CS230: Data Structures

Spring 2021

Course Website

- http://cs.wellesley.edu/~cs230

2/13/21
Why take CS230?

- You will learn the “big picture” of programming
  - Data abstraction
  - Modularity
  - Performance Analysis
  - Basic abstract data types (ADTs)

- You will become a more competent programmer
- You will also become a designer, tester, analyzer, debugger, team member
- You will have fun in the process!

Why use ADTs?

- Allows you to write complex programs more easily
  - To keep mental track of complex data interaction
  - To reuse code
  - To improve code performance

- Allows modularity of large projects
  - Easier to understand large chunks of code
  - Easier to collaborate with large teams

- Basic ADTs
  - Collections
  - Linked List
  - Stack
  - Queue
  - Hash Table
  - Priority Queue

- Less basic:
  - Tree
  - Set
  - Graph
You can study **data structures** using **any language**, but in this course we will use **Java**

You can **write and execute Java programs** in many ways, on the command line or with an IDE, but in this course we will use a simple IDE called **BlueJ**

---

**Instructor and support**

- **Who:**
  - Christine Bassem
  - Takis Metaxas
  - Stella Kakavouli

- **When:**
  - Lectures: Mondays and Thursdays from 10am to 12pm EST
  - Labs: Tuesdays and Fridays (times depend on the lab section)
  - Office hours: Check the course website for details

- **Where:**
  - *In the matrix 😊*

- **No really! We’ll be using:**
  - Zoom (Lectures, labs, and office hours)
  - Piazza (All communication)
  - GradeScope (All submissions and grading)
Textbook

- **Java Foundations**
  - by Lewis, DePasquale and Chase
  - 2nd edition or higher

Course structure

- As you know, we will be learning about Data Structures in 6 condensed weeks.
  - So, this will be a very fast paced course!
  - Please feel free to talk to me if you are not comfortable with the pace of the course.

- The course has the following components:
  - Lab preparation and participation: 10%
  - Lecture preparation and participation: 10%
  - Post-class exercises: 20%
  - Assignments: 20%
  - Mid-term Exam: 20%
  - Final Exam: 20%
Course components (1)

- There are 13 lectures as shown on the course schedule. Each lecture will have:
  - Reading from the textbook (Already posted on the course webpage)
  - Slides (Posted weekly on the course webpage)
  - Videos (On YouTube and shared weekly on the course webpage)
  - Pre-lecture reflection (Posted on the course webpage and submitted via Google Forms)
  - Post-lecture exercise (Posted on the course webpage and submitted via Gradescope)

All of these components complement each other and are necessary for your learning experience.

Course components (2)

- There are 2 virtual exams:
  - Dates are indicated in the course schedule
  - They will be open-notes
  - Might have a bit of programming

- Class participation is very critical in this remote environment.
  - If you don’t attend a lecture, without giving me a heads up, I will get worried and will have to nag-mail you until you re-assure me that you are okay!

- Does participation mean only attendance?
  - Nope 😊
  - ...

2/13/21
Class participation

- Participation has a lot of components.
- Its main goal is for me to know that you are okay and always part of the discussion.

- Here are some examples of good participation:
  - Attending class (I think you might have guessed that already 😊)
  - Keeping your video on during our meetings
  - Reacting to my questions and comments with a thumbs up or thumbs down at least (and laughing at my jokes, JK!)
  - Posting questions on Piazza
  - Answering questions on Piazza
  - Participating in the activities on Slack

Assignments

- There are three types of assignments in this course:
  - Pre-lecture reading and reflection
  - Post-lecture quizzes
  - Weekly programming assignment

- We always offer late passes, because "life happens".
- This term, we will be doing the following:
  - No late submissions allowed for the pre-lecture reading and post-lecture quizzes
  - One day (24 hours) late submissions allowed for post-lecture exercises
    - You don’t need to ask for permission
    - We highly recommend that you finish things on time though
Course readings

- You will notice that the course readings are a lot!
  - You will typically be reading at least 30 pages per lecture

- Here is my advice when you are studying:
  - Do a first pass on the required reading by scanning the content and understanding figures
  - Watch the lecture videos and take notes
  - Go back to the textbook, do a second pass, and mark those notes on the text

- You might prefer to take notes on the slides.
  - They contain a good amount of descriptive text
  - But, the textbook should be your source if you want to know more

Java syntax

Your first program
First Java Program

```
/**
 * First program for CS230
 * @author CS230 Instructor
 */
public class Welcome {

    // Program execution begins with the "main" method
    public static void main (String[] args) {
        System.out.println("Hello and Welcome to CS230!");
    }
}
```

- Multi line JavaDoc comment
- Code goes in a public class, which must be in a .java file of the same name
- Curly braces, not whitespace, denote structure
- Statements end with semicolons

Data Types in Java

- Java is a **statically typed** language
  - You must explicitly define the type of each variable when it is declared
- Unlike Python, not all variables in Java are objects
  - Some are **primitive data types**

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>Integer</td>
</tr>
<tr>
<td>float</td>
<td>Float</td>
</tr>
<tr>
<td>double</td>
<td>Double</td>
</tr>
<tr>
<td>char</td>
<td>Char</td>
</tr>
<tr>
<td>boolean</td>
<td>Boolean</td>
</tr>
</tbody>
</table>
Variable Declaration in Java

- A variable must be declared once before it can be used
- The type of a variable cannot be changed after declaration
- The value of a variable can be changed and read any number of times

```java
int x;
int y;
int z;
x = 7;
y = 5;
z = x + y;
System.out.println(z);
```

Variables declared and initialized in separate statements

```java
int x = 7;
int y = 5;
int z = x + y;
System.out.println(z);
```

Variables declared and initialized in single statement

Java under the hood

- What happens when you execute code in Java?
- Understanding how the memory is manipulated is key to becoming a successful programmer

- Memory ...
  - Do we mean RAM and stuff?

- Well, yes, in a way ...
  - No need to understand exactly what goes on in the computer memory
  - Need to at least understand how the programs we write manipulate data within them

- We will be using memory models to understand how our code works
Memory models

- There is no one standard for writing memory models in CS, but in this course, we will follow a consistent diagram-based model.

- For example
  ```java
  int x;
  int y;
  int z;
  x = 7;
  y = 5;
  z = x + y;
  System.out.println(z);
  ```

- Will result in this final model
  ```
  x (int)
  7
  y (int)
  5
  z (int)
  12
  ```

- And will result in “12” being printed out on the screen

Control flow

- Or in other words,
  - In what order does the code run to produce such an outcome?

- Java is a sequential language
  - Each line of code is executed in the order that it is written
    - For example, lines 1-7 below are executed one after the other
  - The compiler can jump from one part of the code to another if it encounters
    - Method calls
    - Conditionals
    - Loops
  ```java
  1. int x;
  2. int y;
  3. int z;
  4. x = 7;
  5. y = 5;
  6. z = x + y;
  7. System.out.println(z);
  ```
Control flow + memory models

- Again, for the same example,

<table>
<thead>
<tr>
<th>Line number</th>
<th>Line 1</th>
<th>Line 2</th>
<th>Line 3</th>
<th>Line 4</th>
<th>Line 5</th>
<th>Line 6</th>
<th>Line 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory model of the current state of the program</td>
<td>x (int)</td>
<td>y (int)</td>
<td>x (int)</td>
<td>y (int)</td>
<td>x (int)</td>
<td>y (int)</td>
<td>z (int)</td>
</tr>
<tr>
<td>Print value of z to screen</td>
<td>x (int)</td>
<td>7</td>
<td>y (int)</td>
<td>7</td>
<td>y (int)</td>
<td>5</td>
<td>z (int)</td>
</tr>
</tbody>
</table>

Activity time
10 minutes
Decimal Numbers

double num = 5.2;
num = 1.4;
num = num * 2.0;
System.out.println(num);

double fahrenheit = 98.6;
double celsius = (fahrenheit - 32) * 5 / 9;
System.out.println(celsius);

Find the Errors!

// This program has at least 4 errors. Can you
// find them all?
public class Errors {

    public static void main(String[] args)
    {
        int temperature = 80.3;
        double n = 100
        n = “Wait, what?”;
        print(“This is fine.”);
    }
}
Java has a boolean type that can take the value true or false

Booleans arise naturally when using relational operators to compare two values:

- $3 < 5$
- $3 < 2$
- $3 > 2$
- $5 \leq 1$
- $5 \geq 1$
- $5 == 5$
- $5 == 6$
- $5 != 6$
Logical Operators

- Boolean values can be manipulated with the logical operators ! (not), && (and), and || (or)

\[
\begin{align*}
! (3 < 5) & \quad \text{(false)} \\
! (3 == 5) & \quad \text{(false)} \\
(3 > 5) \land (7 < 8) & \quad \text{(true)} \\
(3 < 5) \land (7 < 8) & \quad \text{(true)} \\
(3 > 5) \lor (7 < 8) & \quad \text{(true)} \\
(3 > 5) \lor (7 > 8) & \quad \text{(false)}
\end{align*}
\]

Predicates

- A predicate is any method that returns a boolean value

```java
// determine if n is even
public static boolean isEven(int n)
{
    return (n % 2) == 0;
}

// determine if num is divisible by factor
public static boolean isDivisibleBy(int num, int factor) {
    return (num % factor) == 0;
}

// determine if n is between lo and hi
public static boolean isBetween(double n, double lo, double hi) {
    return (lo <= n) && (n <= hi);
}
```
Predicates

Write your own predicate to determine if n is odd. Then, can you write it another way?

```java
public static boolean isOdd(int n) {
    return (n % 2) == 1;
}
```

```java
public static boolean isOdd(int n) {
    return !isEven(n);
}
```

To choose between two courses of action, to control the program flow, we use conditional statements such as `if`, `else if`, and `else`.

```java
// returns absolute value of n
public static double abs(double n) {
    if (n < 0) {
        return -n;
    } else {
        return n;
    }
}
```

Conditionals

- To choose between two courses of action, to control the program flow, we use conditional statements such as `if`, `else if`, and `else`.
Iteration – **while** loop

- **Iteration** refers to a sequence of steps that is repeated until some stopping condition is reached.

\[
\text{while (boolean_expression)} \{
    \text{statement 1;}
    \text{statement 2;}
    \ldots
\}
\]

1. evaluate boolean expression
2a) if true, execute body of loop and go back to step (1)

\[
\text{int } i = 1;
\text{while } (i < 4) \{
    \text{System.out.println("CS230");}
    i = i + 1;
\}
\]

Iteration – **do** loop

- **Iteration** refers to a sequence of steps that is repeated until some stopping condition is reached.

\[
\text{do}
\{
    \text{statement 1;}
    \text{statement 2;}
    \ldots
\}
\text{while (boolean_expression)}
\]

1. body of the loop is executed
2) evaluate boolean expression
3a) if true, execute body of loop and go back to step (2)

\[
\text{...}
\]

3b) if false, go to statement after while loop
Iteration – for loop

for (init var; boolean expr; update var) {
  statements
}

(1) Execute this statement once before entering loop
(2) If true, execute body of loop
(3a) If the boolean expression was true, the body of the loop will be executed
(3b) If the boolean expression evaluates to false, drop down to here
(4) Execute this statement before next test of the boolean expression, then go to (2)

Write a for loop that prints the numbers from 1 to 10.

```java
for (int i = 1; i < 4; i++) {
    System.out.println("CS230");
}
```

Be the computer

```java
public static void main(String[] args)
{
    int x = 28; String s = "meow";
    if(x < 30 && s.length() < 10){
        x = x + 5;
        int y = s.length();
        if(x+y > 36){
            System.out.println("hello " + x);
        } else if(x+y < 33){
            System.out.println( "howdy " + y);
        } else{
            System.out.println("hi!");
        }
    } else {
        x = x - 10;
        int y = s.length() + 5;
        if( x == 15) System.out.println( "Salut " + x);
        else System.out.println("Ciao " + y);
    }
}
```