Unit 1 Notes

08/21/23 - Welcome and Intro

Teaching approach / concepts

- Based on Psychology of learning (4 week cycles)
  - spacing
  - interleaving
  - practiced recall
  - elaboration
  - reflection
- Grading
  - Formative - Can be redone
    - most things can be redone up until the end of the unit
  - Summative - Demonstrate what you know
    - coding exam
    - canvas exam
      - summative but kinda formative
      - need 100 readings and 100 quizzes to redo
- What you are learning
  - A new language
  - A different way of thinking (divide-conquer-glue)
  - ok to struggle

Course Structure - Follow Canvas

- Readings

    - Due before lectures - sunday, tuesday, thursday nights
- Lectures, attendance, worksheets
    - Active learning, a lot of group discussion and coding - BE HERE - or you will miss out
- Labs
    - Meant to be done after lecture due on lab day
- Knowledge Checks / Retrieval Practice activities
    - Your best study tools
- Exams
    - Canvas Exams
        - reading content
    - Coding exams
        - writing content
- Practical
    - these are large and hard programming assignments (usually 2 weeks), that bring it all together

---

8/23/23 - Intro to Programming

Computers hold STATE

- Think of water states
    - liquid, ice, gas
- Memory has 2 states (binary)
    - On and off / yes and no / true false / 0 1
- This is stored in a BIT
- $8 \text{ bits} = 1 \text{ byte}$, basic unit of information

- All the representational power of computers
    - a bunch of bits
    - representing various states
- Difficult to write it all out
    - `0101010110001101100111000` (gross)
    - therefore we use programming languages

Coding

- Coding used to be literally moving physical wires
    - and was unique to every machine
- Then code became compiled
    - from english to 1s and 0s
- Who do we have to thank?
    - Admiral Grace Murray Hopper
        - designed the COBOL lang
- Her dream was to write once, run everywhere
    - but it took may years for it to come true

Java Bytecode

- Write once - run most anywhere
    - James Gosling - lead designer
- Programs are a set of instructions
    - written in english - very specific instructions
- Android is the most used os in the word
    - written in java

8/28/23 - Identifiers, Variables, Operators

Types

- Type tells the computer how much room to save
- int
    - whole numbers only
    - 1, 2, 3, 101023
- double
    - floating point numbers
    - 1.0, 2.34, 123.123213
- char
    - Every character on a keyboard stored as an int
- boolean
    - true or false
- String
    - collection of ordered characters
    - it is more unique (object)

Variables

- Identifiers are WORDs
    - you use the to *hold* information
    - cannot be a reserved word
    - cannot start with numbers or special characters outside of underscore
    - use real words `int x` doesn’t mean much, but `int puppyCounter` has meaning and readability
- Declaring variables
    - `<TYPE> <IDENTIFIER>;`

    - Can be declared in the same line
    - declaring reserves or allocates memory
- examples

```
int myInt;
double myDouble;
int x, y, z;
double db1, db2;
String firstProgrammer;
```

Assigning / storing values

- single equals sign assigns values
- the value must match type
    - Strongly type'd language
- You can change the value as much as you want
    - but it must still be the same type
- Assigning the first time is called initialization
    - often done in the same line as declaring
- objects are `null` if not assigned

Operators

- operators are MATH
    - = assignment
    - + add
    - - subtract
    - / divide
    - * multiply
    - % modulo - remainder
- Numeric types

  - `int` - always whole number
  - `int myVal = 1 / 2; // evaluates to 0!`
  - `double` - has decimals
  - `double doubleVal = 1.0 / 2; // evaluates to 0.5!`

Integer Division

- `int val = 5 / 6; // sets val to 0`
- `int val2 = 10 / 3; // sets val2 to 3`
- you lose the decimal point
  - truncates, does not round
- this is a very, very common thing
  - both to our advantage
  - and often to our error

Modulo

- `modulo (%)` gives you the remainder
- 4th grade math
- this example

```
int x = 250 % 6; // would be 4

// so combining them

int whole = 250 / 6; // 41
int remainder = 250 % 6; // 4
```

What are some cases to use it?

- Forming groups
  - the remainder is always between 0 and 1
  - Value % 6 has a range of 05

    - Value % 4 has a range of 03
- think about rolling dice
    - `Math.random() % 6;` random number between 0 and 5
- Determining even and odd - `Math.random() % 2` if 0, even, if 1, odd.

Scanner

- Read something from the console / terminal
- First needs to create an object of the class Scanner
    - `Scanner in = new Scanner(System.in);`
- Second needs to know the type of variable that wants to read - that will define the method you ill use
    - `int val in.nextInt();`
    - `double valD = in.nextDouble();`
    - `String str = in.nextLine();`

Convenience Operators

- We often find ourselves doing things like
    - `int value = 100;`
    - `value = value + 10;`
- introducing operator plus assignment
    - `value += 10;`
    - +=
    - -=
    - /=
    - *=
    - %=
- We also like to add and subtract by 1

    - `--value and ++value`

Examples

```
int puppyCounter = 100; // So many puppies!

String puppyName = "Spot";
String puppyLongName = "Cerberus";

double amountOfFoodPerDayLbs = 20.56;
amountOfFoodPerDayLbs = amountOfFoodPerDayLbs + 10.0 // assigns 30.56

boolean isPettable = true;

char singleLetter = 'c'; // one letter, single quote
```

8/30/23 - Objects and Methods

Programming == Problem Solving

- You look at the problem and constraints
    - Clarify the problem and constraints
- Break it up into *smaller* parts (divide)
- Outline the steps needed
    - Solve each step (Conquer)
- Reassemble the pieces (Glue)
- Completed program

Methods

- Methods are ways to modularize / reduce the code
- Methods are designed to implement a specific function in our program
    - Small / repeatable blocks
- Methods are defined inside a class

- When we build a method inside a class that has a main method the method needs to have the following format
  - `public static <TypeOfReturn> methodName(<ListOfParameters>) {}`
  - `public static`: access mode is public, can be called without restrictions
    - `static`method belongs to the class
  - `<TypeOfReturn>`
    - `void`: no return
    - Any class or primitive type
  - `<ListOfParameters>`
    - list of parameters separated by commas
    - Each parameter needs to have a type and name

Objects

- Objects are building blocks
- think of legos
  - blocks
  - assembled in different ways - creates new and interesting things
- Objects contain information in a logical order
- Most objects use the `new` keyword

```
MyCoolObject obj = new MyCoolObject();
obj.myCoolMethod();
```

- We will keep coming back to this

> 💡 Methods belong to objects

  - Even methods that you write

Rectangle? Is a Class?

- In Java (an oop lang) everything must be in a class
- you can create objects out of classes
  - use the new keyword

    ```
    Rectangle myHouse = new Rectangle();
    Scanner scnr = new Scanner(System.in);
    ```

- the `new` keyword reserves memory for that instance / object

Rectangle Class

```
public class Rectangle {
  private int width; // instance variables
  private int length;

  public void setLength(int length) {
    this.length = length;
  }

  public void setWidth(int w) {
    width = w;
  }

  public int calculateArea() {
    return width * length;
  }
}
```

- Instance Variables
  - Represent the Data
- private
  - Means that only the class can access those values directly
- public
  - Others can access public methods
- `this` - keyword

    - means “this object / instance
    - helps keep track of which variable
    - common practice - but not required

Class Constructor

```
public Rectangle() {
  width = 0;
  this.length = 0;
}

public Rectangle(int w, int l) {
  width = w;
  length = l;
}
```

or

```
public Rectange() {
  setWidth(0);
  setLength(0);
}

public Rectangle(int w, int l) {
  setWidth(w);
  setLength(l);
}
```

- Special method that has the name of the class
- No return not even void
- Can be overloaded
    - meaning that we have more then one implementation for the method
    - same name with different parameters
- `Rectangle()` - no parameters
- `Rectangle(int w, int l)` - with parameters
- You can call methods inside of the constructor

- Usually, you call mutators (sets) methods, if the class has them defined

Class Instance Methods

```
public void setLength(int length) {
  this.length = length;
}

public void setWidth(int w) {
  width = w;
}

public int calcArea() {
  return width * length;
}
```

- static methods
  - belongs to the class / self-contained
- Instance methods
  - need to access the instance variables
  - uses the data in the object
  - unique to that instance

Why methods and objects? DRY Code

- Code should be **DRY**
  - **D**on't **R**epeat **Y**ourself
- code should be
  - reusable
  - small snippets
- Reusable code
  - only write once
  - use in multiple applications
- Java

  - objects are blocks of information, with reusable code / methods
  - methods are blocks of reusable code
    - ideally, no more than 20 instructions
  - clue: if you are cutting and pasting code - it should be a method

---

9/1/23 - Character & Strings

Characters (Char)

- Primitive - stores values only
- Example declarations

  ```
  char myChar = 'x';
  char newLine = '\n';
  char myChar2 = 57;
  char what = (char)('A' + 2);
  ```

- A char is an int behind the scenes - and java uses the ASCII table to figure out the character value

Casting

- Type casting is when you assign a value of one primitive data type to another type
  - this allows you to change an int back to a char without an error

  ```
  public static char characterAddition(char a, int b) {
    return (char)(a+b);
  }
  ```

Let’s talk about String

- A `String` is a collection of ordered characters
  - it has data

    - it has functionality (methods)
    - It is also immutable (Can’t be directly modified)
        - Every method that builds a string, returns a copy
        - will cover this more in the future
- But what does a String really look like?

First, Let’s Think about memory

- Storing variables in memory
    - Wherever the computer decides
    - Value stored in the location
    - Variable name points towards location
- Example

```
char c = 'c';
char m = 'm';
```

String Methods

- Ordered collection of characters
    - Indexed - starting at 0
    - ‘c’ is at index 0
    - ‘a’ is at index 1
- `char character = mascot.charAt(2);`
    - returns ‘M’
- How many characters?
    - `int len = mascot.length();` = 3
- No matter how large the string
    - starts at 0

    - has a length
    - always know first and last index

String Concatenation

- Adds two strings together
    - better worded: String objects added to other types (string or primitive or other objects)
- Step 1 converts everything to a String. So 1 number becomes "1"
- Step 2 puts it all together, exactly as written

    ```
    int aNumber = 42;
    String question = "The answer to life";
    System.out.println(question + aNumber);
    ```

String.format / printf

- We know we can write strings via concatenation
- However, that can be awkward - introducing `String.format`

```
String towelColor = "pink";
String dontPanic = String.format("Don't forget to bring your %s towel", towelColor);
String txt = "Don't panic";
String heading1 = String.format("<h1>%s</h1>", txt);
```

Format specifier	Data Type(s)	Notes
`%c`	char	Prints a single unicode character
`%d`	int, long, short	prints a decimal integer value
`%o`	int, long, short	prints an octal integer number
`%h`	int, char, long, short	prints a hexadecimal integer value
`%f`	float, double	prints a float value

Format specifier	**Data Type(s)**	**Notes**
`%e`	float, double	prints a float in scientific notation
`%s`	String	prints the characters in a String variable or literal
`%%`	N/A	Prints the “%” character
`%n`	N/A	prints the platform specific new-line character

Floating point values

Sub-specifier	Description	Example
width	Specifies the min. # of chars to print. If the formatted value has more chars than the width, the value will not be truncated, if the formatted value has fewer characters than the width, the output will be padded with spaces (or 0’s if the ‘0’ flag is specified	`printf("Value: %7.2f", myFloat); Value: 12.34`
.precision	Specifies the number of digits to print following the decimal point. If the precision is not specified a default precision of 6 is used	`printf("%.4f", myFloat);` `12.3400`
flags	-: left aligns the output +: prints a preceding + sign for positive values. Negative numbers are always printed with the - sign 0: pads the out with 0’s when formatted value has fewer characters than the width space: prints the preceding space for positive value	`printf("%+f", myFloat);` `+12.340000`

9/6/23 - Branching

Basic Conditionals

- Logic that evaluates as
    - yes or no
    - true or false
- Essential in all programming langs
    - you do this all the time
    - 100 pennies grater than $1?
- Conditional operators
    - `==` equals
    - `<` less than
    - `>` greater than
    - `<=` less than or equal
    - `>=` greater than or equal
    - `!=` not equal

Conditional Expressions

- Conditional operators
    - Evaluated second to last (storing values or returns last)
- Always two sides / pairs
    - means it is true or false
    - boolean primitive can store it, and you can return boolean from a method

If Statements

- if / else statements
    - only run instructions based on true or false values

    - essentially, choose to run certain lines of code or skip them

```
if (puppies > 100) {
  System.out.println("So many");
} else {
  System.out.println("where they at");
}

if (puppies == 0) System.out.println("dang");
```

- they can be nested

```
if (something) {
  if (somethingElse) {
    if (anotherThing) {
      System.out.print("too much nesting = gross");
    } else {
      if (why) {
        return stop;
      }
    }
  }
}
```

9/8/23 - <u>Loops</u>

What are computers good at?

Computers are good at three things
- Calculations
- Formal Logic
- Repeating what you just asked it to do
- Iterations

These three things are coincidently three things that humans tend to struggle with

If you understand
- Calculations
- Formal logic

- And loops
You will be able to create impressive programs

Loop types in Java (and most languages)

- While loops
- for loops
- for:each loops
- do while loops
- All loops have
  - an iterator
  - a condition to exit

While Loop

While loops are good when
Your iteration variable is created outside the loop
You need your iterator variabe utside of the loop

```
while (true condition) {code}

int start = 0;
int end = 2;
while (start < end) {
  start ++;
}
```

For Loop

Good when
Your loop has a set start and end
You don't need to keep your iterator ouside the loop

```
for (variable; condition; iterator) {code}

for (int i = 0; i < 10; i++) {
  System.out.println(i);
}
```

9/11/23 - Javadocs and UML

Javadoc

- Parses source code along with specially formatted comments to generate documentation
- the documentation generated is known as an API
- start with `/** <enter>`
- You will have

  ```
  /**
  *
  */
  ```

UML

- UML (Unified modeling language)
  - is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems