Welcome to CS230

- Email cs230instructors@wellesley.edu to get the quickest answer for all course related questions
- Class website: https://cs.wellesley.edu/~cs230/

---

Email cs230instructors@wellesley.edu to get the quickest answer for all course related questions
- Class website: https://cs.wellesley.edu/~cs230/
Class sections

Section 01 (11:20am T-F)
Section 02 (12:45pm T-F)
Section 03 (2:20pm T-F)
WITH REQUIRED LAB (Wednesdays)

Section M For students with prior Java experience sign-up here
(SCI E111@2:20 M-Th, no lab attendance, just submission)

Labs: Do them on your own

- This section 04 is special in terms of Labs: You do not have to attend a Lab section to do the weekly labs.
- Follow the steps described in every lab and complete them on your own
- Submit them on Gradescope by the end of the week
- Contact Takis or the TAs if you have questions.
  - Do not contact the Lab Instructors, please!
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 Abstract Data Types (ADTs)?

Allow you to write complex programs easier
To keep mental track of complex data interaction
To reuse code (yes!)
To improve code performance

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

Some basic ADTs:
- Collections
- Linked list
- Stack
- Queue
- Table
- Priority queue

Not so basic:
- Tree
- Set
- Graph
Application examples of ADTs

- Directory tree
- Music playlist list
- Printer queue

Social media
friendship graph

Textbook

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

- A class is like a blueprint from which you can create many of the "same" house with different characteristics

Running Java
Your first program
/* Our first CS230 program.
 * It prints out Wellesley's motto.
 */

public class Motto {
    // Program execution begins with the "main" method
    public static void main(String[] args) {
        System.out.println("Non ministrari");
        System.out.println("sed ministrare");
    }
}

First compiler was created by Admiral Grace Murray Hopper in 1952.
Using Java and BlueJ

- You can study **data structures** using **any language**
  - in this course we use **Java**
- You can **write and execute Java programs** in many ways, on the command line or with an IDE
  - in this course we will use a simple IDE called **BlueJ**

```
// This program has at least 5 errors. Can you
// find them all?
public class Errors {

    public static void main(String[] args)
    {
        String temperature = 80.3;
        int n = 100
        n = "Wait, what?";
        print("This is fine.");
    }
```
Java Constructs

Operators (relational, logical)

Conditionals

Repetitions

Variable declaration
  - int x;

Assignment statements ("gets")
  - x = 5;

Conditional statements
  - if (x == 5) { x = x+1; } else { x = x-1; } //We don’t like five!

Loops
  - while (x >= 5) { x = x-1; }

Functions (aka: Methods)
  - public static int increment(int x) { return x+1; }
Precedence of Arithmetic operators

Consider the following integer assignments and compute the expressions of the previous page:

\[ a = 10, b = 8, c = 6, d = 4, e = 2 \]

\[
\begin{align*}
    a + b + c + d + e \\
    a + b \cdot c - d / e \\
    a / (b + c) - d \% e \\
    a / (b * (c + (d - e)))
\end{align*}
\]

Answer:

30 when \( a + b + c + d + e \)
56 when \( a + b \cdot c - d / e \)
0 when \( a / (b + c) - d \% e \)
0 when \( a / (b * (c + (d - e))) \);

Relational Operators

- Java has a boolean type that can take the value true or false
- Booleans arise naturally when using relational operators to compare two values

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 &lt; 5</td>
<td>true</td>
</tr>
<tr>
<td>3 &lt; 2</td>
<td>true</td>
</tr>
<tr>
<td>3 &gt; 2</td>
<td>false</td>
</tr>
<tr>
<td>5 &lt;= 1</td>
<td>false</td>
</tr>
<tr>
<td>5 &gt;= 1</td>
<td>true</td>
</tr>
<tr>
<td>5 == 5</td>
<td>true</td>
</tr>
<tr>
<td>5 == 6</td>
<td>false</td>
</tr>
<tr>
<td>5 != 6</td>
<td>false</td>
</tr>
</tbody>
</table>
Logical Operators

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

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

Conditionals

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

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

21

### Iteration – while loop

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

```java
while (boolean_expression) {
    statement 1;
    statement 2;
    ...
}
```

1. Evaluate boolean expression
2. If true, execute body of loop and go back to step (1)
3. If false, go to statement after the while

```java
int i = 1;
while (i < 4) {
    System.out.println("CS230");
    i = i + 1;
}
```
**Do-loop vs While-loop vs For loop**

<table>
<thead>
<tr>
<th><strong>Iteration – do loop</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>- <strong>Iteration</strong> refers to a sequence of steps that is repeated until some stopping condition is reached</td>
</tr>
</tbody>
</table>

```
int i = 1;
do {
    System.out.println("CS230");
i = i + 1;
} while (i < 4);
```
**Iteration – for loop**

```java
for (initialization of index variable; boolean expn; update index variable) {
    statements
}

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

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

**Predicates (methods that return a boolean)**

- 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) {
    //...}

//determine if n is between lo and hi
public static boolean isBetween(double n, double lo, double hi) {
    //...}
```
Write your own 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);
}
```
Data Types in Java

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

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Storage</th>
<th>Range of values</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>32 bits</td>
<td>-2,147,483,648 to 2,147,483,647</td>
</tr>
<tr>
<td>long</td>
<td>64 bits</td>
<td>9,223,372,036,854,775,808 to 9,223,372,036,854,775,807</td>
</tr>
<tr>
<td>float</td>
<td>32 bits</td>
<td>Approx. –3.4E+38 to +3.4E+38 with 7 significant digits</td>
</tr>
<tr>
<td>double</td>
<td>64 bits</td>
<td>Approx. –1.7E+308 to +1.7E308 with 15 significant digits</td>
</tr>
<tr>
<td>char</td>
<td>16 bits</td>
<td>65,535 Unicode characters</td>
</tr>
<tr>
<td>boolean</td>
<td>1 bit</td>
<td>true or false</td>
</tr>
</tbody>
</table>

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

```java
double fahrenheit = 98.6;
double celsius = (fahrenheit - 32) * 5 / 9;
System.out.println(celsius);
```
Comparing Float Values

- You should rarely use the equality operator (==) when comparing two floating point values (float or double)

- Two floating point values are equal only if their underlying binary representations match exactly

- Computations often result in slight differences that may be irrelevant

- In many situations, you might consider two floating point numbers to be “close enough” even if they aren't exactly equal

To determine the equality of two floats, you may want to use the following technique:

```java
if (Math.abs(f1 - f2) < TOLERANCE)
    System.out.println("Essentially equal");
```

- If the difference between the two floating point values is less than the tolerance, they are considered to be equal

- The tolerance could be set to any appropriate level, such as 0.000001
A variable holds either a primitive type or a reference to an object.

A class name can be used as a type to declare an object reference variable:

```java
String title;
```

No object is created with this declaration.

The object itself must be created separately.

After its creation, an object reference variable holds the address of an object stored in the main memory of the computer.

Before its creation, it holds `null`.
Creating Objects

- Generally, we use the `new` operator to create an object:

```java
title = new String("Hello CS230!");
```

This calls the String constructor, which is a special method that sets up the object

- Creating an object is called *instantiation*
- An object is an *instance* of a particular class

Creating Strings

- Because strings are so common, we don’t have to use the `new` operator to create a String object

```java
title = "Java rocks!";
```

- This is special syntax that works only for strings
- Each string literal (enclosed in double quotes) represents a String object
Invoking Methods

- We've seen that once an object has been instantiated, we can use the *dot operator* to invoke its methods

  \[ \text{count} = \text{title}.\text{length}() \]

- A method may *return a value*, which can be used in an assignment or expression

- A method invocation can be thought of as asking an object to perform a service

Object References

- A primitive variable contains the value itself
  - \text{int num} = 38;

- An object variable contains the address of the object
  - \text{String name} = “Steve Jobs”

- An object reference can be thought of as a pointer to the location of the object

- Rather than dealing with arbitrary addresses, we often depict a reference graphically

  \[ \text{num} \quad 38 \]

  \[ \text{name} \quad \rightarrow \quad \text{"Steve Jobs"} \]
Assignment Revisited

- The act of assignment takes a copy of a value and stores it in a variable

- For primitive types:

  Before:

<table>
<thead>
<tr>
<th>num1</th>
<th>38</th>
</tr>
</thead>
<tbody>
<tr>
<td>num2</td>
<td>96</td>
</tr>
</tbody>
</table>

  num2 = num1;

  After:

<table>
<thead>
<tr>
<th>num1</th>
<th>38</th>
</tr>
</thead>
<tbody>
<tr>
<td>num2</td>
<td>38</td>
</tr>
</tbody>
</table>

- For object references, the address is copied:

  Before:

<table>
<thead>
<tr>
<th>name1</th>
<th>&quot;Steve Jobs&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>name2</td>
<td>&quot;Steve Wozniak&quot;</td>
</tr>
</tbody>
</table>

  name2 = name1;

  After:

<table>
<thead>
<tr>
<th>name1</th>
<th>&quot;Steve Jobs&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>name2</td>
<td></td>
</tr>
</tbody>
</table>
Aliases

- Two or more references that refer to the same object are called aliases of each other
- That creates an interesting situation: one object can be accessed using multiple reference variables
- Aliases can be useful, but should be managed carefully
- Changing an object through one reference changes it for all of its aliases, because there is really only one object

Garbage Collection

- When an object no longer has any valid references to it, it can no longer be accessed by the program
- The object is useless, and therefore is called garbage
- Java performs automatic garbage collection periodically, returning an object’s memory to the system for future use
- In other languages, the programmer is responsible for performing garbage collection explicitly
The String Class

- Once a `String` object has been created, neither its value nor its length can be changed.
- Thus we say that an object of the `String` class is immutable.
- However, several methods of the `String` class return new `String` objects that are modified versions of the original.

It is occasionally helpful to refer to a particular character within a string.

This can be done by specifying the character's numeric index.

The indexes begin at zero in each string.

In the string "Hello", the character 'H' is at index 0 and the 'o' is at index 4.
Some methods of the `String` class:

```java
String s1 = new String("Grace Hopper");
String s2 = "CU_L8R";
String s3 = " :)";

System.out.println(s1.toLowerCase());
System.out.println(s1.length());
System.out.println(s2.length());
System.out.println(s2.equals(s3));
System.out.println(s2.equals("CU_L8R"));
System.out.println(s2.charAt(1));
System.out.println(s1.substring(7, 11));
System.out.println(s2.substring(0, 2).toLowerCase());
```
<table>
<thead>
<tr>
<th>Statement</th>
<th>Result</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>string str = &quot;Ja&quot;;</td>
<td>str is set to &quot;Java&quot;</td>
<td>When applied to strings, + denotes concatenation.</td>
</tr>
<tr>
<td>str = str + &quot;va&quot;;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>System.out.println(&quot;Please&quot; + &quot; enter your name: &quot;);</td>
<td>Prints Please enter your name:</td>
<td>Use concatenation to break up strings that don’t fit into one line.</td>
</tr>
<tr>
<td>team = &quot;49 + &quot;ers&quot;</td>
<td>team is set to &quot;49ers&quot;</td>
<td>Because &quot;ers&quot; is a string, 49 is converted to a string.</td>
</tr>
<tr>
<td>String first = in.next();</td>
<td></td>
<td></td>
</tr>
<tr>
<td>String last = in.next();</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(User Input: Harry Morgan)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>String greeting = &quot;H &amp; S&quot;;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>int n = greeting.length();</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>n is set to 5</td>
<td>Each space counts as one character.</td>
</tr>
<tr>
<td>String str = &quot;Sally&quot;;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>char ch = str.charAt(3);</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ch is set to 'a'</td>
<td>This is a char value, not a String. Note that the initial position is 0.</td>
</tr>
<tr>
<td>String str = &quot;Sally&quot;;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>String str2 = str.substring(1, 4);</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>str2 is set to &quot;all&quot;</td>
<td>Extracts the substring starting at position 1 and ending before position 4.</td>
</tr>
<tr>
<td>String str = &quot;Sally&quot;;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>String str2 = str.substring(1);</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>str2 is set to &quot;ally&quot;</td>
<td>If you omit the end position, all characters from the position until the end of the string are included.</td>
</tr>
<tr>
<td>String str = &quot;Sally&quot;;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>String str2 = str.substring(1, 2);</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>str2 is set to &quot;a&quot;</td>
<td>Extracts a String of length 1; contrast with str.charAt(3).</td>
</tr>
<tr>
<td>String last = str.substring( str.length() - 1);</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>last is set to the string containing the last character in str</td>
<td>The last character has position str.length() - 1.</td>
</tr>
</tbody>
</table>

The Java API
The Java API

- A class library is a collection of classes that we can use when developing programs
- The Java API is the standard class library that is part of any Java development environment
- API stands for Application Programming Interface
- Various classes we've already used (System, Scanner, String) are part of the Java API
- Other class libraries can be obtained through third party vendors, or you can create them yourself

Packages

- The classes of the Java API are organized into packages
Import Declarations

- When you want to use a class from a package, you could use its **fully qualified name**
  
  `java.util.Scanner`

- Or you can `import` the class, and then use just the class name:

  `import java.util.Scanner;`

- To import all classes in a particular package, you can use the * wildcard character:

  `import java.util.*;`
The java.lang Package

- All classes of the java.lang package are imported automatically into all programs
- It's as if all programs contain the following line
  
  ```
  import java.lang.*;
  ```
- That's why we didn't have to import the System or String classes explicitly in earlier programs
- The Scanner class, on the other hand, is part of the java.util package, and therefore must be imported

Math Class

```java
System.out.println(Math.max(100, 50));
System.out.println(Math.sqrt(25));
System.out.println(Math.log(10));
```

// Given area of circle, returns the circle's radius.
// Since area=pi*r*r, we have r = squareroot(area/pi).
```java
public static double getCircleRadius(double area) {
    return Math.sqrt(area/Math.PI);
}
```

```java
System.out.println(getCircleRadius(100));
```
Random Class

```java
import java.util.Random;

public class RandomExample {

    public static void main(String[] args) {

        Random rand = new Random();
        for (int i = 0; i < 15; i++) {
            System.out.println(rand.nextInt(10));
        }
    }
}
```

Count Vowels

```java
// Returns true if character is lower-case vowel (a, e, i, o, u), false otherwise.
public static boolean isVowel(char ch) {
    return ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u';
}

// Returns the number of occurrences of vowels in the String s
public static int countVowels(String s) {
    int count = 0;
    for (int i = 0; i < s.length(); i++) {
        char c = s.charAt(i);
        if (isVowel(c)) {
            count++;
        }
    }
    return count;
}
```