starting memory address

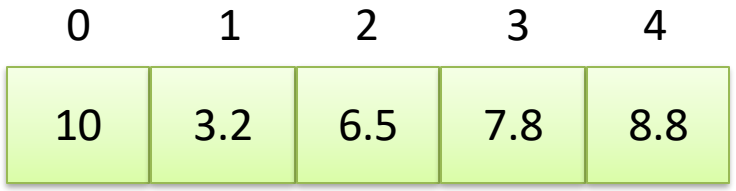
Output

```
$ javac MultipleVariablesArray.java
S java MultipleVariablesArray
[D@1dbd16a6
```

One way to loop over array elements is to use a C-style for loop

Code

```
public class MultipleVariablesArray {
   public static void main(String[] args) {
      int numberOfScores = 5;
      double[] scores = new double[numberOfScores]; //store quiz scores
      scores[0] = 10;  //zero indexed in Java
      scores[1] = 3.2;
      scores[2] = 6.5;
      scores[3] = 7.8;
      scores[4] = 8.8;  //valid indices are 0..4
      //scores[5] = 9;    //error, index out of bounds!
      System.out.println(scores);

      System.out.print("[");
      for (int i= 0; i < numberOfScores-1; i++) {
         System.out.print(scores[i] + ", ");
      }
      System.out.println(scores[numberOfScores-1] + "]");
   }
}
```



Commonly use a variable to declare array size

C-style for loop

Three components:

1. **Initialization**
2. **Conditional**
3. **Increment**

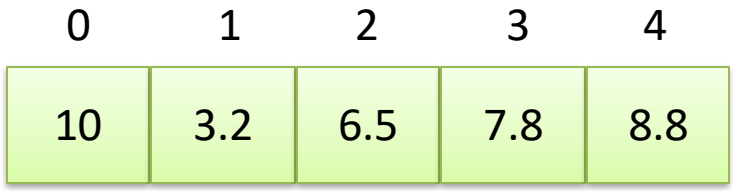
Output

```
$ javac MultipleVariablesArray.java
$ java MultipleVariablesArray
D@1dbd16a6
[10.0, 3.2, 6.5, 7.8, 8.8]
```

One way to loop over array elements is to use a C-style for loop

Code

```
public class MultipleVariablesArray {
   public static void main(String[] args) {
      int numberOfScores = 5;
      double[] scores = new double[numberOfScores]; //store quiz scores
      scores[0] = 10;  //zero indexed in Java
      scores[1] = 3.2;
      scores[2] = 6.5;
      scores[3] = 7.8;
      scores[4] = 8.8;  //valid indices are 0..4
      //scores[5] = 9;    //error, index out of bounds!
      System.out.println(scores);

      System.out.print("[");
      for (int i= 0; i < numberOfScores-1; i++) {
         System.out.print(scores[i] + ", ");
      }
      System.out.println(scores[numberOfScores-1] + "]");
   }
}
```



Access array element at index *i* using square brackets

Note: using *print* not *println* here *println* adds a new line character

Output

```
$ javac MultipleVariablesArray.java
$ java MultipleVariablesArray
D@1dbd16a6
[10.0, 3.2, 6.5, 7.8, 8.8]
```

Java also has multidimensional arrays

Code

MultidimensionalArray.java

```
public class MultidimensionalArray {
   public static void main(String[] args) {
      int numberOfStudents = 10;
      int numberOfQuizes = 5;
      double scores[][] = new double[numberOfStudents][numberOfQuizes];

      //set score for student 3 on quiz 2
      scores[2][1] = 9.2; //remember zero-indexing!

      //print all scores
      int quiz;
      for (int student = 0; student < numberOfStudents; student++) {
         for (quiz = 0; quiz < numberOfQuizes-1; quiz++) {
            System.out.print(scores[student][quiz] + ", ");
         }
         System.out.println(scores[student][quiz]);
      }
   }
}
```

Store quiz scores for several students in 2-dimensional array
One row for each student
One column for each quiz

Remember zero indexing!
Student 3 is at index 2
Quiz 2 is at index 1

Can declare variable outside for loop so its scope goes beyond for loop

Nested loops
Loop over each student
loop over each quiz
print quiz score for student

Because quiz declared outside for loop, it is still in scope here (would be out of scope if declared as part of for loop)

Arrays holding numeric values are initialized to zero

Code

MultidimensionalArray.java

```
public class MultidimensionalArray {
   public static void main(String[] args) {
      int numberOfStudents = 10;
      int numberOfQuizes = 5;
      double scores[][] = new double[numberOfStudents][numberOfQuizes];

      //set score for student 3 on quiz 2
      scores[2][1] = 9.2; //remember zero-indexing!

      //print all scores
      int quiz;
      for (int student = 0; student < numberOfStudents; student++) {
         for (quiz = 0; quiz < numberOfQuizes-1; quiz++) {
            System.out.print(scores[student][quiz] + ", ");
         }
         System.out.println(scores[student][quiz]);
      }
   }
}
```

Java initializes numeric array values to zero

Value set

Output

```
$ javac MultidimensionalArray.java
$ java MultidimensionalArray
0.0, 0.0, 0.0, 0.0, 0.0
0.0, 0.0, 0.0, 0.0, 0.0
0.0, 9.2, 0.0, 0.0, 0.0
0.0, 0.0, 0.0, 0.0, 0.0
0.0, 0.0, 0.0, 0.0, 0.0
0.0, 0.0, 0.0, 0.0, 0.0
0.0, 0.0, 0.0, 0.0, 0.0
0.0, 0.0, 0.0, 0.0, 0.0
0.0, 0.0, 0.0, 0.0, 0.0
0.0, 0.0, 0.0, 0.0, 0.0
```

Short Assignment 0 (SA-0) is out, complete survey before 8:00am tomorrow

SA-0

Find assignment on Canvas

1. Take course survey to understand your background and assign you to a section
2. Set up development environment
   - Instructions and screen shots provided on website
   - We will use IntelliJ IDEA for this course
3. Create your first Java class
4. Read and acknowledge course policies and honor code

Complete survey **before 8:00am tomorrow** (or risk getting assigned to inconvenient section time!)

Key points

1. Java is a compiled, strongly typed language
2. In Java we declare each variable and give it a data type
3. Data types cannot be changed
4. Arrays are just a contiguous block memory, (that's all they are!)
5. Arrays are different from Java's ArrayLists and Python's lists! We will soon see how they are different