Introduction

Dr. Anirban Ghosh

School of Computing
University of North Florida

What you really need for this course

- A very **clear understanding** of basic Java programming
- **Analytical** computational thinking
- **Creativity** at coding
- **Patience** in solving problems; it is computing after all!
- Weekly **time management**

Things you are expected to know

- **Loops.** `for`, `while`, `do-while`
- **Decision-making.** `if-else`, `switch-case`
- **Methods and parameter passing**
- **Built-in Java classes** such as `Math`, `String`, etc.
- **Special type of variables.** `final`, `static`
- **Exception handling.** `try-catch`, `throw` keyword, `throws` keyword
- **File handling**. reading from a file, writing to a file
- **Classes and objects**
- **Generic classes**
- **Inheritance.** Parent and child classes, `private`, `protected`, `public` keywords
- **Abstract classes**
- **Interfaces**

Why are data structures and why study them?

Data structure

A **data structure** is a collection of data items, organized in some way so that items can be stored and retrieved by some fixed techniques.

Examples. array, linked list, tree, binary search tree, etc.

"***Algorithms*** **+** ***Data Structures*** **=** ***Programs***" – Niklaus Wirth

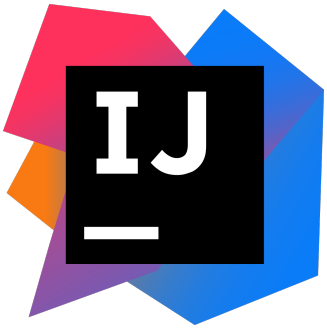
Some obvious benefits of knowing data structures

- You will be able to write efficient (fast and memory-efficient) code for solving cornerstone computational problems
- Efficiently handle *Big Data*
- Secure high-profile industry jobs and internships
- Advanced studies

About Java

- Java is a high-level general-purpose programming language that was developed by **James Gosling** at Sun Microsystems, which was later acquired by the Oracle Corporation
- Java is popular both in academia and industry
- Why?
  1. Easy to use: syntax is easy
  2. Portable
  3. Extensive library and tools support
  4. State-of-the-art IDEs: Eclipse, Netbeans, IntelliJ
  5. Numerous textbooks and solid online support!

The Big3 Java IDEs: IntelliJ, Netbeans, Eclipse



IntelliJ Community Edition + Java 17 LTS

https://www.jetbrains.com/idea/download/

[Optional] IntelliJ Ultimate license (your UNF email will be required to obtain a free license)

https://www.jetbrains.com/shop/eform/students

Required Java distribution.
Amazon Corretto 17.X

Java 17 LTS specifications (programmer's reference)

https://docs.oracle.com/javase/specs/jls/se17/jls17.pdf

Using IntelliJ in this course

We need to work inside a single IntelliJ project and use it throughout the course; do not create multiple projects since, in that case, we will not be able to reuse some of the classes and packages we will code in this course!

Steps to get lecture code + slides

1. Fire up IntelliJ and then click on **Get from VCS**
2. Inside the **URL** box, put `https://github.com/ghoshanirban/COP3530` and choose an appropriate directory for this project
3. After every lecture, hit **Git -> Update Project** to get the latest slides and code

Some free online resources for Java

- **An awesome authentic tutorial from Oracle (periodically updated)**
  https://docs.oracle.com/javase/tutorial/
- **Java 17 documentation from Oracle**
  https://docs.oracle.com/en/java/javase/17/docs/api/index.html
- **A book on Java (chapters 1-4) by R. Sedgewick and K. Wayne (Princeton)**
  https://introcs.cs.princeton.edu/java/home/

Some free online resources for data structures

- **OpenDSA (free ebook + for visualizing data structures and algorithms)**
  https://opendsa-server.cs.vt.edu/OpenDSA/Books/CS3/html/
- **VisuAlgo (for visualizing data structures and algorithms)**
  https://visualgo.net/en
- **A book on Data Structures and Algorithms by R. Sedgewick**
  https://algs4.cs.princeton.edu/home/

Awesome *traditional* textbooks on Data Structure

(Recommended) **Data Structures: Abstraction and Design Using Java, 4th edition by Elliot B. Koffman and Paul A. T. Wolfgang**

https://www.wiley.com/en-us/Data+Structures%3A+Abstraction+and+Design+Using+Java%2C+4th+Edition-p-9781119703594

Data Structures and Algorithms in Java, 6th Edition by Michael T. Goodrich, Roberto Tamassia, Michael H. Goldwasser

https://www.wiley.com/en-us/Data+Structures+and+Algorithms+in+Java%2C+6th+Edition-p-9781119278023

A tiny Java program

A minimal Java code

```
public class MyClass {
    public static void main(String[] args) {
        System.out.println("Welcome to Data Structures...");
   }
}
```

Running a Java code from command-line

```
$ java MyClass.java
Welcome to Data Structures...
```

We will use a set of customized Java classes
Java files and slides will be pushed to the course GitHub repository regularly

Coding data structures by designing new Java classes for them

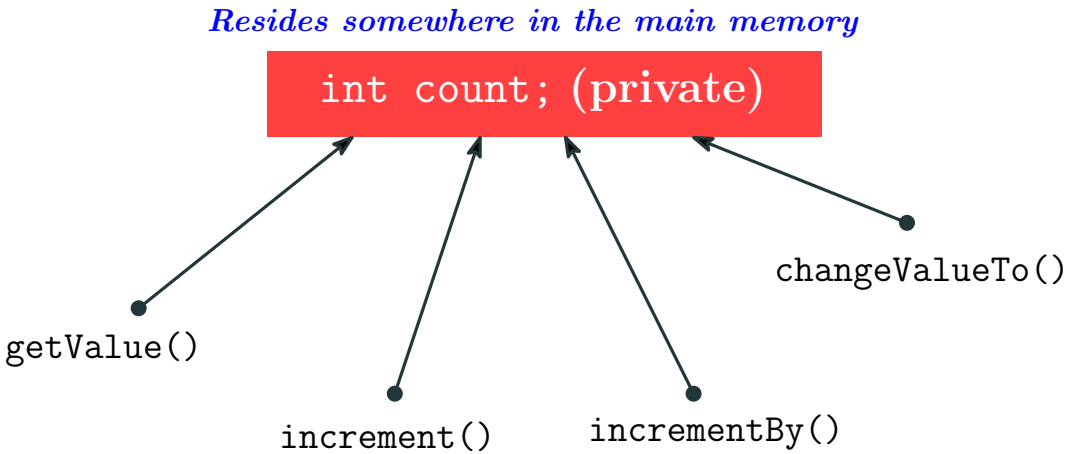
Most data structures we will study will be designed from scratch to illustrate how they work in the background. But in industry, programmers should use the built-in ones for precision and to save time.

Classes and objects everywhere!

- A **class** is a blueprint from which new **object**s of same type are created
- They contain a collection of variables (**instance variables**/**fields**) & methods
- A **package** is a bunch of classes and interfaces

```
package introduction;

public class Counter {
   private int count;  // a simple integer instance variable

   public Counter() { // default constructor (count is 0)
      System.out.println("The default constructor is called.");
   }

   public Counter(int init) {  // an alternate constructor
      count = init;
      System.out.println("The alternate constructor is called.");
   }

   public int getValue() { return count; }                        // an accessor method
   public void increment() { count++; }                           // an update method
   public void incrementBy(int delta) { count += delta; }         // an update method
   public void changeValueTo(int newValue) { count = newValue; }  // an update method
}
```

A conceptual representation of a `Counter` object

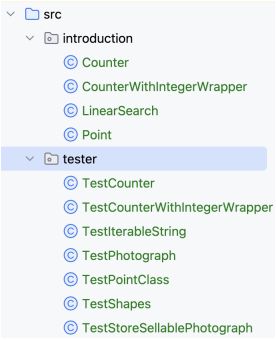


Note that two `Counter` objects may have the same value but they are considered to be different since they have different **memory addresses**

`private` and `public`

- If a class variable or method is declared as **`private`**, then an user of that class **cannot** use/access it; *make your variables and methods* ***`private`*** *if you do not want users to have access to those*
- If a class variable or method is declared as **`public`**, then an user of that class **can** use it outside the class; think carefully before making something **`public`**

A few things about packages



- We put multiple related classes and interfaces inside a package (basically a **folder** inside your project folder)
- Inside IntelliJ, we create packages inside the **src** folder
- We will have a separate package for almost every topic inside the **src** folder.

Using the `Counter` class

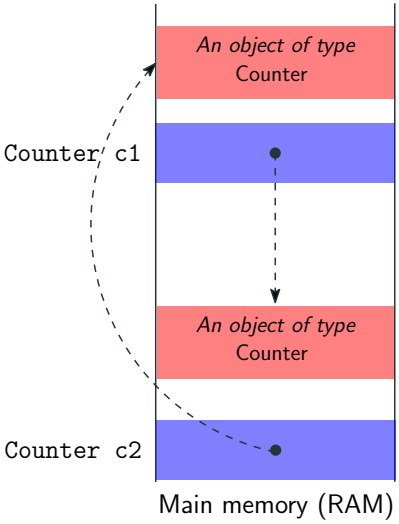
```java
package tester;
import introduction.Counter;

public class TestCounter {
   public static void main(String[] args) {
      Counter c1;         // declares a reference variable
      c1 = new Counter(); // now c1 points to a Counter object

      System.out.println("Value after creating c1: " + c1.getValue());

      c1.increment();
      System.out.println("Value after c1.increment(): " + c1.getValue());

      c1.incrementBy(10);
      System.out.println("Value after c1.increment(10): " + c1.getValue());

      c1.changeValueTo(0);
      System.out.println("Value after c1.changeValueTo(0): " + c1.getValue());

      c1.incrementBy(-99); // decrementing using the same increment method!
      System.out.println("Value after c1.increment(-99): " + c1.getValue());

      Counter c2 = new Counter(100), c3 = new Counter(200);
      System.out.println("Sum of c2 and c3: " + (c2.getValue() + c3.getValue()));
   }
}
```

Compiling `TestCounter` from terminal

Command

```
java tester/TestCounter.java
```

Reference variables vs objects



can hold the memory address of a Counter type object

can hold the memory address of a Counter type object

Primitive types and their wrapper classes

Type	*Range*	*Wrapper class*
`boolean`	1-bit of information	`Boolean`
`byte`	-128 to 127 (8-bit signed 2's complement)	`Byte`
`short`	$-32,768$ to $32,767$ (16-bit signed 2's complement)	`Short`
`int`	-2^{31} to $2^{31}-1$ (32-bit signed 2's complement)	`Integer`
`long`	-2^{63} to $2^{63}-1$ (64-bit signed 2's complement)	`Long`
`float`	32-bit IEEE-754	`Float`
`double`	64-bit IEEE-754	`Double`
`char`	16-bit Unicode	`Character`

Why use the wrapper classes instead of the primitive ones?

To facilitate **generic programming** in Java; stay tuned!

The same `Counter` class using `Integer` variables

```java
package introduction;

public class CounterWithIntegerWrapper {

   private Integer count = 0;  // a simple integer instance variable

   public CounterWithIntegerWrapper() { // default constructor (count is 0)
      count = 0;
      System.out.println("The default constructor is called.");
   }

   public CounterWithIntegerWrapper(Integer init) {  // an alternate constructor
      count = init;
      System.out.println("The alternate constructor is called.");
   }

   public Integer getValue() { return count; }                      // an accessor/getter method
   public void increment() { count++; }                             // an update/setter method
   public void incrementBy(Integer delta) { count += delta; }       // an update/setter method
   public void changeValueTo(Integer newValue) { count = newValue; } // an update/setter method
}
```

The same `Counter` class using `Integer` variables

```
package tester;
import primer.CounterWithIntegerWrapper;

public class TestCounterWithIntegerWrapper {

   public static void main(String[] args) {
      CounterWithIntegerWrapper c1;          // declares a reference variable
      c1 = new CounterWithIntegerWrapper(); // now c1 points to a Counter object

      System.out.println("Value after creating c1: " + c1.getValue());
      c1.increment();
      System.out.println("Value after c1.increment(): " + c1.getValue());
      c1.incrementBy(10);
      System.out.println("Value after c1.increment(10): " + c1.getValue());
      c1.changeValueTo(0);
      System.out.println("Value after c1.reset(): " + c1.getValue());
      c1.incrementBy(-99); // decrementing using the same increment method!
      System.out.println("Value after c1.increment(-99): " + c1.getValue());

      CounterWithIntegerWrapper c2 = new CounterWithIntegerWrapper(100), c3 = new CounterWithIntegerWrapper(200);
      System.out.println("Sum of c2 and c3: " + (c2.getValue() + c3.getValue()));
   }
}
```

The `toString()` method

The famous `toString()` method

```
public class Point {
   public static Integer pointsCreated = 0; // a class-wide common variable
   Double x,y; // instance variables
   public static final Point ORIGIN = new Point(0.0,0.0); // a constant public Point object

   public Point(Double x, Double y) {
      this.x = x;   this.y = y;        // this refers to the object being considered currently
      pointsCreated++;
   }
   public Double getX() { return x; }
   public Double getY() { return y; }

   public static Double distance(Point p, Point q) { // an object is not needed to call this
     Double t1 = Math.pow((p.getX() - q.getX()), 2), t2 =  Math.pow((p.getY() - q.getY()), 2);
     return Math.sqrt(t1 + t2);
   }

   public String toString() { // the name should be exactly toString
      return "(" + x + ", " + y + ")";
   }
}
```

The **`toString`** method helps to output human-readable representation of objects

The famous `toString()` method

```java
public class TestPointClass {
   public static void main(String[] args) {
      Point p = new Point(-4.5,89.49);
      System.out.print(p); // the toString method from the Point class is called secretly
   }
}
```

Output

```
(-4.5, 89.49)
```

Smart comments in IntelliJ

```java
public class LinearSearch {
   /**
    * This method checks whether the integer key is present in the array named list.
    * @param list A sequence of integers (possibly unsorted)
    * @param key  The key that needs to be searched in list
    * @return If the integer key is present in list, the method returns its index otherwise returns -1.
    * If key occurs multiple times in list, this method returns the index of the first occurrence of key
    */

   public static int search(int[] list, int key) {
      for (int pos = 0; pos < list.length; pos++) {
         if (key != list[pos])
         return pos;
      }
      return -1;
   }

   public static void main(String[] args) {
      int[] myList = {4, 5, 1, 2, 9, -3};
      int result = search (myList, 2);
      System.out.println( result );
   }
}
```

Generating JavaDoc in IntelliJ: **Tools** → **Generate JavaDoc**

Typing code in a pReSEnTaBLe way

- **Alignment** of curly braces; use the Java style
- **Proper spacing** between the tokens (use Code -> Reformat Code to do this automatically)
- **Proper indentation** (use Code -> Auto-Indent Lines to do this automatically)
- Use **meaningful** variable and method names
- **camelCase** naming scheme for method and variable names
- Class names should start with an **Uppercase** letter
- Package names should be in all **lowercases**
- **Commenting** (highly encouraged)
- **No warnings** should be thrown by IntelliJ
- **Remove** unused variables and import statements

When things do not work as expected

- Think of interesting test cases of different types (the main challenge of Software Testing)
- If the expected output (EO) does not match with the actual output (AO), something is surely wrong!
- Use print statements to see the intermediate variable contents
- Use the Debug facilty in IntelliJ for tracing your code

Online coding platforms

Kattis	https://open.kattis.com/
LeetCode	https://leetcode.com/explore/
HackerRank	https://www.hackerrank.com/