Day 2 - January 30th, 2023

---

2 Data Types:

- Primitive - small, single data
  - int, float, char, bool, etc.
- Objects/Class (most are classes)

Static vs Instance:

- If a variable belongs to the class itself, it is a static variable
  - Variable is shared among all instances
- If a variable belongs to the object, it is an instance

  Example:
  - The color and gender of a cat depends on the *instance* of each cat. It is an *instance variable*.
  - The number of legs of a cat is *the same* for all cats. It is a *static variable*.

```
class Cat {
     static int Num_Legs = 4;    // Static Variable
     string gender;              // Instance Variable
     string breed;               // Instance Variable
}
```

Instance variables change with each cat
Static variables don't change with each cat

Composition (Referencing)

```
     int leg = Cat.Num_Legs;          // Static comes from Class
     string myCatBreed = pet1.breed;  // Instance comes from instance
```

Static Variables CAN be changed

If you don't want it to change, use the *final* modifier:

```
     static final int Num_Legs = 4;
```

With n bits, integers in the range $-(2^{n-1})$ --> $(2^{n-1})-1$ can be expressed

Day 3 - February 1st, 2023

Instance = Object = Specific
Static = Class = General

Methods - Functions in Java

- pet1.getBreed()
  - Get method returns value to outside variable
- pet1.setNeutered()
  - Set method changes a value within the object
- Cat.getNumLegs()
  - Static methods also exist with respect to the class

Class String:

- String s = new String("Hello");
- String s = "Hello";
  - Both work the same, but all other classes require "new"

String Methods: have parenthesis; attributes do not (.length)

- int length() - num of characters in string
  - System.out.println(**s.length()**); ---------> 5
- char charAt(**<int>**) - returns character at position <int>
  - char c = s.charAt(1);
  - System.out.println(c); ---------> e
- bool equals(**<string>**) - determines if two strings are equal
  - if(s.equals("Goodbye"))...
  - if(reply.equals("STOP"))...
- bool equalsIgnoreCase(**<string>**) - equals but ignores case

Let's say we want to count the number of 'o's in a string:

```
for(int i=0; i < s.length(); i++){
|    if(s.charAt(i) == 'o'){    // == used to compare primitives
|    |    oCounter++;           // .equals used on classes
|    }
}
```

Day 4 - February 6th, 2023

---

subArrayLength returns the length of the subarray calculated in the function inputFromFile

Short.parseShort(line) is a wrapper class that turns the string representation of a number into a short representation that can be used with the array of type short. This is Java's version of typecasting strings to primitives, and most primitives have one.

The value 'null' means "no value". It does NOT mean 0.

Day 5 - February 8th, 2023

When calling a method, we pass in "actual parameters"
Inside the method, the shadow variables are called "formal parameters"

Primitive type parameters are **PASS BY VALUE** - actual parameter variable does not change if the formal parameter is changed inside the method. The **VALUE** of the variable is sent.

Object type parameters are **PASS BY REFERENCE** - actual parameter variable is sent through the method, and the value is changed inside the method. The **ADDRESS** of the variable is sent.

Changes to an object INSIDE A METHOD are permanent
- *see "Methods and Parameter Passing" slides 15*

If you want to change the values of an object (array) in a method (like *Selection Sort)*, you must pass the array into the method so it is passed by reference.

Day 6 - February 15th, 2023

Program Modularity and Error Handling

Day 7 - February 21st, 2023

GUI and Inheritance - Using a Simple GUI

JFrame - object that deals with panes/windows;import javax.swing class
.setSize(*width, length*)
.setLocation(*x, y*) - relative to top-left corner of screen/JFrame

We do not need to rewrite the code for our new class that mimics another. We have inheritance, which will copy all of the elements of one class into the new one.

```
public class SSNGUI extends JFrame(){
     //extends JFrame allows for the inheritance from JFrame to SSNGUI
     //SSNGUI is the subclass of JFrame, the superclass
}
```

The ContentPane can be divided and sized to allow for multiple TextAreas or other objects using a LayoutManager, such as:

- BorderLayout - divides into North, East, South, West, Center
  - Leaving sections blank will squish the sections together
  - myContentPane.add(myTextArea, BorderLayout.EAST);
- GridLayout - divides into specific number of rows and columns
  - All sections are the same size as one another
  - mySSNGUI.setLayout(new GridLayout(1,2)) //#rows, #cols

Day 8 - February 22nd, 2023

You can have two constructors of the same name and type SO LONG AS they have different signatures (different names AND parameters)

```
public SSNGUI(String title){
public SSNGUI(String title, int height, int width){
```

These are two different constructors that can be called independently of one another depending on the parameters passed in during
SSNGUI *name* = new SSNGUI()

For Project1, put all files in a single project (folder)

Day 9 - February 27th, 2023

Slide 3 shows basic class setup

Error Checking:
Public variables can be changed by the user
Private values can only be changed within the method

throw new <errorType>(“Error Message!”)

Private methods cannot be called from outside the object it is defined inside of.

A method is static if it is not a behavior of the object; it is not concerned with anything about the object.

Overriding Methods of Class Object
All objects automatically inherit from the class **Object**

Slide 8 **getClass() returns class of the object the method is in**
(SSN) is type casting other from Object to SSN
This will only work if other was declared with the intention of becoming a SSN. It needs all of the required information

Note: all methods on these slides are located inside the same class file, the SSN class file

this operator - used to refer to the class it is called in
Note: scope of variables applies locally, so the more local variable will be called

Day 10 - March 1st, 2023

Analysis of Algorithms

Efficiency can be measured in terms of runtime or file space. Let's say we want to see which program, A or B, is more efficient.

We can measure the time it takes to run a program by setting a timer and measuring how much time it takes for each program to complete.

- This requires us to write, debug, and run the code. This takes effort and time, of which I have very little to give
- Not only this, but one program gets scrapped after all this time and effort has gone into creating the program

As such, we want to come up with a way to analyze the efficiency of code without writing the code.

For this course, we will perform "Worst Case Analysis". If something could happen to make the runtime longer, let's assume it will happen. How long will it take to run?

- This requires us to look at the size of the problem, n, which will be very large (we can easily sort 5 numbers, so no need to analyze for small n)

Order function - Big O() [like in Discrete 220]

O(c) - constant time; does not change as n changes - very good
ex) program that multiplies: ans = n*2;

O(n) - linear time; changes proportionally to the size of the problem
ex) printing the elements of an array of size n

$O(n^2)$ - quadratic time;
ex) selection sort of an array
We had two nested loops to find the largest number and then to order the whole array

Each nested loop increases the degree of n in $O(n^x)$

Binary search: You have a sorted array and you want to see if something is in the array.

1. Divide the array in half according to the subscripts
2. Call the middle box k; the left half contains all values less than k. The right contains all values greater than k.
3. Compare k to the value you want to find and pick the array (L/R) you want to search through.
4. Repeat the steps until you are left with an array of length 1. If it contains your value, it is in the original array. If it does not, then it is not.

The length of the array goes from n to n/2 to n/4 …
Binary search is $O(\log_2[n])$

How many games of chess can be played?

One node branches to other nodes, each of which branches to other nodes. You get a tree (data structure) of $n \rightarrow n^2 \rightarrow n^3 \rightarrow \ldots n^n$

For a simpler example, say each turn, there are 2 possible moves.
1 -> 2 -> 4 -> 8 -> … 2^n

These are examples of exponential time $O(2^n)$

List of all order functions mentioned so far:

$O(c)$ - constant time; does not change as n changes - very good
ex) program that multiplies: ans = n*2;

$O(n)$ - linear time; changes proportionally to the size of the problem
ex) printing the elements of an array of size n

$O(n^2)$ - quadratic time;
ex) selection sort of an array
We had two nested loops to find the largest number and then to order the whole array

O(log[n]) - logarithmic time;
ex) Binary search - we cut an array in half repeatedly until there is only 1 element left

$O(2^n)$ - exponential time;
ex) chess positions - each move leads to a wide variety of new moves

Day 11 - March 6th, 2023

Static Structures have a fixed size and cannot be changed.
- Arrays, for example, are set in size when they are initialized. You cannot add an element to the array at the end or in the middle unless you delete another element.

Dynamic Structures use the exact amount of memory that it needs. Their size and element ordering can be changed

Real class for a “node” in a linked list:

```
// 1) Data in the node such as a string
// 2) Reference to the next node in the list

class ListNode{
     String data;
     ListNode next;

     // constructor
     public ListNode(String d, ListNode ln){
          data = d;
          next = ln;
     }
}
```

We need a first node that contains no data and points to the second node which contains data. We also need the last node to point to null. The list will be stored in that first node

```
public class LinkedList{
      private ListNode first;
      private ListNode last;
      private int length;   // length of Linked List

      public void append(String s){
            ListNode n = new ListNode(s);
            last.next = n;
            last = n;
            length++;
      }

}
```

Midterm Notes

```
String inputLine = JOptionPane.showInputDialog(null,"Enter a string");
JOptionPane.showMessageDialog(null, “There are” + sum + “digits”);

//Note: ^^^ Requires import.javax.swing.*;
```

Constructor:

```
public <className>(parameters){

}
```

get’s just return
set’s check and set

JFrame = entire window

```
JFrame mywd = new JFrame();

            mywd.setSize(700,700);

            mywd.setLocation(0,0);
```

```
        mywd.setTitle("My Window");

        mywd.setVisible(true);
```

Text Areas/Content Panes

```
    import java.awt.*;

    public static void main(String[] args) {

        JFrame mywd = new JFrame();


        Container myContentPane = mywd.getContentPane();

        TextArea myTextArea = new TextArea();
```

PRIVATE, PUBLIC, & STATIC

- A method/variable is **private** because it should not be accessible from outside of the object it is defined in

- A method is **static** if it is not a behavior of the object; it is not concerned with anything about the object.

Day 14 - March 15th, 2023

“Is a” relationship - If x **is a** y, x **extends** y

Private instance variables are not inherited; they exist only in the class they are declared.

- The “protected” modifier grants access to lower classes
- The “public” modifier grants access to any class
- The “private” modifier grants access to no class other than itself

All variables that get inherited should be protected

When a new CUNY Queens College Undergraduate Student is created, you need to create an instance of a Queens student, which requires you to create an instance of a CUNY student. Call the constructor of a super class using super(<parameters>). Must be the first line in the lower constructor (**SLIDES ARE A GREAT EXAMPLE**)

Polymorphism - all objects take the class of their predecessors. For example, a CUNY Queens College Undergraduate Student is a:

- CUNY Queens College Undergraduate Student
- CUNY Queens College Student
- CUNY Student

When an object is printed, an automatic call to the method toString() is called.

Abstract Classes: “Go to the pet store and get me a pet” - which one??

- Cannot be instantiated
- A class is abstract if:
    - It is declared as abstract
    - It contains an abstract method
    - It inherits an abstract method and does not overload it

An example of abstract classes would be CUNYStudent and QueensStudent from the student example previously. You can only instantiate a new (Under)Graduate student, not a new CUNY or Queens Student

instanceof operator will tell you what class the object is - more often used when inheritance comes into play

getClass() is used when you don’t know what the class will be
Instanceof is used when you have a general idea of the class

Day 15 - March 20th, 2023

Shape example - see Blackboard

1. A “shape” class is pretty generic: it is an abstract class
    a. A “circle” class is a child of shape, so it will extend “shape”. A circle may have instance variable such as coordinate and radius
    b. A “polygon” class will extend “shape” as well, but there are many kinds of polygons. A good instance variable for this class would be numSides
        i. A quadrilateral would extend polygon
            1. A rectangle would extend quadrilateral
                a. A square would extend rectangle

The inheritance hierarchy may look like this:

- Shape(abstract)
    - Circle[coord, radius]
    - Polygon[numSides]
        - Quadrilateral
            - Rectangle[height, width]
                - Square[side]

The code for these classes is on Blackboard under Inheritance and Polymorphism

Additions:

- Shape Class
    - Needs constructor public Shape(){}
- Polygon Class
    - Constructor needs to call “super();” first;
- Rectangle Class
    - Instance variables should be “protected int height, width;”
- Square Class
    - **Should** have a constructor with no parameters to create a unit square, calling “super();”

Day 16 - March 22nd, 2023

GUIs and Event-Driven Programming

Clicking on something is an event

Main program just needs to instantiate the GUI

Inside the GUI, we create the menu

We need to write the createMenu() method in the GUI file

Inside createMenu(), we create the “Open” and “Quit” items
Can add as many as you want

At the end of createMenu(), we want to add the fileMenu to the menuBar and put the menuBar on the GUI

When we click on the items, we want something to happen. We use an event handler for this. We will call it FileMenuHandler and it is its own class. It will be passed the parameter this, which is a reference to the GUI since createMenu() is a method written in the GUI class
We then add an “action listner” to the item.

One handler can handle multiple events. Every event needs a handler.

An interface “if you’re gonna do something, you gotta do it completely, so here’s everything you need to do”. It is used as a way to check that everything has been met.

ActionListener is an interface

We create the FileMenuHandler(this), create the items, and register each with the FileMenuHandler

FMH points back to the class itself

<Slides show the procedure of doing this>
Everything is now connected together

---

actionPerformed() is located within the FileMenuHandler class and determines what method to call

openFile() is located within FileMenuHandler. It allows you to choose a file using the JFileChooser() as shown on the slide
readSource is a helped method that takes a file object and reads the contents of the file

readSource() code
Note: TextFileInput will only search your working directory. If the file is not found in the working directory, an error will result. Rather than using File.getName(), use File.getAbsolutePath(), which avoids this issue.

The appending is in the FileMenuHandler class, which contains all of the helper methods

Day 17 - March 27th, 2023

We know how to throw exceptions using **throw new <Exception>** but what if we want to make our own errors? What about **catching** errors?

Exceptions are objects, so we just need to extend the IAE object and construct it by calling super with our new message.

Exceptions are objects, and therefore part of a hierarchy
Note the difference between RuntimeExceptions and IOExceptions

When an exception is thrown, the Runtime System (JVM) looks for a method that can handle the exception. It does this by tracking the code backwards to find a method. If a method doesn’t exist, the Runtime System handles the exception and termination. (See animation)

Calling methods A,B,C are *exception propagators*.

The Try/Catch Block

```
try {
      problematic code that may throw an exception
}
catch (Exception e) {
      statements to execute if an exception happens in the try block
}
```

If an exception is thrown and caught, code resumes after the final catch spot

But catching any exception is not specific enough. If we want to catch a **SPECIFIC** exception, we see what error will be thrown and we replace `Exception` with `NumberFormatException`

If we have a try block that can throw multiple exceptions, we use multiple catch blocks. The order of catch blocks matters, because the computer goes through the catches top to bottom.

Statements can be skipped in the try block. If a statement throws an error, all following statements in the try block will be skipped when the code jumps to the catch block

A *finally* block will ALWAYS execute whether an exception occurs or not. It follows the catch block, and ALWAYS executes, even if a return is used

Checked and Unchecked Exceptions:
Aside from *RuntimeException*, all exceptions need to be caught or propagated. As such, *RuntimeException* is an unchecked exception

Day 18 - March 29th, 2023

A **regular expression (regex)** is a pattern that can be matched against a string
This has the effect of simplifying work

import java.util.regex.*;
Pattern comes from this class^
(Slide demonstrates usage)
Take the string SSN_Pattern and compile it into an internal processable format. Assign it to p.
Assign m the matcher of it
Return m.matches
Pattern gives the pattern
Matcher is the substring of the pattern found in the long string

Regular expressions allow us to put characters inside brackets, which allows us to match them to the string we are looking for.
(See Slide for syntax and examples)

The SSN_PATTERN expression only requires one match. So anything with a single digit 0-9 will match. This is not what we want

There are also predefined character classes that will make this easier. They are listed on the slide.

“\\d” == “[0-9]”. We need to do better (\\ for escape key literal)

Quantifiers also help us. They are listed on the slide

“\\d{9}” as a SSN_PATTERN expression will match any string that contains exactly 9 digits in a row. However, it will also match a string that has additional non-digit characters

^ signifies the beginning of the regex
$ signifies the end of the regex
“^\\d{9}$” literally means

- Start the regex
- Get 9 digits

- End the string

This works, but what if we add hyphens between the numbers? We want to accept this format, but this regex will not accept it

A correct quantifier for the regex is “^\\d{3}-?\\d{2}-?\\d{4}$”

- Start the regex
- The next(first) 3 characters are digits
- Next, there may or may not be a hyphen
- The next 2 characters are digits
- Next, there may or may not be a hyphen
- The next 4 characters are digits
- End the regex

This will, however, accept something of the form
999-999999 or 99999-9999. We don’t want this

The **split** method works like a tokenizer. It breaks up items separated by a delimiter, as shown in the code on the slide. Returns a string array. Note that inside the Pattern.compile(“”) are the delimiters (comma and whitespace \\s). + signifies one or more. You can put however many delimiters you want.

Slide demonstrates how you can pull out number strings from a longer string with noise
Pattern gives the pattern
Matcher is the substring of the pattern found in the long string

How to capture the substring that matches the pattern:
Use parentheses to signify “I want to capture this”
group(0) is the match to the entire pattern
group(1) is the first capture / match in parentheses

Slide demonstrates how captures can be used to get around the issue of only one hyphen
The regular expression says

- Find 3 digits and capture it (capture 1)
- What comes next is either a dash or a dot. Capture it (capture 2)
- What comes next is 3 digits
- What comes next is WHATEVER WAS IN CAPTURE 2 \\2

        - Note: If the first capture wasn’t taken, this would be \\1
    - What comes next is 4 digits

Slide shows password validation regex / password requirements. Don’t need to fully understand, just an example of how powerful regex can be

Slide shows email address validation regex

Day 19 - April 3rd, 2023

Java Collections Framework

ArrayList and LinkedList inherit from InterfaceList

.add() appends unless an index is specified
.remove() automatically searches the ArrayList for the first instance of the specified element and removes it. You can also specify an index and it will remove that element
.set() goes to a particular cell of the ArrayList and sets the value to the specified value. Essentially an override
.indexOf() returns first index of specified value
.get() returns the value of ArrayList[specified index]
.size() returns the number of elements in ArrayList
.contains() returns bool - is the element in ArrayList?

.add() appends unless an index is specified????
.remove() automatically searches the LinkedList for the first instance of the specified data and removes that node. You can also specify an index and it will remove that node.
.set() goes to a particular node of the LinkedList and sets the data to the specified value. Essentially an override
.indexOf() returns first index of the node containing specified data
.get() returns the data of LinkedList of specified node
.size() returns the number of nodes in LinkedList
.contains() returns bool - is the data in LinkedList?

You can convert arrayList into LinkedList using LinkedList.addAll(ArrayList)
You can prepend or append to a LinkedList using .addFirst() and .addLast()
You can retrieve the data at the ends of the list using .getFirst() and .getLast()

Day 20 - April 19th, 2023

Java Collections Framework (cont)

Interface Map - The interface contains the method definitions
The “key” is the unique identifier, the “value” is the object the key represents

2 Main maps: Hash Maps and Tree Maps

Class HashMap
Hash function maps keys to an index in the hash map --- h(“cat”)
Elements are not stored in order of addition
It may be the case that hashing “dog” will give index 0, which is already occupied; this is a collision

Can check if hashmap contains a particular value or key

You can create an iterator to iterate through all of the elements of a HashMap
keySet() is the set of all keys; entrySet() is the set of all values
Searching a hashmap can be done in O(c) - constant time
Elements do not come out the same way they went in!
Do not use a HashMap if order matters!

Class TreeMap
TreeMap <*type of key*, *type of value*> = new TreeMap <*type of key*, *type of value*> ();
You put in and get out the same way as the HashMap

TreeMaps arrange data keys in a predetermined, sorted order of the keys

The order can be determined if the class implements Comparable

For user defined objects, such as SSN or RomanNumeral, the TreeMap needs to know how to order the keys
A class that implements Comparator has a method int compare(obj, obj)

The TreeMap is based on the Red-Black Tree
Every node has two children nodes, a left child and a right child
Each left child has a lesser value than its parent
Each right child has a greater value than its parent
Searching a tree map involves searching the greater or lesser branch to find the key being searched for. It is done in O($\log_2(n)$)

Day 21 - April 24th, 2023

SCANNER

The main purpose of a scanner is to read and parse information from a file.
If being used, surround with try/catch block for FileNotFoundException. Can get around by using a file chooser to guarantee the file exists

The scanner can also read input from the keyboard

Scanners are a little more complex than reading from a file; it works with REGEX and delimiters to parse and check the line read in.

Scanners are only used for input sources. To output to a file, use a FileWriter.

Class File can take an absolute path for its constructor

With a File object, there are many things we can do, as shown on the slide

---

GENERICS

The object **Comparable** has the method compareTo, which takes an object. If we say we have a variable Comparable c that is a new Date() and we compare it to the string “red”, that makes no sense. How do you compare “red” to “0502200209020”?
However, according to the methods, we can pass “red” to the compareTo method and will try to compare them. It will try, fail, and return an exception. This will be a RunTime error

Generics allow us to say “we are gonna take some type of information and use a placeholder for it. The type of parameter o is not an object, but rather the type used to instantiate the comparable class. What type of information do I want to use for the parameter for Comparable?”
We have a new date object assigned to c, and it is comparable with dates. In the interface above, <T> gets replaced with <Date>, and when we call compareTo“red”, the compiler will not allow this. This will be a Compiler Error, better than RunTime.

In the SSNListNode class, the only data that can be stored is type SSN. If we wanted to use this class to store RomanNumerals, we would have to rewrite the class.

We could have made ListNode a general structure by making the type of data Object, which would make it general, but not generic. This is because we could have a linked list where the first node is a SSN, the second is a RomanNumeral, the third a char, etc. We want them to contain the same type everywhere.

We can make the ListNode generic by giving the class an element type (first <E>) in order to restrict the type to only one.

The same can be done with the LinkedList by using the generic placeholder <E> in the class as well as the append function. Wherever data is being added, that is where you'd need the placeholder.

Concrete example making a LinkedList of strings

Concrete example making a LinkedList of SSN

Day 22 - April 26th, 2023

Recursive Programming

Activation records are stored on the RunTime Stack
Each call to function adds a new activation record to the RTS

Can print LinkedList in reverse by recursive-ing through until last is reached, and then returning to go backwards

Day 25 - May 8th, 2023

Model/View/Controller

Model - How the data is stored
View - (Display) What the viewer sees
Controller - Received and updates the data

Model

- representation of data (for example, an array whose indices represent temperature, wind speed, wind direction, etc.)
- must extend "Observable" class to be used in the Model/View/Controller

View
- Display of the data using GUI components by observing the model
- Implements “Observer” class

Controller
- A *Listener* that responds to events and updates the *Model*
- For example, “raise” and “lower” buttons to control the temp.

Day 26 - May 10th, 2023

Threads

A thread/process is an instance of program execution
Most Java applications have multiple threads

Process States - Ready, Running, Waiting
The OS or JVM is responsible for moving threads between states

Threads are objects and have methods

Day 27 - May 15th, 2023

REVIEW

The source code (.java) compiles into bytecode (.class) for the JVM to run

Primitives are single values variables
A class is a blueprint for the data and behaviors
An object is an instance of a class
Instantiation with “new” takes memory from the storage pool

Know how selection sort works (not memorized)
- Find the location of the smallest number, swap with the top of the array. Repeat, changing the “top” of the array

Primitives are passed by value
Objects are passed by reference

Inheritance from JFrame and setting up a GUI with “set” methods
Import javax.swing.* for JFrame
Import java.awt.* for TextArea related stuff

Static variables belong to a class
Instance variables belong to an object (public, private, protected)
All classes inherit from class Object - methods must make sense for object being defined

LinkedLists are generally better than arrays because they are dynamic, not static. Allows for easier manipulation of data.
Nodes store data and pointer to next node

Abstract classes cannot be instantiated (Money)
Polymorphism, e.g. *Pet p = new Cat()*

- A cat is a pet, so p is a Cat, which inherits from class Pet
- Pet is the super class of Cat
- An array of Pet[] can store Cats, Dogs, and Fish

A JFrame contains a JMenuBar contains a JMenu contains a JMenuItem.
A JMenuItem is handled by a menu handler file which implements ActionListener. The actionPerformed method in ActionListener is called when the JMenuItem is clicked on

Exception handling with try/catch/finally blocks
Creating new exceptions

```
public class IllegalSSNException extends IllegalArgumentException {

      public IllegalSSNException(String message){
            super(message);
      }
}
```

Regular Expressions (REGEX) can be used to match strings using certain patterns of characters

Predefined Character Classes

.	Any character (may or may not match line end)
\d	A digit: [0-9]
\D	A non-digit: [^0-9]
\s	A whitespace character: [\t\n\x0B\f\r]
\S	A non-whitespace character: [^\s]
\w	A word character: [a-zA-Z_0-9]
\W	A non-word character: [^\w]

Character classes

match any character inside [ ]

[abc]	a, b, or c (simple class)
[^abc]	Any character except a, b, or c (negation)
[a-zA-Z]	a through z, or A through Z, inclusive (range)
[a-d[m-p]]	a through d, or m through p: [a-dm-p] (union)
[a-z&&[def]]	d, e, or f (intersection)
[a-z&&[^bc]]	a through z, except for b and c: [ad-z]
[a-z&&[^m-p]]	a through z, and not m through p: [a-lq-z]

Quantifiers

X?	X, once or not at all
X*	X, zero or more times
X+	X, one or more times
X{n}	X, exactly n times
X{n,}	X, at least n times
X{n,m}	X, at least n but not more than m times

The Java Collections Framework contains:

- ArrayList - better array
- LinkedList - similar to one we made
    - Both can .add(), .remove(), and .set() with or without a given index
- HashMap - O(c)
- TreeMap - Red-Black Tree O(log(n))

Generics allow for reusable code by passing in the data types during compilation/runtime
Interface Comparable

The Controller tells the Model; The Model tells the Viewer

- The Model is the data we want to work with
- The View is how the data looks
- The Controller updates the Model

Threads are a single instance of program execution

Three Thread States:

- Running - has control of CPU
- Ready - ready for a turn on the CPU
- Waiting - waiting for I/O or sleep()