# Intro to Data Structures

CS151

Fall 2022

## Course Goals

1. Become a better computer scientist
2. Learn about common data structures
    1. Implementation
    2. How and when to use each
3. Understand Object Oriented program design and its implementation in Java
4. Become a better Java programmer
5. Develop an understanding of UNIX / working at the command line

## Things to Know

- Course website
  - cs.brynmawr.edu/cs151
    - usually updated before and after each class
      - lecture notes and code sample will be posted before class
      - updates after class with revisions, etc
- Syllabus
  - cs.brynmawr.edu/cs151/syllabus2.html
    - usually updated on weekend for next week's material

## More Things to Know

- CS account
  - You should have gotten email from ddiaz1@brynmawr.edu
- Lab:
  - Park 231
  - Th 11:25 - 12:45
  - Lab work may be done in groups!
    - I encourage you to do so.
- Software: Java, Visual Studio Code, Unix

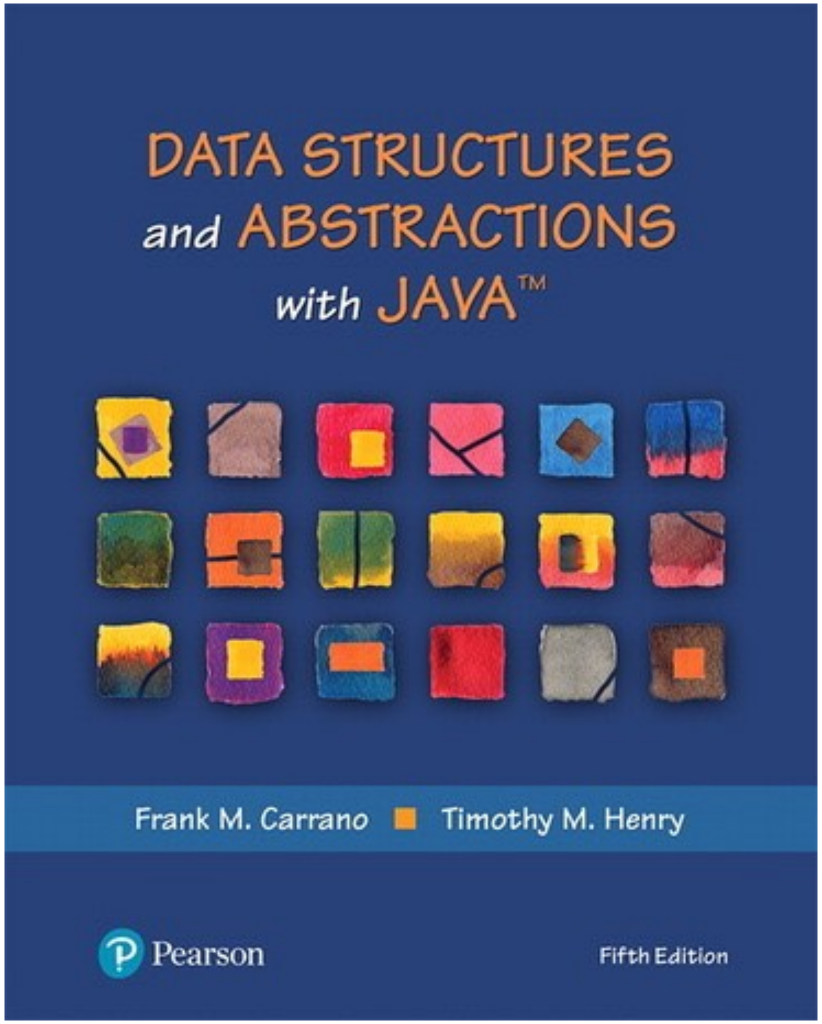
There will be a lab this week!!

# Yet More

- Homeworks
  - Approximately weekly
  - Almost always due Wednesday before midnight
  - Almost always assigned on Thursday
  - TAs in Park 231
    - Sun--Thu:  7-10
      - posted and updated on the class web site.

There will be an assignment this week

# Textbook


DATA STRUCTURES and ABSTRACTIONS with JAVA™
Frank M. Carrano ■ Timothy M. Henry
Pearson
Fifth Edition

## Data Structure?

- Wikipedia: a **data structure** is a **data** organization, management, and storage format that enables efficient access and modification
- We will talk about approximately 8 data structures
    - How to use
    - Why to choose this one
    - How to implement

## Data Structures

- Array
- ArrayList
  - it grows and shrinks
- Maps / Hashtables
  - going beyond numeric indexes
- Stacks and Queues
- Linked Lists
- Trees
- Graphs

## Programming techniques and concepts

- Object oriented programming
  - inheritance, generics, …
- Asymptotic Analysis
- Recursion
- Searching
- Sorting

## Java

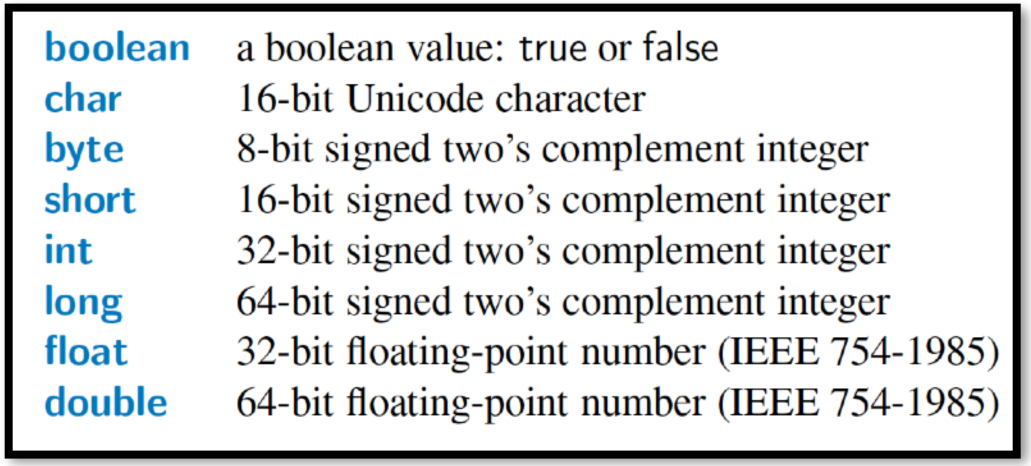
- “Object Oriented” Language
- Data Types
    - Base
        - fixed set
        - Initial lower case letter (e.g. int)
    - Objects (Classes)
        - User extensible
        - Initial capital letter (by convention)

# Base/Primitive Types

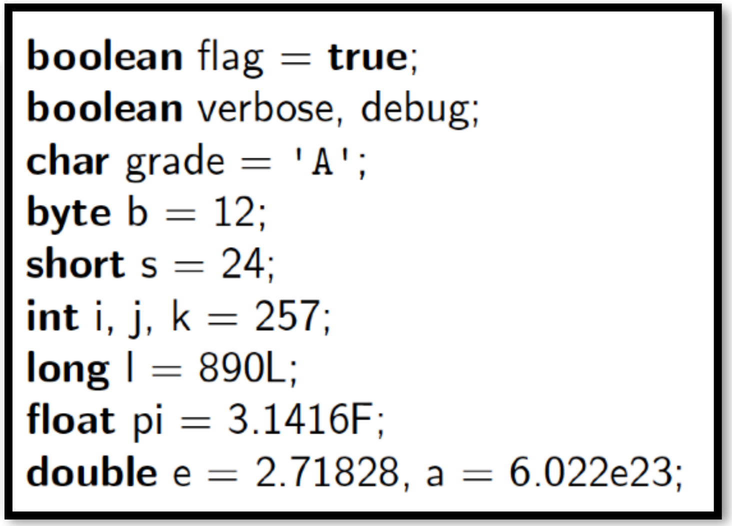
- Primitive types precisely define the way memory used to store the data

Extant definitions of  primitives

subject to change



| | |
|---|---|
| boolean | a boolean value: true or false |
| char | 16-bit Unicode character |
| byte | 8-bit signed two's complement integer |
| short | 16-bit signed two's complement integer |
| int | 32-bit signed two's complement integer |
| long | 64-bit signed two's complement integer |
| float | 32-bit floating-point number (IEEE 754-1985) |
| double | 64-bit floating-point number (IEEE 754-1985) |



```
boolean flag = true;
boolean verbose, debug;
char grade = 'A';
byte b = 12;
short s = 24;
int i, j, k = 257;
long l = 890L;
float pi = 3.1416F;
double e = 2.71828, a = 6.022e23;
```

## Testing max Integer

```
public class BoundTest {
    public static void main(String[] args) {
        System.out.println("MAX:" + Integer.MAX_VALUE + "    MIN:" +
Integer.MIN_VALUE);
        BoundTest bt = new BoundTest();
        bt.testInt(1);
    }
    public void testInt(int startingValue) {
        int intV = startingValue;
        for (int jj = 1; jj < 100 && intV > 0; jj++) {
            intV *= 2;
            System.out.println("Pow " + jj + " " + intV);
        }
        for (int jj = 0; jj < 10; jj++) {
            System.out.println("minus " + jj + " " + (intV - jj));
        }
    }
}
```

# How integers are stored

- Everything is bits
  - 0 or 1
- the int type uses 32 bits with number in base 2
- To show +/- the leftmost bit
  - “sign bit”
  - 0—positive
  - 1—negative
  - “two’s complement”

Suppose you have 4 bits for a number

| base 10 | in bits |
|---|---|
| 0 | 0000 |
| 1 | 0001 |
| 2 | 0010 |
| 3 | 0011 |
| 4 | 0100 |
| 5 | 0101 |
| 6 | 0110 |
| 7 | 0111 |
| -8 | 1000 |
| -7 | 1001 |

## Classes and Variables

- A class is a description of what an object stores (its data) and how it functions
  - ▫ instance variables
  - ▫ methods
- ▫ Every variable is either a base type or a reference to an object
- Every object is an instance of a class
    - Object names — Convention: initial capital
    - instances — Convention: initial lower case
        - camel case thereafter, camelCaseThereAfter

## Creating and Using Objects

- In Java, a new object is created by using the `new` operator followed by a call to a constructor for the desired class.
- A constructor is a special method that shares the same name of its class. The new operator returns a reference to the newly created instance.
  - every method other than a construction must give the type of information it returns
- **Almost everything in Java is a class**
  - More properly, almost all variables in Java store references to instances of a class

# Defining Objects

```
/**
 * A simple class from a simple song
 * Created: Sep 2020
 * Modified: Jan 2022
  * @author gtowell
 */
public class Inchworm
{
    //instance variable comment
    private int measurement;
    /**
     * Create an inchworm starting the given value
     * @param startingMeasurement  the starting measurement
     */
    public Inchworm(int measurement) {
        this.measurement = measurement;
    }
```

# Class Part2

```
/**
 * Create an inchworm with a default starting position (1).
 */
public Inchworm() {
    this(1);
}
/**
 * The constructor copies the state of an existing inchworm
 * @param iw the inchworm to be copied
 */
public Inchworm(Inchworm iw) {
    this.measurement = iw.getMeasurement();
}
/**
 * Get accessor for measurement.
 * Get accessors need NOT be commented
 * @return the measurement
 */
public int getMeasurement() {
    return this.measurement;
}
```

Always use accessors. No public instance variables

# Class Part3

```java
/**
 * Change the measurement by doubling.  It is all inchworms can do.
 */
public void doubleMeasure() {
    this.measurement *= 2;
}
/**
 * The toString function.  Normally this does not need a comment.
 * @Override indicates that function is defined in ancestor
 */
@Override
public String toString() {
    return "The marigold measures " + this.measurement + " inches";
}
/**
 * Put the inchworm back in its base state
 */
public void reset() {
    this.measurement=1;
}
```

## Class Part4

```
/**
     * Function to be executed at start.
     * @param args NOT used.
     */
    public static void main(String[] args) {
        Inchworm inchworm = new Inchworm();
        inchworm.doubleM();
        System.out.println(inchworm);
        Inchworm inchworm2 = new Inchworm(inchworm);
        inchworm2.doubleM();
        System.out.println(inchworm2 + " " + inchworm);
    }
```

# Access Control Modifiers

- `public` — all classes may access
- `private` — access only within that class.
- `protected` — access only from descendents
- “” (aka package) — access only by classes within the package
    - (I hate significant whitespace)
  - The package is generally the code you are working on.
  - packages are useful in large development projects (>10 people)
  - DO NOT use in this course

# Static

- When a variable or method of a class is declared as `static`, it is associated with the class as a whole, rather than with each individual instance of that class.
- **Only acceptable use** (at least for this course):
  - In methods …
    - `public static void main(String[] args)`
  - In variables .. to declare constants
    - `public static final double GOLDEN_MEAN =1.61803398875;`

## Casting (of base types)

- Assignment REQUIRES type equality
- Use casting to change type
- Must explicitly cast if there is a possible loss of precision
  - see Casting.java

```
private void trial()
    {
        int x = 5;
        double y = 1.2;
        y = x;
        ~~x = y;~~

        y = (double) x;
        x = (int) y;
    }
```

## `.equals:` Object Equality

- Do not use `==`
  - Use `==` only when comparing base types
- Review your strings and `String` class methods

```
public class StringEqual {
  public static void main(String[] args) {
    String str1 = new String("one");
    String str2 = new String("one");
    System.out.println("str1==str2: "
             + str1 == str2);
    System.out.println("str1==str2: "
             + (str1 == str2));
    System.out.println("str1.equals(str2): "
             + str1.equals(str2));
  }
}
```

## What you should know/review

- variables
- operators
- methods
  - parameters
  - return value
- conditionals
- `for`/`while` loops
- class design and object construction
  - instance variables
  - constructor
  - getters/setters
  - class methods
  - `new`
- arrays
- arrays of objects
- `String`