Computer Science II
Data Structures

Instructor
Sukumar Ghosh
201P Maclean Hall
Office hours: 10:30 AM – 12:00 PM
Mondays and Fridays

Course Webpage
homepage.cs.uiowa.edu/~ghosh/2116.html
Course Syllabus

- Constructs in Java, the language we will use
- Algorithm complexity and Big-O notation
- Arrays, Linked lists
- Solving problems using recursion
- Stacks, queues, lists and trees
- Searching and sorting
- Priority queues, hash tables, binary search trees
- Graphs and basic algorithms on graphs

Teaching Assistants

- Kyle Diederich
- Adrian Pereira
- Thamer Alsulaiman
- Dhuv Vyas
About this course

Main class (AAA) and six sections A01-A06
Discussion sections meet on Thursdays only.
You must go to your own section.

Textbook


Prerequisites

Computer Science I (CS: 1210 / 22C:016/ ENGR 2730)
(Note: CS 2210: Discrete Structures is a corequisite, if not taken as a prerequisite earlier)
Grading

Eight Home assignments (30%)
Two quizzes (2x5% = 10%)
Two in-class midterms (2x20% = 40%), and
(Monday, Sep 26 and on Monday, Oct 31)
One Final exam (20 %)

Limited collaboration is OK, assuming you have first spent some time (about 60 minutes) working on the problem yourself. However, your solution should not be a copy (whole or in part) of a fellow student.

Late Homework Policy

Quota of two days for the entire semester
How Java works

1. Java program
2. Compiler
3. Bytecode
4. Java Virtual Machine
5. Result
Which IDE will we use?

We will use **NetBeans**.

You can download it on your machines.
They are installed in all lab machines. WE will demonstrate it today in the class.
Object-oriented programming

**An Object** is a repository of data

**Typed data.** Always declare before you use.

**Primitive types.** int, char, boolean, float etc

**Class.** A template for creating objects

Think of *shoppingList* as a class. We can define two objects for this class.

Example:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
myList
```

```
yourList
```

Object *myList* belongs to the **Class** *ShoppingList*.

A **Class** denotes the type of object
y = new Counter( )

public class Counter
private int count
public Counter( ) { }

y
reference
Class Human{
Public int age;
Public String name;
Public void introduce() {
System.out.println("I’m" + name + “and I’m” + age + " years old")
}
}

Now, continue as

Human Alice = new Human();// Create Alice
Alice.age = 21;       // Set Alice’s fields
Alice.name = “Alice”;
Alice.introduce( );
Structure of a Java program

```java
public class MyFirstJavaProgram {
    public static void main(String []args) {
        System.out.println("Hello World");
    }
}
```

**Class** = blueprint of an object  
Class name starts with a capital letter  

**Object** = instance of a class, created using a **constructor**  

**Instance variables** = Unique set of variables for an object  

**Methods** = Actions to manipulate data  
Method name starts with lower case letters  

**Program file name** = Must exactly match the class name. Saved as `filename.java`  

**Package** = a group of related class definitions
What are public, private, protected?

These are Access Control Modifiers.

- **Private**: Visible to the class only
- **Public**: Visible to the world
- **No modifier**: Visible to the package, the default
- **Protected**: Visible to the package and all subclasses

### Access Control

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Same class</th>
<th>Same package</th>
<th>Subclass</th>
<th>Universe</th>
</tr>
</thead>
<tbody>
<tr>
<td>private</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>default</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>protected</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>public</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>
1. Constants
2. Instance variables
3. Constructors
4. Methods

```java
public class Counter {
    private int count; // a simple integer instance variable
    public Counter() { } // default constructor (count is 0)
    public Counter(int initial) { count = initial; } // an alternate constructor
    public int getCount() { return count; } // an accessor method
    public void increment() { count++; } // an update method
    public void increment(int delta) { count += delta; } // an update method
    public void reset() { count = 0; } // an update method
}
```
More on Types

**Primitive types**. Integer, Boolean, character etc

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>“true” or “false”</td>
<td></td>
</tr>
<tr>
<td>Char</td>
<td>A character (like ‘s’)</td>
<td></td>
</tr>
<tr>
<td>Byte</td>
<td>8-bit integer</td>
<td>-128 to +127</td>
</tr>
<tr>
<td>Short</td>
<td>16-bit integer</td>
<td>-32768 to +32767</td>
</tr>
<tr>
<td>Int</td>
<td>32-bit integer</td>
<td></td>
</tr>
<tr>
<td>Long</td>
<td>64-bit integer</td>
<td></td>
</tr>
<tr>
<td>Float</td>
<td>32-bit floating-point number</td>
<td></td>
</tr>
<tr>
<td>Double</td>
<td>32-bit floating-point number</td>
<td></td>
</tr>
</tbody>
</table>

**Reference variables**. A variable whose type is a class is called a reference variable.

```java
int x = 23;

Car c = new Car();
ShoppingList myList = new ShoppingList();
```
Method

A Method is a procedure or a function that operates on an object to produce a result.

An accessor method returns a value. An update method only makes changes in the fields, but no value (void) is returned.

Objects are manipulated by methods.

Consider a Rectangle class, and an object R belonging to this class

Rectangle R;
R = new Rectangle(5,-10,100,200);
int w = R.getWidth(); // This is OK
In the above example, **getWidth** is a method that returns the width of the rectangle.

**Note.** If you write

```java
Rectangle R;
int w = R.getWidth(); // This is wrong!
```

It will not work since R is just the name of a handle. The object has not been created yet.

---

**Static modifier (of a method or a variable)**
The value is associated with the entire class, and not to a particular instance of it)

**Abstract method**
Contains only the signature but no body

**Final method**
Attributed to a variable that does not change.
A final method cannot be overridden by a subclass.
**Objects and Methods**

Let us start with strings (sequence of characters). *Constructors* are used to initialize a new object or a variable, and always use the name of the class.

```java
String s1; // Declare a string variable
s1 = new String(); // Assign a value, an empty string
String s2 = new String(); // Short cut
s1 = “Hello”; // See below
```
s2 = s1; // See below, both point to the same string

So, how to make a separate string s2 with “Hello”?

s2 = \textbf{new} \text{String} (s1); or
s2 = “Hello”;

Java strings are \textit{immutable objects}, so instead of modifying a string create a new one. Let us look at a method:

\begin{verbatim}
    s2 = s1.toUpperCase(); // toUpperCase is a method
    // Here, s2 will be a \textbf{new} string “HELLO”
    String s3 = s2.concat(“!”); // s3 = s2 +”!”
\end{verbatim}
That is, s3 becomes a new string “HELLO!”

Java will also allow

\[
s2 = s2 + '!'
\]

// Now s2 = “HELLO!”

But it involves copying the old string and adding ‘!’ to it to generate a new string, and then switch the reference from s the handle s2. It is inefficient for long strings.

The **StringBuilder** class allows efficient editing of strings, and in essence creates a mutable string. We will discuss it later.
Arrays

Consists of a **numbered list of variables**. An array variable is a **reference variable**. So

```
int[ ] X;
```

Is only a declaration, and no allocation is made. To construct an array by allocation space in the memory, use the `new` operator

```
int[ ] X = new int[8]
```

and then initialize it.

```
char[ ] c; // Creates an array of char
c = new char[4] // Creates array of size 4
c[0] = ‘b’;
c[1] = ‘l’;
c[2] = ‘u’
c[3] = ‘e’
```
An array is an **immutable object**. The first index is always zero (not 1).

```java
for (int i = 0 ; i < A.length ; i++) A[i] = 10 ;
for (int i = 0 ; i < B.length ; i++) B[i] = 20 ;
B = A; // what happens here? See below.
```

{Note: `A.length` gives the length of the array A}
An array can also contain reference to objects.

```
string[ ] X = new string[6]
string[0] = Rene
string[1] = joseph
```
Multidimensional Arrays

```java
int[][] myMatrix = new int[8][10];

Java will treat this as an array of arrays. So, myMatrix[7] will denote the seventh row of the (8x10) matrix.

class MultiDimArrayDemo {
    public static void main(String[] args) {
        String[][] names = {
            {"Mr. ", "Mrs. ", "Ms. "},
            {"Smith", "Jones", "Ford"}
        };
        // Mr. Smith
        System.out.println(names[0][0] + names[1][0]);
        // Ms. Jones
        System.out.println(names[0][2] + names[1][1]);
        // Mrs. Ford
        System.out.println(names[0][1] + names[1][2]);
    }
}
```
**Example: Game entries**

| Alice  
250 | Lu  
220 | Jill  
200 | Mary  
180 | Natalia  
140 | Lucia  
120 |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

Scoreboard is an array of type GameEntry

In an event in the Rio Olympics, we want to record the **best six scores**, sorted in the non-decreasing, and save them in an array.

Each array element is a **GameEntry** that consists of a pair: **name** and **score**.
Here is the GameEntry class.

```java
public class GameEntry {
  private String name; // name of the person earning the score
  private int score; // the score value

  /** Constructs a game entry with given parameters. */
  public GameEntry(String n, int s) {
    name = n;
    score = s;
  }

  /** Returns the name field. */
  public String getName() { return name; }

  /** Returns the score field. */
  public int getScore() { return score; }

  /** Returns a string representation of this entry. */
  public String toString() {
    return "(" + name + ", " + score + ")";
  }
}
```
Here is the Scoreboard class.

```java
/** Class for storing high scores in an array in nondecreasing order. */
public class Scoreboard {
  private int numEntries = 0; // number of actual entries
  private GameEntry[ ] board; // array of game entries (names & sc
  /** Constructs an empty scoreboard with the given capacity for storing entries.
   * 
   */ public Scoreboard(int capacity) {
    board = new GameEntry[capacity];
  }
  ...
  // more methods will go here
}
```

Think of how to update the scoreboard when a new score (Tammy 170) or (Hillary, 100) is recorded.