CS 10: Problem solving via Object Oriented Programming

Welcome to the class!

Class goal, syllabus, starting on OOP and Java

Today main learning goals

1. Get to know each other
2. Align expectations about CS10, outcomes, teaching methods/tools, coursework, collaboration
3. Describe the pillars of Object-Oriented Programming (OOP)
4. Identify some main differences between Python and Java
5. Program hello world
6. Use variables and arrays

Agenda

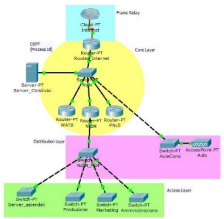
1. You, the teaching team, and this course

2. Object Oriented Programming (OOP)

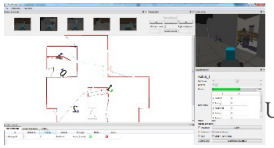
3. Java intro
    1. Hello world
    2. Variables
    3. Arrays

My background: roboticist

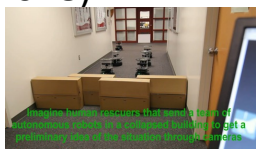
B.Sc. in CS&Eng (2006-2009)



M.Sc. (2009-2011) and Ph.D. in CS&Eng (2012-2015)



Postdoc/Research faculty (2015-2018)



Dartmouth Reality and Robotics Lab (2018-present)

Taught courses
Problem Solving via Object Oriented Programming (COSC10)
Artificial Intelligence (COSC76/276)
Principles of Robot Design and Programming (COSC81/181)
Machine Learning for Robotics (COSC89/189)
Multirobot Systems (COSC69/169)
Robotics Perception Systems (COSC69/169)

Teaching team

Alberto Quattrini Li

Julien Blanchet

Aimen Abdulaziz

Akshee Chopra

Alex Nanda

Ava Weinrot

Bill Zheng

Eren Saglam

Leina Sato

Jackson Easley

Marina Frayre

Nand Patel

Renata Hoh

Sarah Levesque

Shahidullah Dost

Trung Nguyen

Yelynn Kim

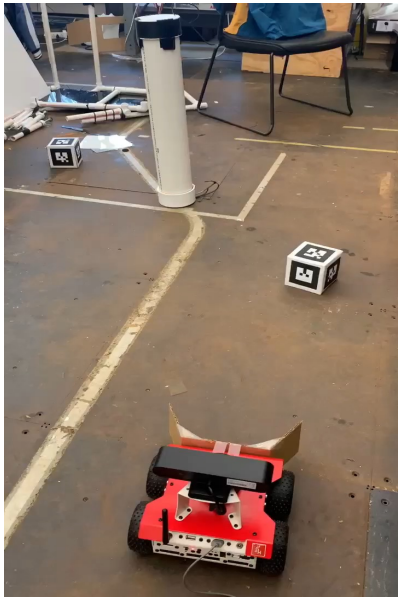
Warren Shepard

Your background

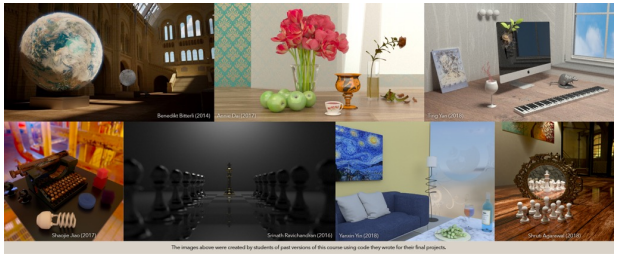
- How did you satisfy the pre-reqs?
  - CS 1
  - ENGS 20
  - AP exam
  - Other
- What's your future plan?
  - CS majors?
  - Minors?
  - Not sure?
- How many of you programmed in Java before?

Primary objective of the course

Reinforce the foundational perspective and skills needed to develop **computational solutions** to **real interesting problems.**



CS81/181 Robotics



CS87/187 Rendering algorithms

CS83/183 Computer Vision

And many others, including Artificial intelligence, Machine learning, Bioinformatics, …
https://web.cs.dartmouth.edu/undergraduate/undergraduate-courses

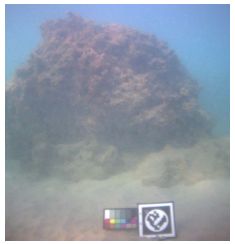
Main learning outcomes:

Problem solving via *Object Oriented Programming*

(not simply how to program in Java)

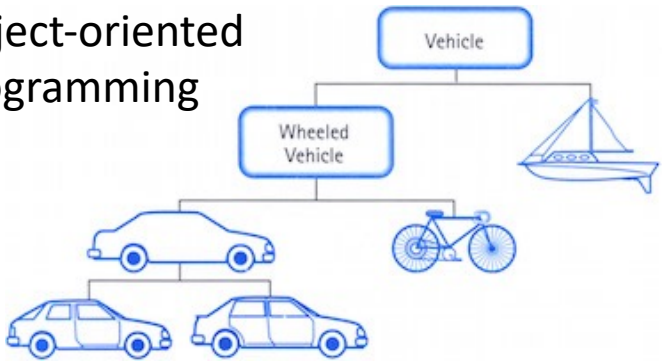
Problem → Program → Output

e.g., Image processing

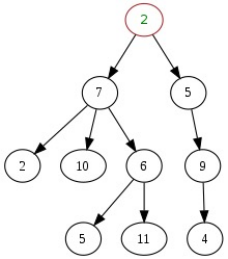


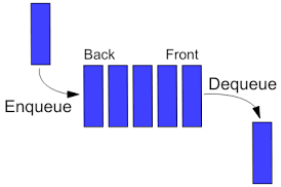
?

Object-oriented programming



Data structures



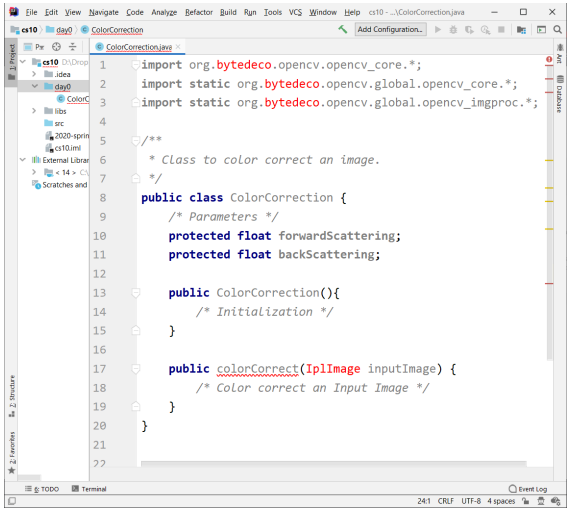


Algorithms

```
procedure DFS-iterative(G, v) is
    let S be a stack
    S.push(v)
    while S is not empty do
        v = S.pop()
        if v is not labeled as discovered then
            label v as discovered
            for all edges from v to w in G.adjacentEdges(v) do
                S.push(w)
```

…

Learn by solving real problems, implementing and debugging!



Java

IntelliJ IDEA

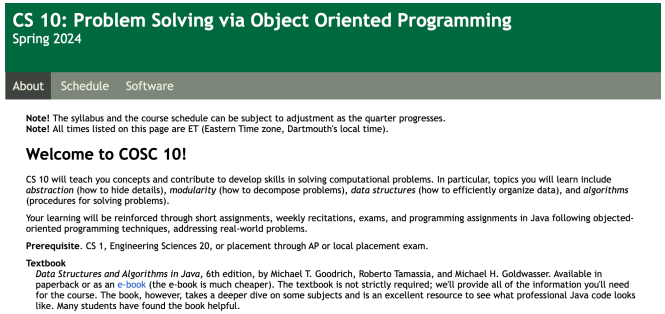
Learning resources: multimodal

Lectures for covering concepts

- 12 time slot MWF 12:50-1:55 pm EDT (recording posted afterwards)
  X Tu 1:20-2:10pm EDT (see on Canvas if used)
- Feel free to ask questions

Recitation section meetings for hands-on:

- 1 hour/week (starting this week)



CS 10: Problem Solving via Object Oriented Programming
Spring 2024

About Schedule Software

Note! The syllabus and the course schedule can be subject to adjustment as the quarter progresses.
Note! All times listed on this page are ET (Eastern Time zone, Dartmouth's local time).

Welcome to COSC 10!

CS 10 will teach you concepts and contribute to develop skills in solving computational problems. In particular, topics you will learn include *abstraction* (how to hide details), *modularity* (how to decompose problems), *data structures* (how to efficiently organize data), and *algorithms* (procedures for solving problems).

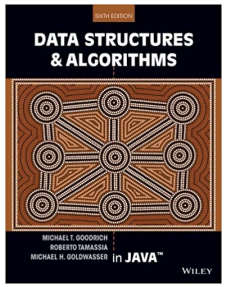
Your learning will be reinforced through short assignments, weekly recitations, exams, and programming assignments in Java following objected-oriented programming techniques, addressing real-world problems.

Prerequisite. CS 1, Engineering Sciences 20, or placement through AP or local placement exam.

Textbook
Data Structures and Algorithms in Java, 6th edition, by Michael T. Goodrich, Roberto Tamassia, and Michael H. Goldwasser. Available in paperback or as an e-book (the e-book is much cheaper). The textbook is not strictly required; we'll provide all of the information you'll need for the course. The book, however, takes a deeper dive on some subjects and is an excellent resource to see what professional Java code looks like. Many students have found the book helpful.

Website for notes, slides, and assignment instructions (https://www.cs.dartmouth.edu/cs10/)

- Syllabus
- Schedule



Textbook for additional examples and explanations: Data Structures & Algorithms in Java , 6th ed, by Goodrich, Tamassia, and Goldwasser

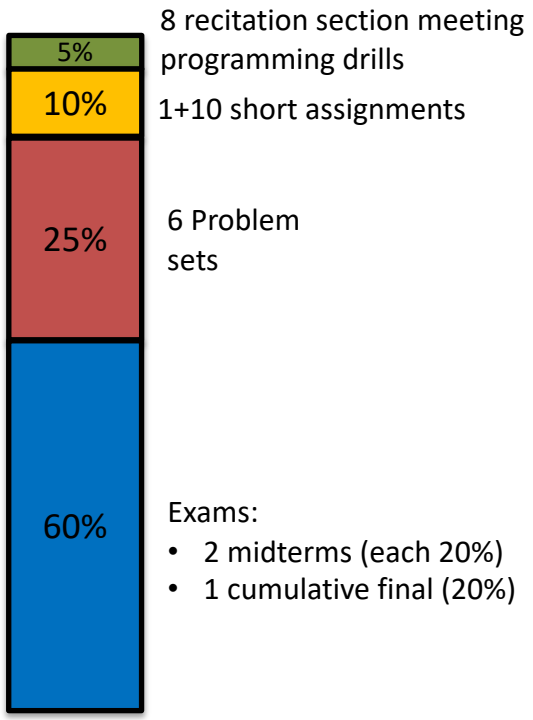
Learning resources: assessment

To assess your learning, you will:

1. Solve recitation section meeting programming drills and short assignments by writing relatively small code snippets to consolidate the concepts learned every week in class.
2. Solve problem sets by writing programs to apply the concepts learned in class to real-world problems.
   - You can find one problem set collaborator
3. Answer exam questions.

Please see the details on the syllabus (late policy, grade, extra credit, …)
https://www.cs.dartmouth.edu/cs10/#coursework

Please reach out if you're falling behind – we're happy to help



Learning resources: how to get help

(access via Canvas)

- Few channels to simplify the use
  - #classes-discussion: for questions and discussions on the concepts covered in the class.
  - #help-assignments: for questions and discussions on the assignments.
- Help each other (please remember the **honor code**; if in doubt, ask)
  - Don't post solution!
- The teaching team will typically respond within 24 hours
  - Use the public channels first
  - Don't hesitate to ping in case we missed your message
- Anonymous messages are possible with `/anonymous` command
- Let's build an inclusive community!

Office hours (~3 hours/week each member of the teaching team)

- Profs. and TAs
  - Each of you will have a reference TA to have a long-term support
- ECSC building
  - Zoom as needed

Learning resources: other tools used

Canvas (https://canvas.dartmouth.edu/courses/65523)

- Course announcements
- Calendar with office hours
- Section assignments
- Link to Slack
- Link to Gradescope
- Link to Panopto
- Record of assignment and current overall grading

Recordings on Panopto folder (access through Canvas)

Submissions via Gradescope (access through Canvas) for consistent grading

If you have any problem, please let us know

Tentative schedule

Please keep an eye as everything will be posted and updated here

w1	Day1	Mar 25	intro	class goal, syllabus, Java basics	1	SA-0	
	Day2	Mar 27	encapsulation	classes, instance variables, contructors, overloading	2	SA-1	SA-0 (Survey due Mar 25, 11:59pm)
	Day3	Mar 29	inheritance	base classes, subclasses, overriding		SA-2	SA-1
w2	Day4	Apr 1	graphics	buffered images, video		SA-3	SA-2
	Day5	Apr 3	abstraction	abstract data types, asymptotic notation	7.1	PS-1	SA-3
	Day6	Apr 5	lists	linked list implementation	3.2	SA-4	
w3	Day7	Apr 8	lists 2	growing array implementation	7.2, 3.1.1, 4-4.3, 7.4		SA-4
	Day8	Apr 10	hierarchies	trees and recursion	8	PS-2	PS-1
		Apr 12	midterm 1	6:00pm - 8:00pm Room TBA			
w4	Day9	Apr 15	hierarchies 2	binary search trees	11.1		
	Day10	Apr 17	hierarchies 3	balance, 2-3-4 trees, red/black trees	Ch 11.2, 11.5, 11.6	SA-5	
	Day11	Apr 19	info retrieval	maps, sets	10.1	SA-6	SA-5 & PS-2
w5	Day12	Apr 22	hashing	hash functions, tables	10.2	SA-7	SA-6
	Day13	Apr 24	keeping order	stacks, queues	6	SA-8	SA-7
	Day14	Apr 26	prioritizing	priority queues, heaps	9.1 - 9.4	PS-3	SA-8
		Apr 29	midterm 2	6:00pm - 8:00pm Room TBA			
w6	Day15	May 1	relationships	graphs	14.1, 14.2	SA-9	
	Day16	May 3	graph traversal	breadth- and depth-first search	14.3	PS-4	PS-3
w7	Day17	May 6	shortest paths	Dijkstra's algorithm, A* search	14.6		SA-9
	Day18	May 8	pattern matching	finite automata			
	Day19	May 10	pattern recognition	hidden Markov models		PS-5	PS-4
w8	Day20	May 13	(no class, video) web services	acronym soup (URL, REST, XML, GUI)	Java tutorials		
	Day21	May 15	(no class, video) client/server	sockets, threads	Java tutorials		
	Day22	May 17	(no class, video) synchronization	synchronized blocks, monitors, semaphores	Java tutorials	SA-10 & PS-6	
w9	Day23	May 20	producer/consumer	streams	Java tutorials		PS-5 & SA-10
	Day24	May 22	string finding	Boyer-Moore, tries, suffix trees	Ch 13.2, 13.3		
		May 24	review	Final recap			
w10		May 27	Memorial day				
		May 29	review	ask questions about the final			PS-6
		Jun 3	final exam	3:00pm - 6:00pm Cummings 100			

https://www.cs.dartmouth.edu/cs10/schedule.html

Short Assignment (SA-0) out, complete survey before 11:59pm today

- Find it on Gradescope (via Canvas)
- Take course survey to understand your background and assign you to a section
- Set up development environment (IntelliJ IDEA) and get Hello World running
  - Instructions and screen shots provided on website https://cs.dartmouth.edu/cs10/software.html
- Create your first Java class
- Read and acknowledge course policies and honor code
  - If you have any questions or comments, please feel free to leave a comment
- Complete survey ASAP **before 11:59pm today** (or risk getting assigned to inconvenient section time!)

Learning resources: advice

Successful learning in the class is typically associated with

- Reading the material recommended
- Actively Participating in the class
- Starting all assignments as soon as they are out
- Reaching out for help immediately when stuck

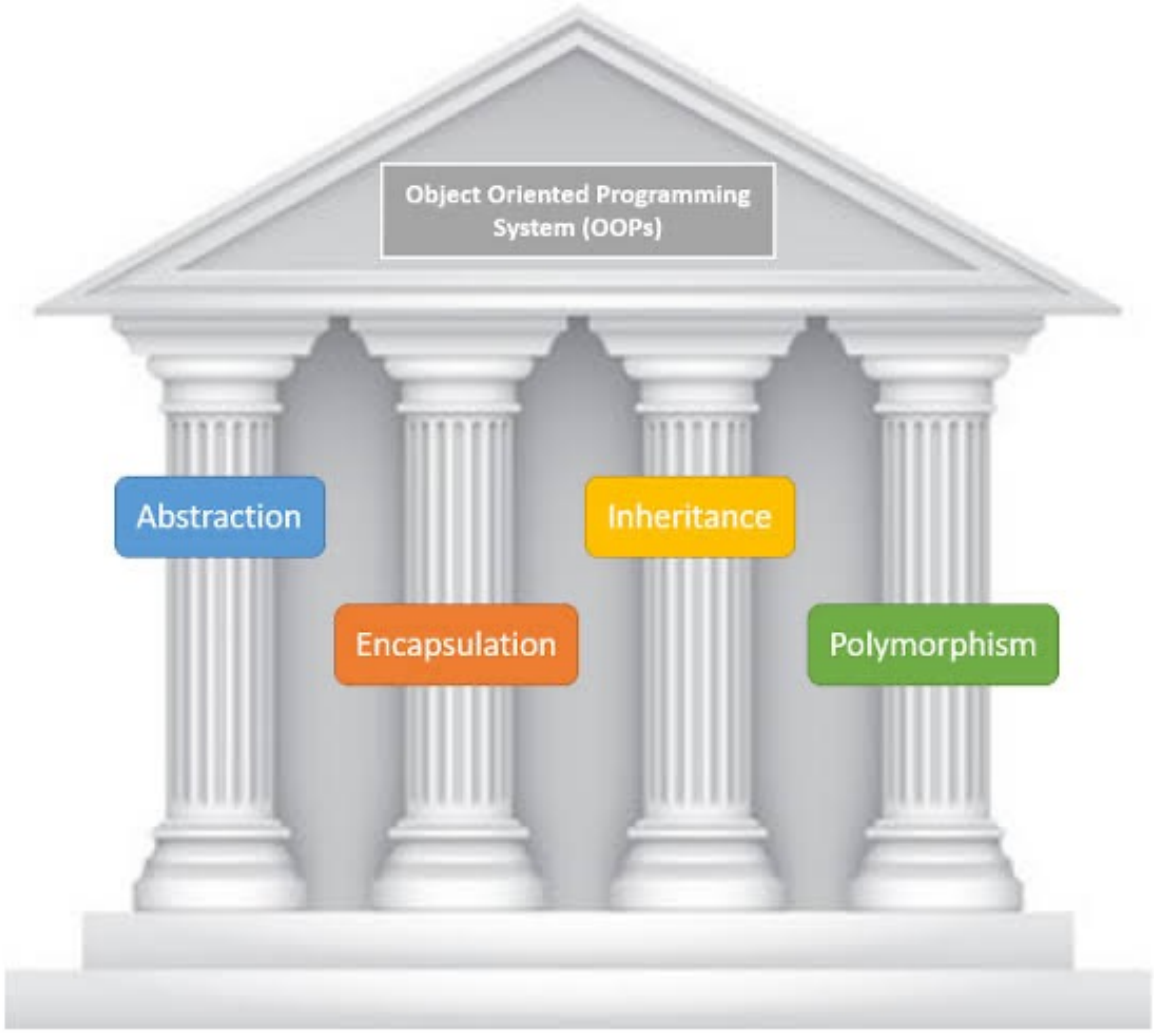
We will succeed if we work together

- Please talk to us if you are running behind or if you have any questions/comments

Agenda

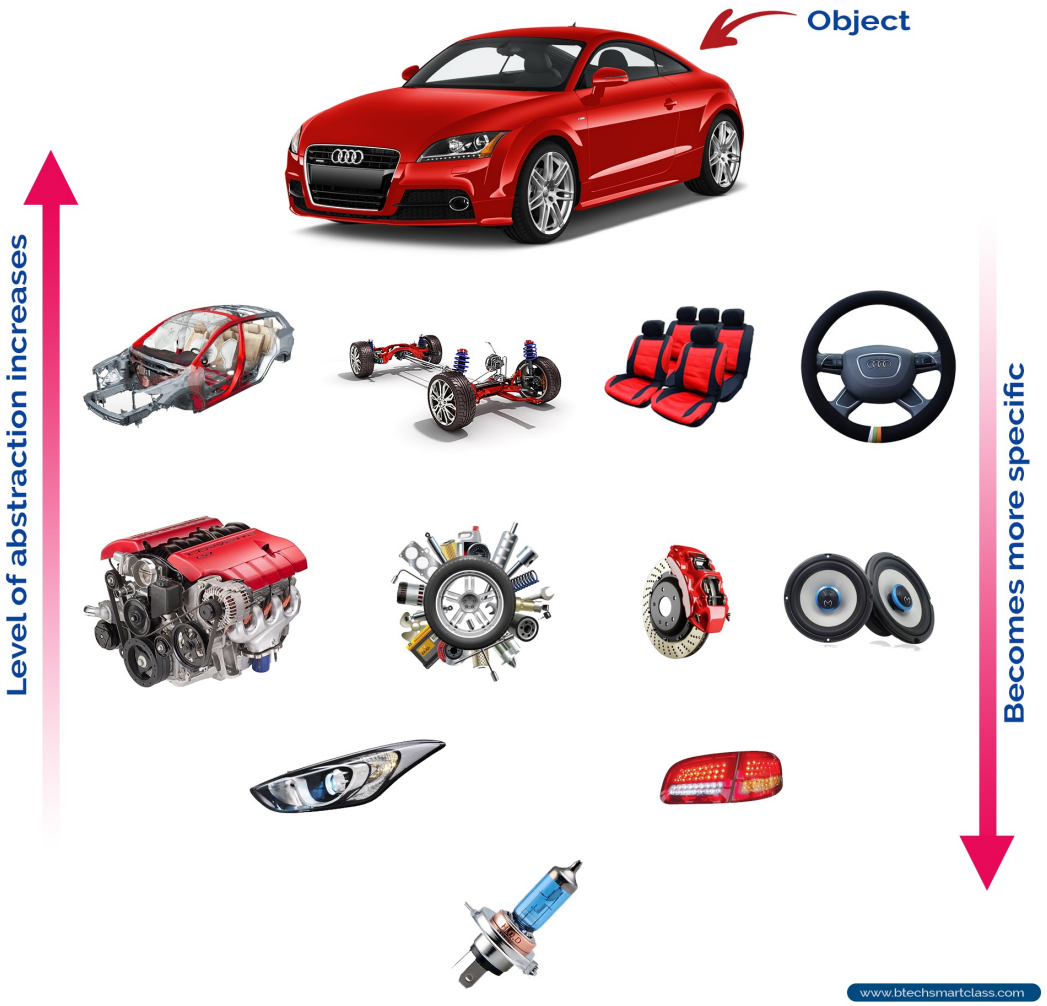
1. You, me, and this course
2. Object Oriented Programming (OOP)
3. Java intro
    1. Hello world
    2. Variables
    3. Arrays

OOP four main pillars for robust, adaptable and reusable code

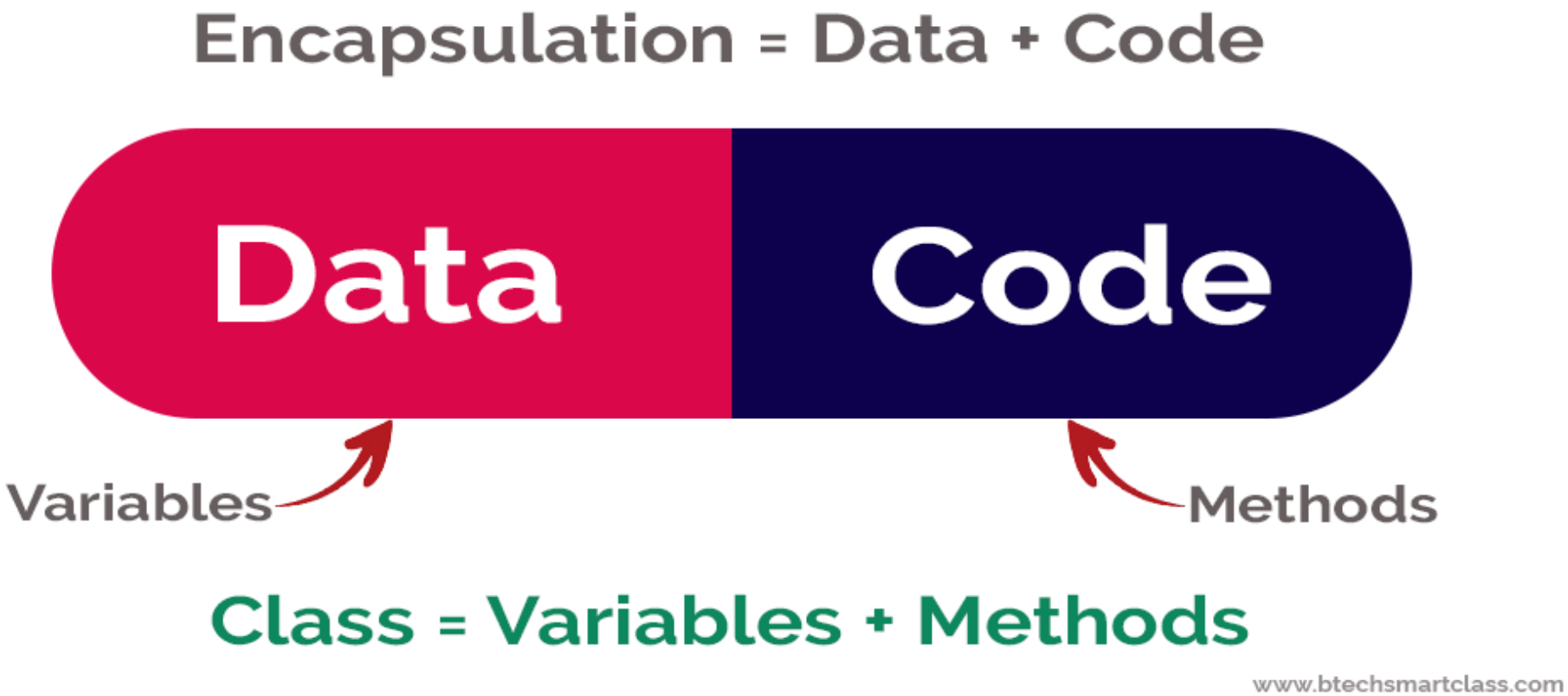


https://learningcode.tech.blog/2022/07/08/oops-pillars/

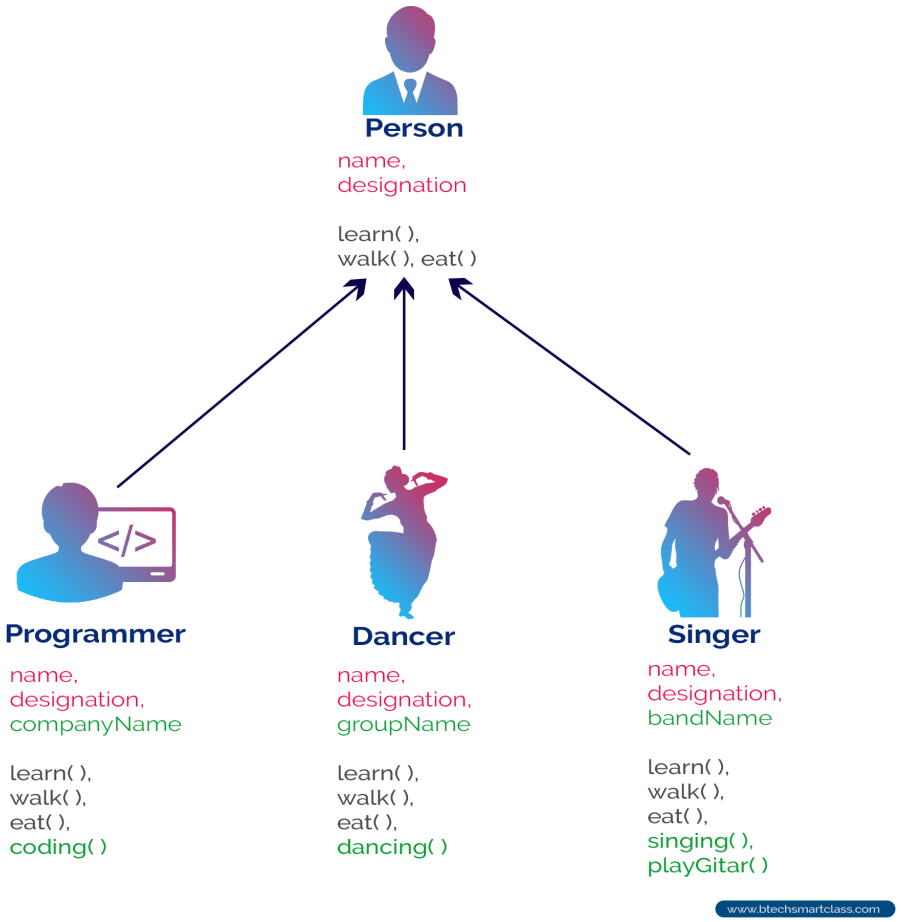
Abstraction



Encapsulation

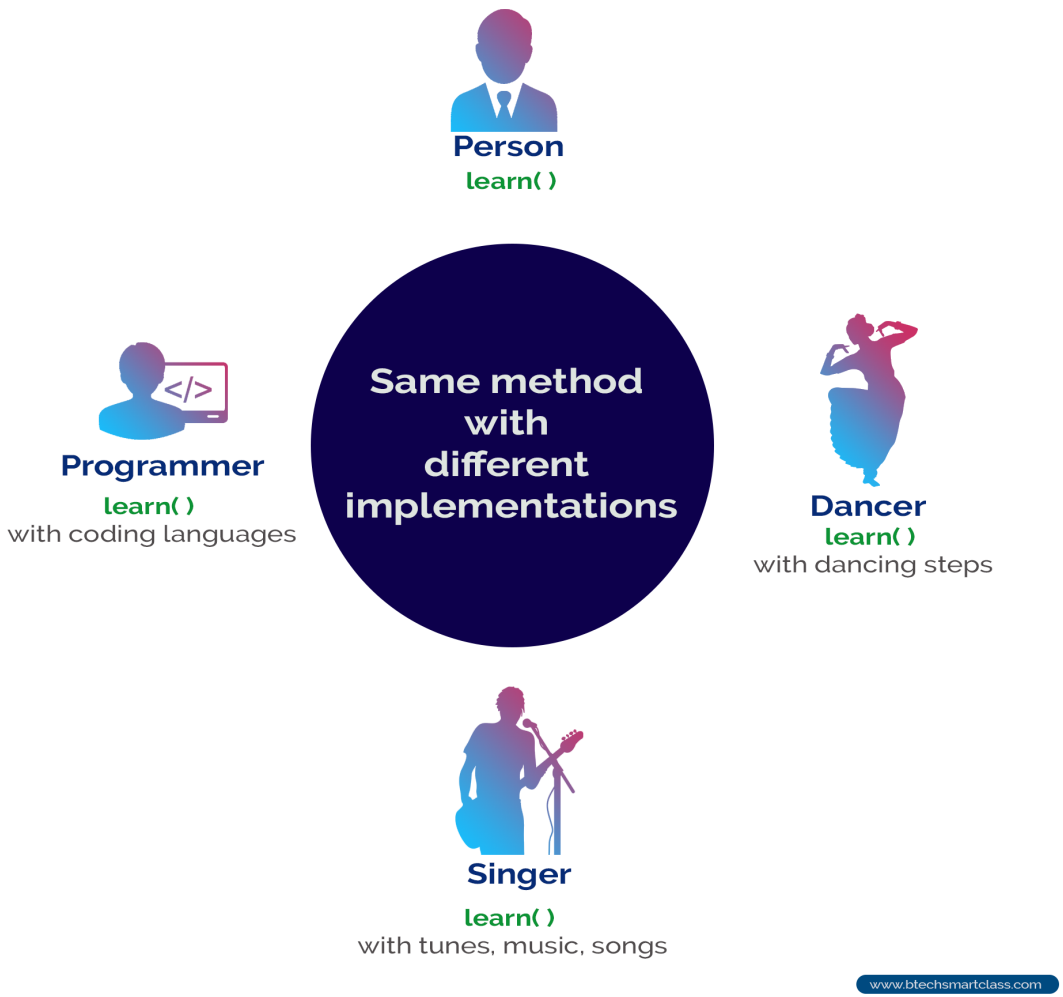


Inheritance



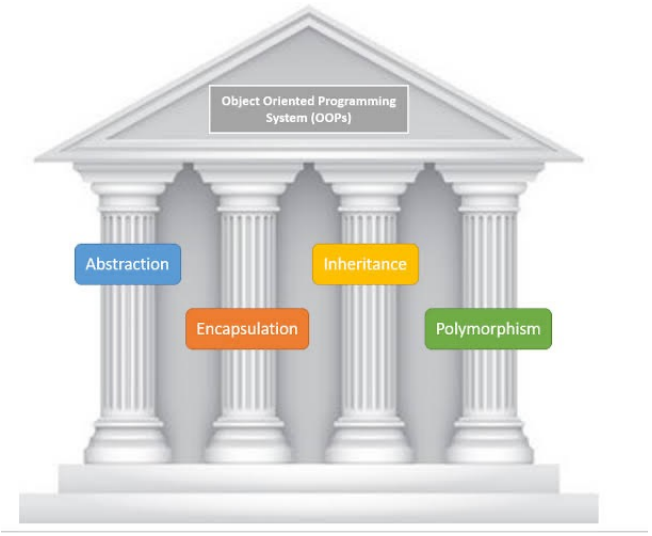
reduces code redundancy

Polymorphism



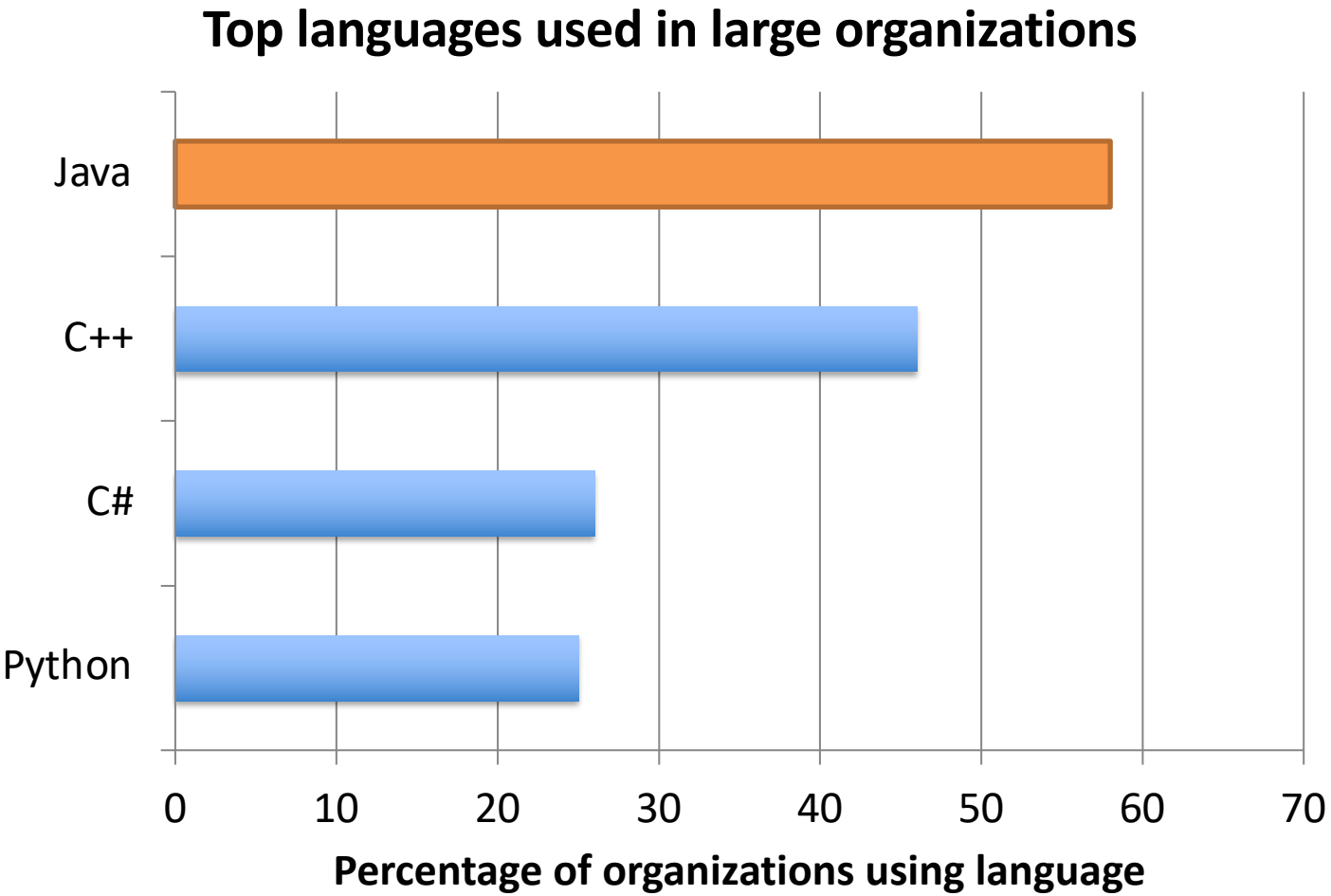
Why is OOP in general so popular?

- Well-organized: Relative data and functions are grouped in the same object.
- Code reusability: Abstraction allows you to reuse code throughout a project.
- Testing and debugging: The self-contained nature of OOP can make testing and debugging easier.
- Promotes collaboration among developers: The use of classes, objects, and well-defined interfaces facilitates teamwork.
- Improves code readability and documentation: Makes it easier for developers to understand and contribute to the codebase



https://learningcode.tech.blog/2022/07/08/oops-pillars/

OOP is popular, especially in large organizations



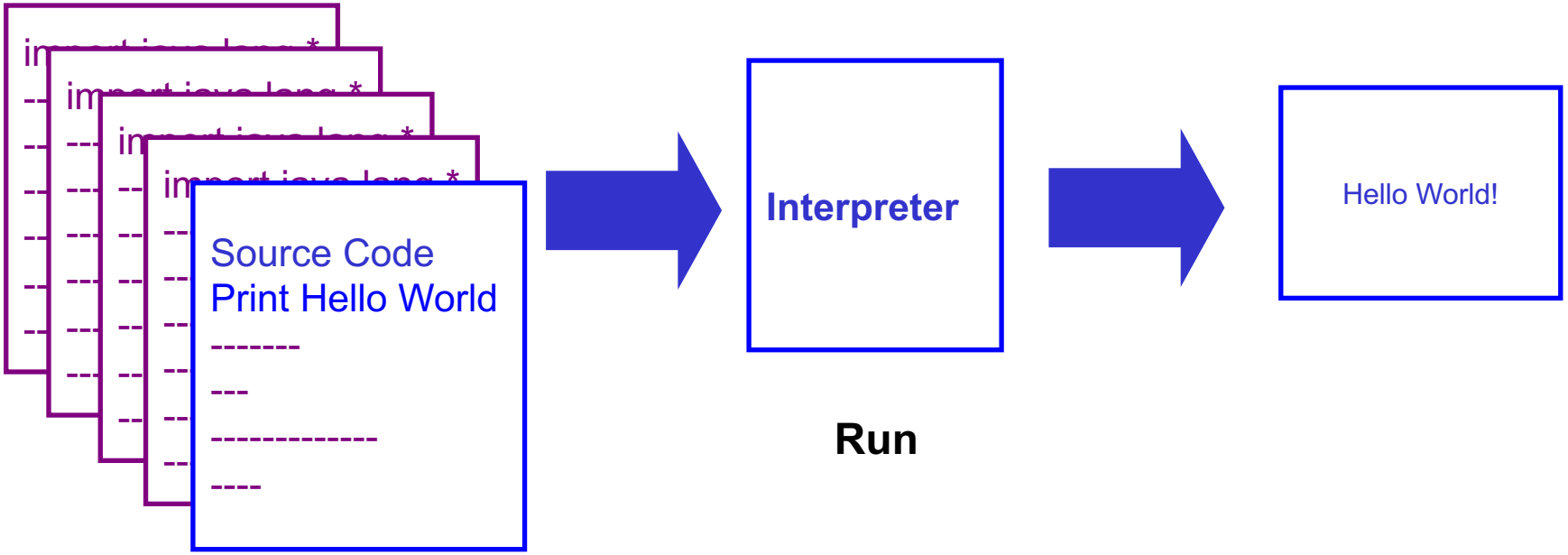
- Each of the most common languages is object oriented
- Java is particularly popular in large organizations

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

Agenda

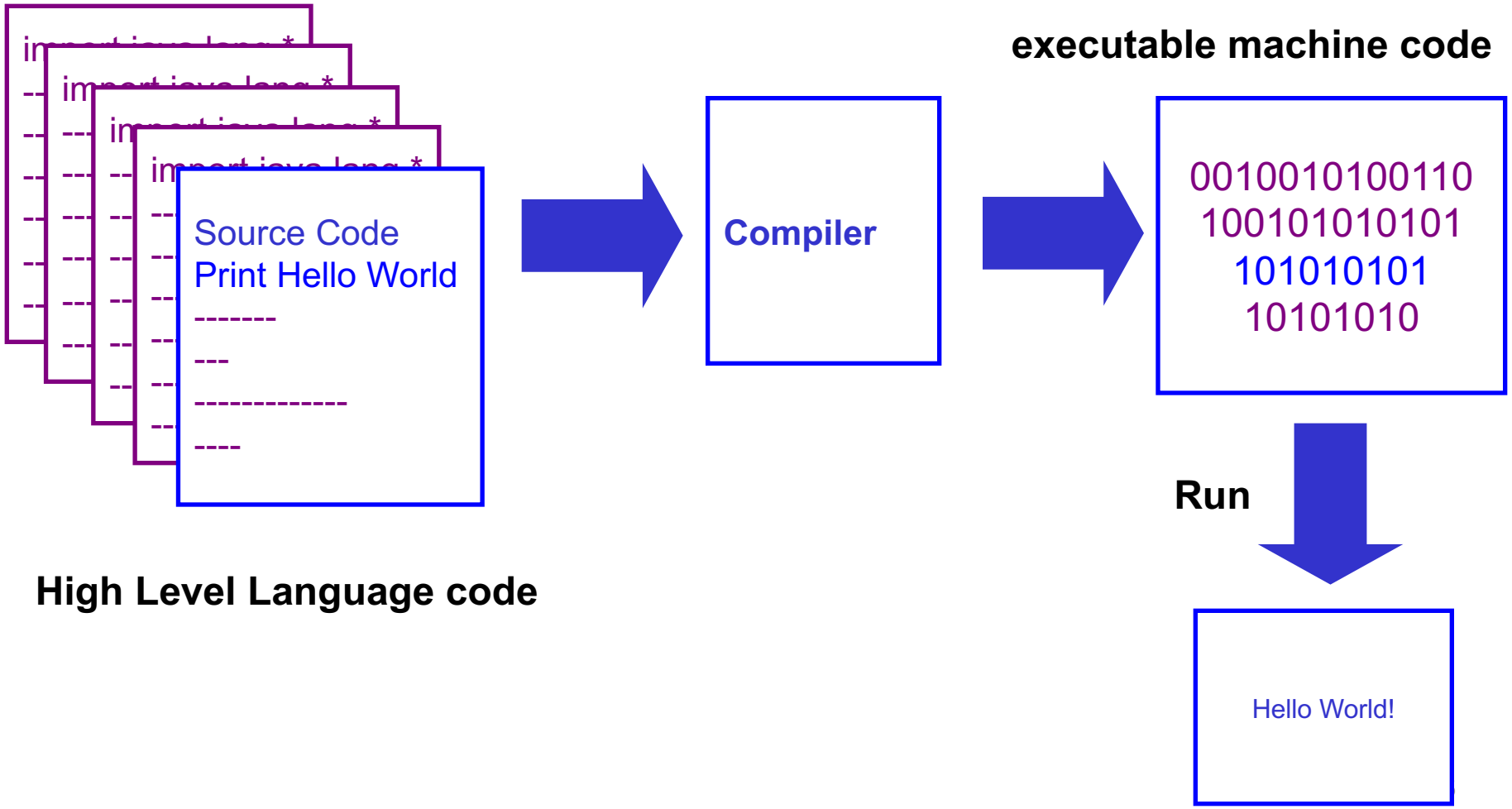
1. You, me, and this course
2. Object Oriented Programming (OOP)
3. Java intro
    1. Hello world
    2. Variables
    3. Arrays

Interpreted vs. compiled language



High Level Language code

Interpreted vs. **compiled** language



Agenda

1. You, me, and this course
2. Object Oriented Programming (OOP)
3. Java intro
   1. Hello world
   2. Variables
   3. Arrays

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

Python

```python
"""
   Standard 'Hello World' first program

   @author Everyone who has ever written about programming
"""

print("Hello World!")
```

Java

```java
/**
 * Standard 'Hello World' first program
 *
 * @author Everyone who has ever written about programming and Tim Pierson too,
 * Dartmouth CS 10, Fall 2018
 */

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

    }
}
```

Agenda

1. You, me, and this course

2. Object Oriented Programming (OOP)

3. Java intro
    1. Hello world
    2. Variables
    3. Arrays

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

Python example **python_variables0.py**

Code

```
print(x)
```

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)
```

Note: we 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))
```

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))
```

Output

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

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

<u>Common</u> 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 does not initialize local variables

JavaVariables0.java

Code

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

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);
   }
}
```

Output

```
$ javac JavaVariables0.java
JavaVariables.java0:4: 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);
   }
}
```

Output

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

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);
   }
}
```

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);
   }
}
```

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. Object Oriented Programming (OOP)
3. Java intro
    1. Hello world
    2. Variables
    3. Arrays

We can use multiple variables to store multiple values

MulitpleVariables.java

Code

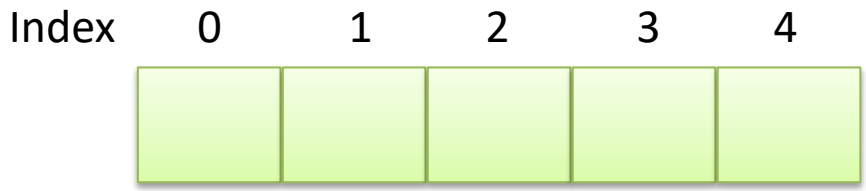
```
public class MultipleVariables {
   public static void main(String[] args) {
      int score1 = 5, score2 = 7;
      System.out.println(”score1 = "+ score1 + ", score2 = " + score2);
   }
}
```

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;
      scores[5] = 9;
      System.out.println(scores);
   }
}
```

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

Code

Index	0	1	2	3	4
	10	3.2	6.5		

```
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;
      scores[5] = 9;
      System.out.println(scores);
   }
}
```

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

Index	0	1	2	3	4
	10	3.2	6.5	7.8	8.8

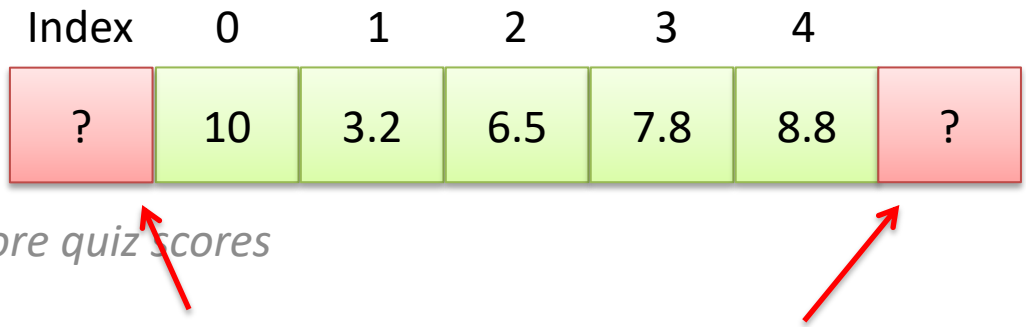
Code

```
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);
   }
}
```

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);
   }
}
```

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

Index	0	1	2	3	4
	10	3.2	6.5	7.8	8.8

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);
   }
}
```

Output

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

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

Index	0	1	2	3	4
	10	3.2	6.5	7.8	8.8

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] + "]");
   }
}
```

Output

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

Java also has multidimensional arrays

MultidimensionalArray.java

Code

```
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]);
      }
   }
}
```

Arrays holding numeric values are initialized to zero

MultidimensionalArray.java

Code

```
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]);
      }
   }
}
```

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
```

Summary

- Syllabus: active learning, how to get help, honor code
- Object-oriented programming pillars
  - Abstraction
  - Encapsulation
  - Inheritance
  - Polymorphism
- Java intro:
  - Hello world
  - Variables with types
  - Arrays
  - Errors indicated at compile or run time

Next

- Encapsulation

Additional Resources

CREATING HELLO WORLD

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"

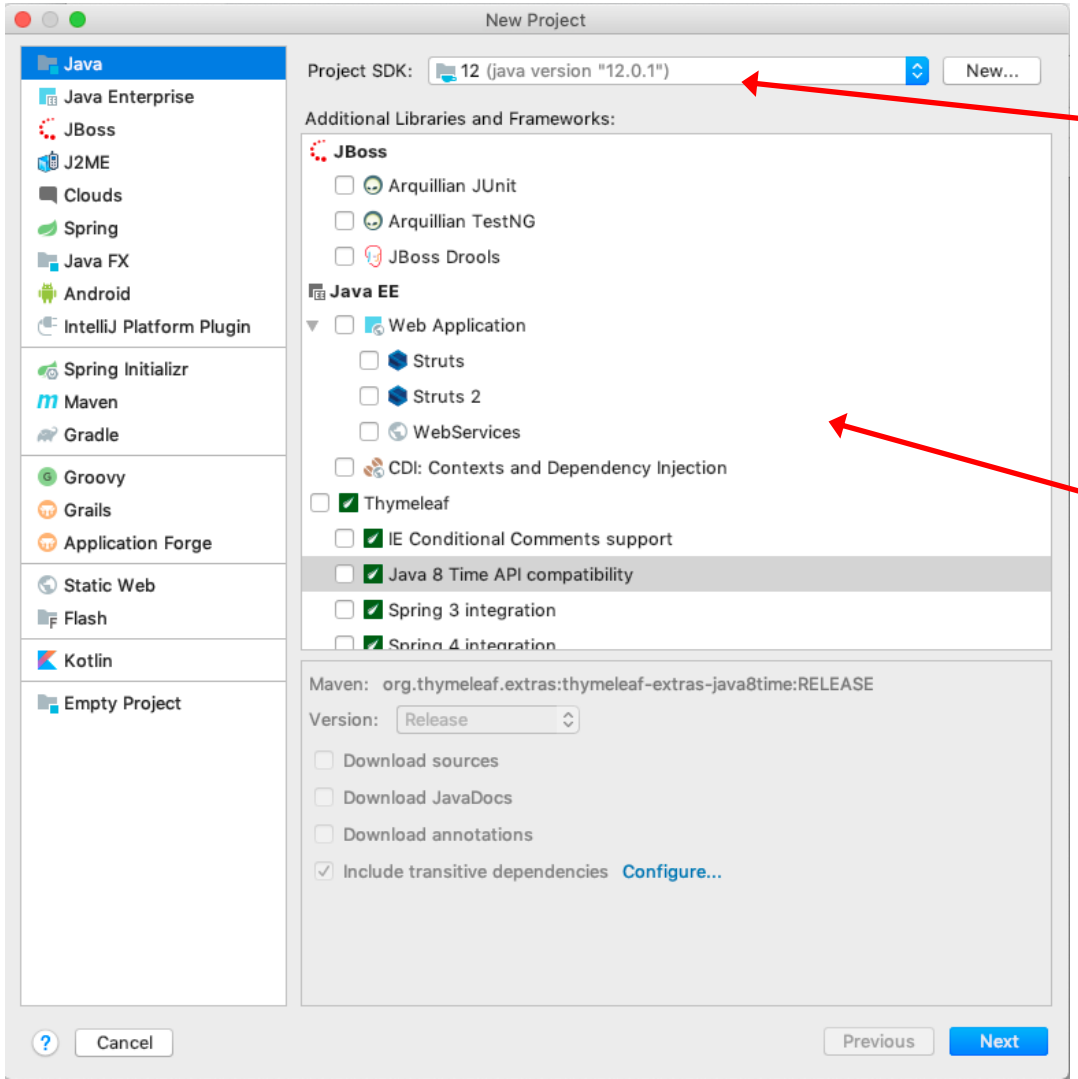
main() is where action starts

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) Choose Java

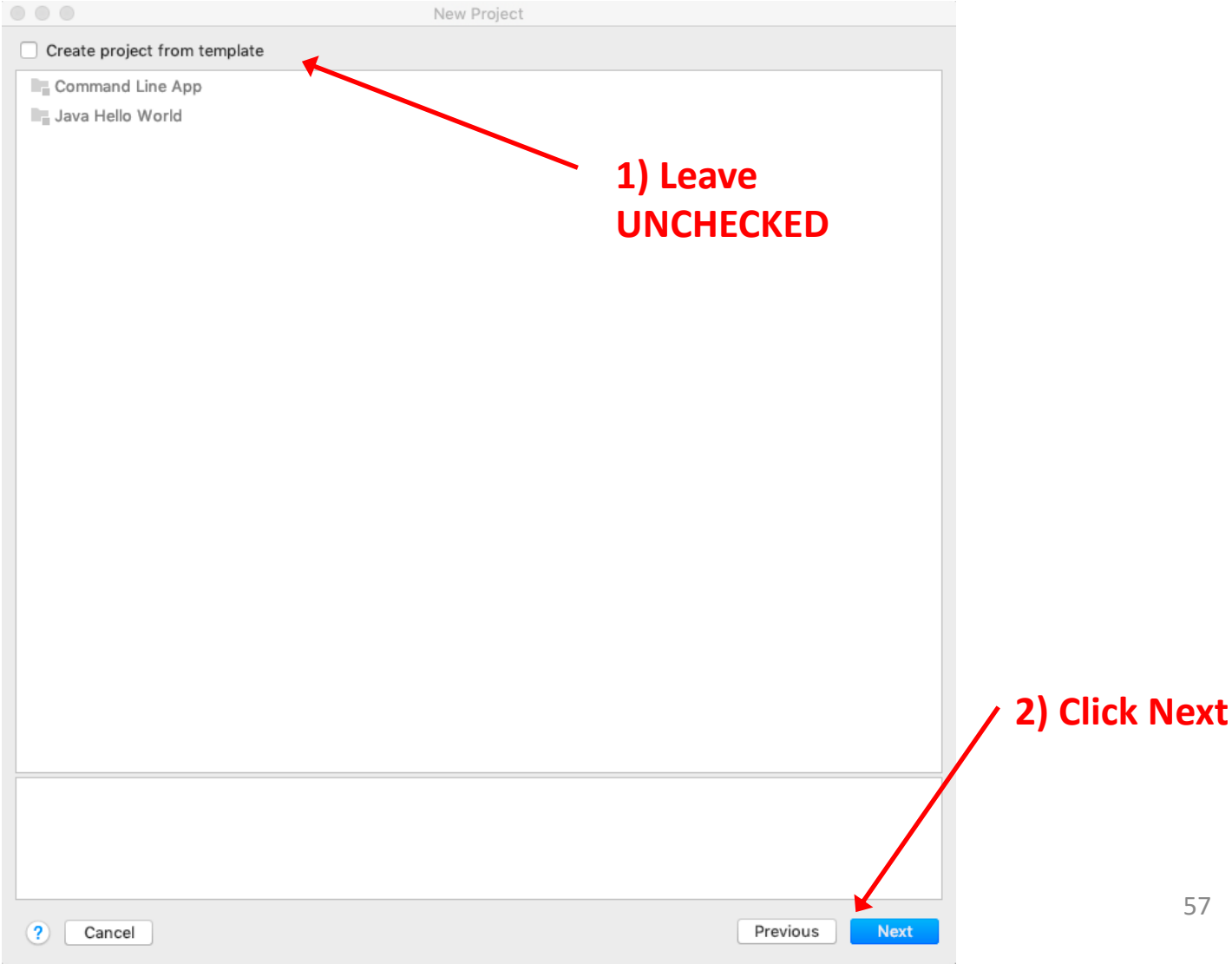
2) Choose Java version

3) Take defaults

4) Click Next

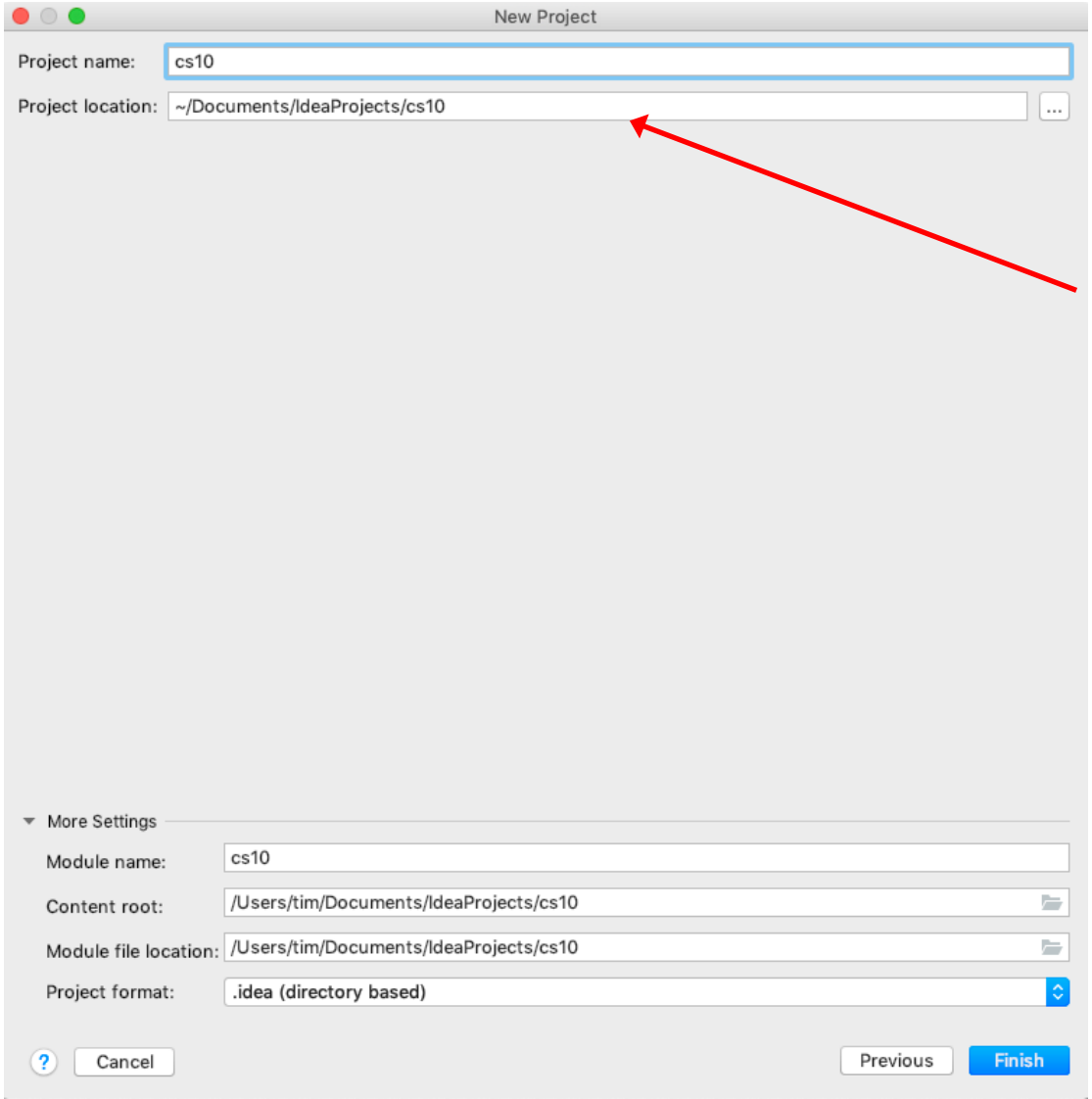
1. Create “cs10” Project to hold source code (only need to do this one time)

Do not create project from template



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

Name project “cs10” and set directory on disk where code will be stored



1) Choose Project name (“cs10”)

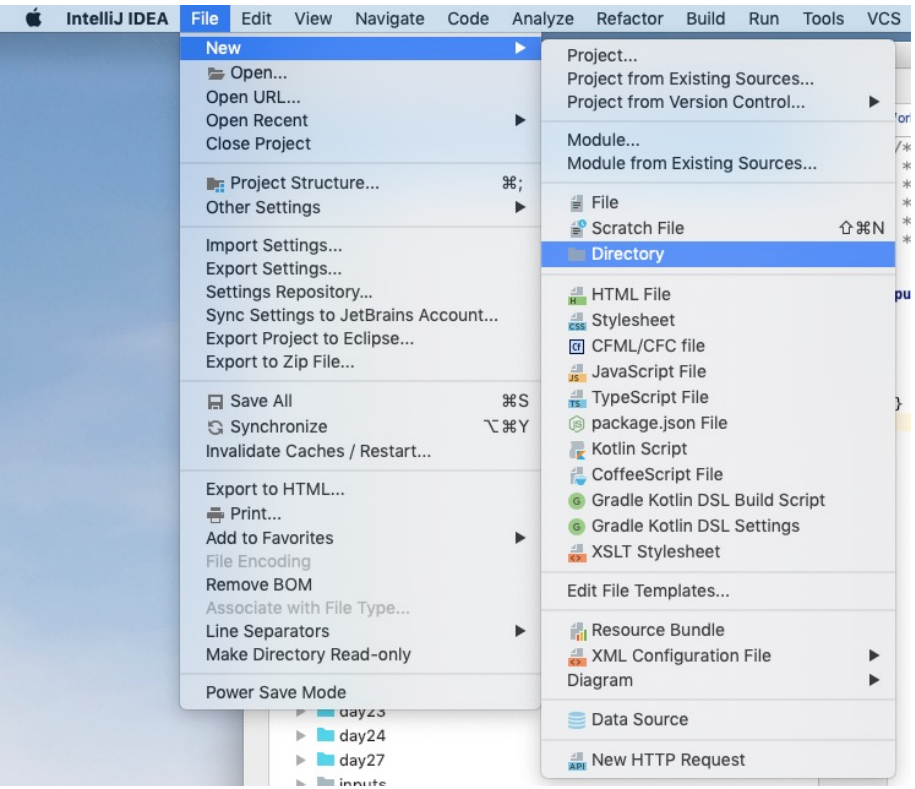
2) Choose directory where code will be stored

3) Click Finish

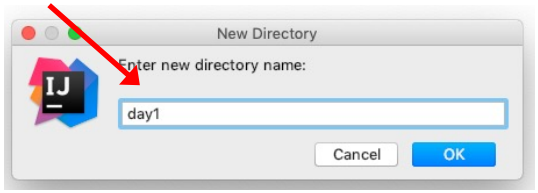
2. Create Source folder to hold your source code for day one of class

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

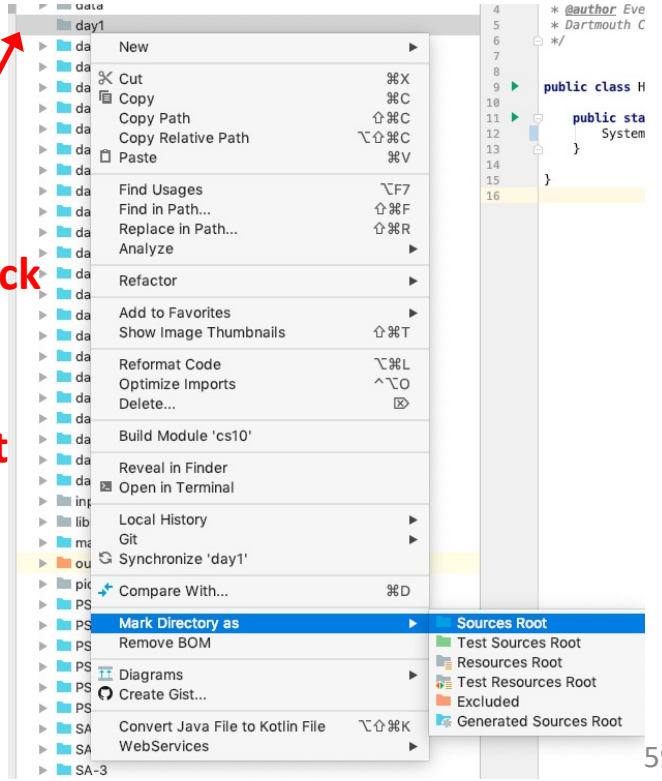
1) Click File->New->Directory



2) Give directory a name



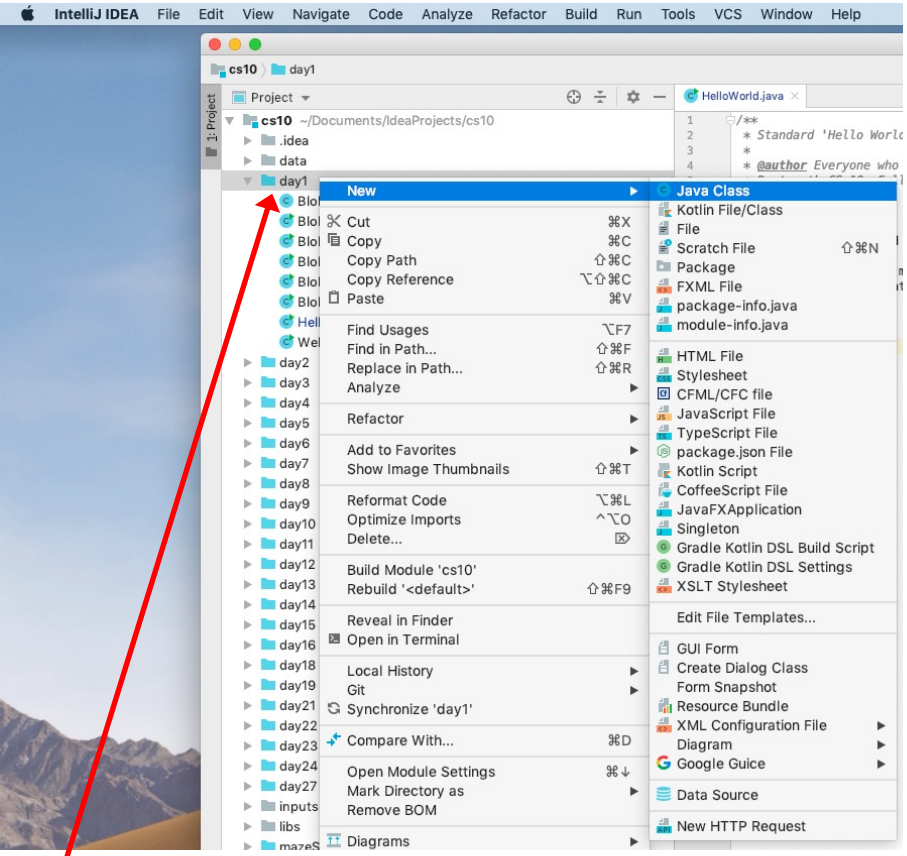
3) Right click on new directory then select “Mark Directory as” and “Sources Root”



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

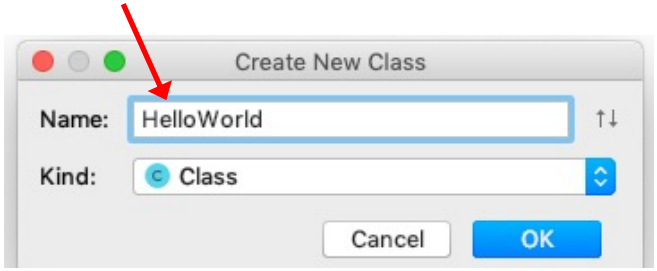
3. Create new “HelloWorld” class in “day1” source folder

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

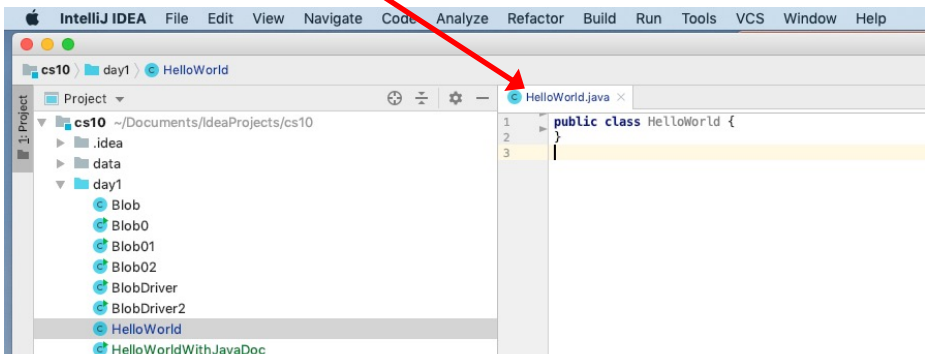


1) Right click on Source folder (e.g. “day1”), then select New->Java Class

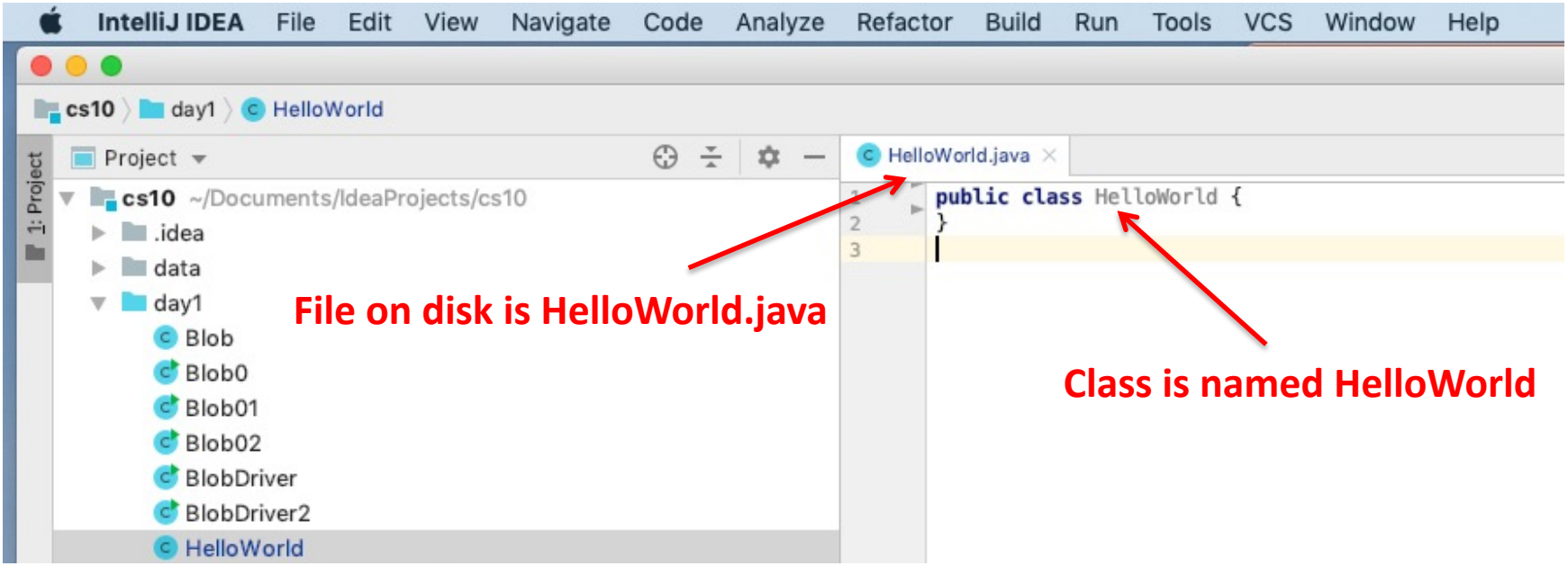
2) Give class a name (starting with capital letter)



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



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

```
public class HelloWorld {
    main
}
    main                                  main() method declaratio
Press ^. to choose the selected (or first) suggestion and insert a dot afterwards >>
```

```
public class HelloWorld {
    public static void main(String[] args) {

    }
}
```

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!)

```
public class HelloWorld {
    public static void main(String[] args) {
        sout
}
        sout                              Prints a string to System.ou
        soutm          Prints current class and method names to System.out
        soutp       Prints method parameter names and values to System.out
        soutv                              Prints a value to System.out
Press ^. to choose the selected (or first) suggestion and insert a dot afterwards >>
```

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println();
    }
}
```

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, Fall 2018
 */

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

```
/**
 * Standard 'Hello World' first program
 *
 * @author Everyone who has ever written about programming and Tim Pierson too,
 * Dartmouth CS 10, Fall 2018
 */

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

    }
}
```

Copy Reference ⌥⇧⌘C
Paste ⌘V
Paste from History... ⇧⌘V
Paste without Formatting ⌥⇧⌘V
Column Selection Mode ⇧⌘8
Refactor
Folding
Analyze
Go To
Generate... ⌘N
Recompile 'HelloWorld.java' ⇧⌘F9
Run 'HelloWorld.main()' ^⇧R
Debug 'HelloWorld.main()' ^⇧D
Run 'HelloWorld.main()' with Coverage
Save 'HelloWorld.main()'
Reveal in Finder
Open in Terminal
Local History
Git
Compare with Clipboard
File Encoding
Diagrams
Create Gist...
WebServices

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

Output appears in console below

```
/Library/Java/JavaVirtualMachines/jdk-12.0.1.jdk/Contents/Home/bin/java "-javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.ja
Hello World!

Process finished with exit code 0
```

Variables – Python example

ANNOTATED SLIDES

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 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))
```

Python allows the type of a variable to change

Still guesses variable type based on value assigned

Now thinks *x* is a String

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

Output

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

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

Variables – Java example

ANNOTATED SLIDES

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);
   }
}
```

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

When a variable is declared Java allocates memory for it

Here Java allocates memory for one integer (4 bytes)

Output

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!

Output

```
$ javac JavaVariables0.java
JavaVariables.java0:4: 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 (JavaVariables.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
```

Arrays

ANNOTATED SLIDES

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

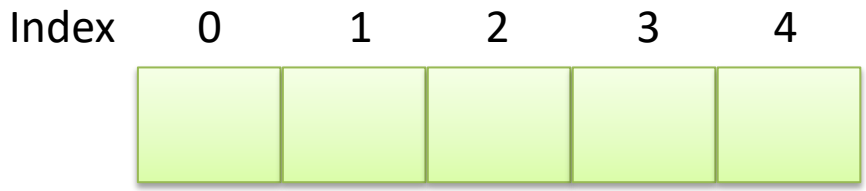
Output

```
$ 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)

***scores* holds the starting address of contiguous memory block**

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);
   }
}
```

Index	0	1	2	3	4
	10	3.2	6.5		

Access values the array holds by telling Java which index to use

Java gets address:
starting address + index * size of values

Here index 2 is at starting address of *scores* array + 2 * 8 bytes/double (16 bytes offset from start)

Can find the first element in same time it takes to find the last element

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

Index	0	1	2	3	4
	10	3.2	6.5	7.8	8.8

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);
   }
}
```

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);
   }
}
```

Index	0	1	2	3	4	
?	10	3.2	6.5	7.8	8.8	?

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

Index	0	1	2	3	4
	10	3.2	6.5	7.8	8.8

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);
   }
}
```

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

Index	0	1	2	3	4
	10	3.2	6.5	7.8	8.8

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] + "]");
   }
}
```

Normally 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

Index	0	1	2	3	4
	10	3.2	6.5	7.8	8.8

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 elements 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

MultidimensionalArray.java

Code

```
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]);
      }
   }
}
```

Value set

Java initializes numeric array values to zero

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
```