COMS W1004 Introduction to Computer Science and Programming in Java

Course Notes

By Griffin Newbold

Version 1.0.2

Icon

Description automatically generated

Course Contents:

1. Introduction to the Course
2. Introduction to Algorithms

2.1 What is an Algorithm?

2.2 Examples of Algorithms in Java

2.2.1 Selection Sort

2.2.2 Insertion Sort

2.2.3 Linear Search

2.2.4 Binary Search

1. Setting up Java Files

3.1 Class Structure

3.2 Methods and The Main Method

3.3 Import Statements

3.4 Writing Hello World

3.5 Compiling and Executing Code

1. Java Building Blocks

4.1 Variables

4.2 Primitive Data Types

4.3 Objects

4.3.1 Constructors

4.4 The Math and String Class

4.5 Overloading Methods

1. Iteration and Conditionals

5.1 The While Loop

5.2 The For Loop

5.2.1 Traditional For Loop

5.2.2 Enhanced For Loop

5.3 If Statements

5.3.1 Else if Blocks

5.3.2 Else Blocks

5.4 The do-While loop

5.5 The Switch Case

5.6 The Ternary Operator

1. The Java Style Guide
2. Java Operators and Precedence
3. Introduction to Boolean Logic
4. Reading the Compiler
5. Type Casting
6. Class Design

11.1 Top-Down Design

11.2 Bottom-Up Design

1. The String Class Reprise
2. Introduction to Arrays
3. Using Arrays
4. The Mutable Array
5. Using ArrayLists
6. Autoboxing and Unboxing
7. Inheritance and Polymorphism

18.1 Extends Keyword

18.2 Super and Sub Class Relationship

18.3 Overriding Inherited Methods

18.4 Statics in Java

1. Interfaces in Java
2. Files and Exceptions
3. Generics in Java
4. Introduction to Algorithm Analysis and Big O
5. Introduction to Recursion
6. Linked Lists
7. Stacks and Queues
8. Beyond COMS W1004

26.1 Circuits and Logic Gates

26.2 Computer Networks

26.3 Turing Machines

1. Introduction to this Course

Welcome to COMS W1004 Introduction to Computer Science and Programming in Java! For some of you this is your first taste of Computer Science, for others you may have taken 1002 or have prior experience from other sources. My name is Griffin Newbold, I am a TA for Professor Cannon in the CS Department at Columbia. I took 1004 in the Fall of my freshman year and then began TAing for Cannon starting in the Spring. This course notes set should be good regardless of whether you are taking 1004 with Blaer or Cannon. The version of the course I took was organized much differently then what you are experiencing, after the 2021-2022 academic year the CS department is said to make some changes to the curriculum for 1004 and 3134, removing most of the non-programming concepts that were taught in the course. I will still include them here, but they will be at the very end of the course notes in the event the content is still necessary in the future. Currently there are two textbooks that are required for 1004: Invitation to Computer Science by Schneider and Gersting and Big Java: Early Objects by Cay Horstmann when we get to talking about Algorithm Analysis and More advanced Data Structures, I will be pulling from a third textbook that I will mention at that time. Whether you are a Columbia College Student or Columbia Engineering student it is very important to understand what Computer Science is and how it has impacted the world around us. Hopefully by the end of this set of course notes you will find yourself believing that to be the case. Best of luck to you as you continue onward. Let us begin learning about Algorithms

2. Introduction to Algorithms

2.1 What is An Algorithm?

In basic terms, an **algorithm** is a set of well-defined steps or rules that you need to follow to obtain a pre-determined result. Algorithms need to have input and produce an output in a reasonable finite amount of time. What does it mean to be well-defined? Well to put it simply, there should be no uncertainty as to what the next step should be. For example, when you read the directions on a bottle of shampoo, and it says to “Wet your hair, Lather, Rinse, and Repeat” what steps exactly are you repeating? As the bottle does not specifically say which steps to repeat, meaning the steps are not well defined and thus not a proper algorithm.

Timeline

Description automatically generated

Figure Steps for Baking a Cake

From Figure 1, we can see the instructions for how to bake a cake, this cake in fact not being a lie, there are well defined steps with inputs – in this case that would be our ingredients listed above – and we know the output is going to be a delicious cake. Upon reading the steps in the figure it is clear that this will be completed in a finite amount of time, meaning that figure 1 is an algorithm.

2.2 Examples of Algorithms in Java

In Java, you will need to implement several algorithms to achieve the desired result. Some of these algorithms you will be making from scratch throughout the problem sets and others are already well defined for you that you should know. We will go through these four introductory algorithms and detail anything about them that you need to know for 1004.

2.2.1 Selection Sort

There will be times throughout the course of your programming journey where you will need to potentially sort items in a collection. Selection sort is the simplest sorting algorithm that there is. The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning. The algorithm maintains two subarrays in a given array.

1) The subarray which is already sorted.  
2) Remaining subarray which is unsorted.

In every iteration of selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray.

A screen shot of a computer

Description automatically generated with low confidence

Figure Selection Sort Implementation

If the syntax looks confusing do not worry about it, it will be covered in time. All you need to know is that we have a collection of numbers of length n, and we start from the first element and for every subsequent element after it we check to see if we can perform a swap, if we can we swap and then we move on to the next element and perform the process again.

2.2.2 Insertion Sort

Another sorting algorithm to consider, is insertion sort. The way to think about insertion sort is like how you would imagine sorting playing cards that you are holding in your hands. It is easier to envision but arguably more difficult to program off the top of your head. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.

**Algorithm**   
To sort an array of size n in ascending order:

1: Iterate from arr[1] to arr[n] over the array.   
2: Compare the current element (key) to its predecessor.   
3: If the key element is smaller than its predecessor, compare it to the elements before. Move the greater elements one position up to make space for the swapped element.

Text

Description automatically generated with medium confidence

Figure Insertion Sort Implementation

Sorting is only one of the major operations algorithms can perform, the other is searching which we will see momentarily.

2.2.3 Linear Search

Sometimes you will need to search for a specific element within a collection of items. Assume you have a collection of items called arr that is of length n that you need to search for element x. The simplest approach to perform a linear search through the items for the index that holds element x, that is compare each element in arr until you either find x or go through all the elements, in the event you do not find the desired element, it is customary to return -1.

Text

Description automatically generated

Figure Linear Search Implementation

Linear Search is a very simple searching algorithm, however because it must make comparisons on every single element it can be a relatively slow algorithm in the event we are dealing with large collections of elements. Let’s dive into an algorithm that is much faster approach to searching for an element within a collection.

2.2.4 Binary Search

In order to achieve a faster algorithm, we will need to assume – and in application guarantee – that our collection of elements is sorted prior to performing the search. Binary Search starts from the middle element of the collection performs a comparison and then chops the collection in half, then repeats until the element is found or you perform the last comparison. Since we are disposing of half the collection per comparison, the algorithm runs in logarithmic time – we will come back to that concept later – for now just know that it is much faster on average than a linear search.

Text

Description automatically generated

Figure Implementation of Binary Search

It is important to note that with this algorithm, if there are duplicates there is no guarantee of which index will be returned. You may have also noticed that this algorithm calls itself, that type of algorithm is known as a recursive algorithm, and we will discuss those in more detail later. Finally, it is important to note the inclusion of a method signature rather than just the algorithm itself, methods are integral to Java programs as we will see shortly.

3 Setting Up Java Files

3.1 Java Class Structure

There are several components that make up a Java class. We will discuss each component later in this chapter but first here is a look at a fully implemented class in Java, do not worry about figuring out the logic but focus on the structure.

Text

Description automatically generated

Figure A Fully Implemented Java Class

There are a lot of components to look at here, but this is what essentially all Java files look like. The first thing to take note of is the first line. This is called a class declaration and contains the words “public class” and then the name of the class which is often in CamelCase starting with a capital letter. For now, do not worry about the words “extends Object” we will talk about it again when we talk about Inheritance and Polymorphism in chapter 18. Lines 3 and 4 are called instance variables, we will talk about those in Chapter 4.

3.2 Methods and The Main Method

One of the main aspects of a Java Class that you should know about is methods. Methods are chunks of code that allow us to perform whatever operation we desire them to do. Methods are designated using method signatures. The following is an example of a method that adds two numbers together.

Graphical user interface

Description automatically generated

Figure Java Method Adding Two Numbers

Going from left to right, the first part of a method is the access modifier, for purposes of 1004 it will either be public or private. For now, just assume it will be public as we will discuss use cases for private methods when we talk about class design in Chapter 11. The second keyword is the return type of the method. It can be any data type or void – meaning it returns nothing – next up we have the method name, which is written in camelCase. Finally, we have the parameter list which can either be empty () or filled with parameters like the example above. Now for a Java program to execute anything, your program must contain a main method. The main method is always defined like so:

A picture containing graphical user interface

Description automatically generated

Figure Main Method Implementation in Java

The Main Method is always defined as a public method, with the static modifier attached – do not worry about why the method is static right now we will circle back to this later – the return type of the main method is void because the main method does not return any value. The method is always named “main” and be careful because it is indeed case sensitive. Finally, the parameter list contains a single argument, a String array called “args” most often this isn’t taken full advantage of, but it allows for initial values to be passed into the program from the command line.

3.3 Import Statements

Sometimes in Java we will want to write code that already has been written. For any code that you would like to use that is not in the same directory as your current project then you will need to import the code you wish to use. Later we will discuss a data structure called an ArrayList which needs to be imported from a specific directory created by Java developers.

Text

Description automatically generated

Figure Three examples of Import Statements in Java

The first example is the import statement required to gain access to the functionality of the ArrayList data structure. This is the most specific import you can make for ***non-static*** imports. For purposes of 1004 you do not need to know static imports, but they do exist and may be useful in real world applications. The second example is an import statement that lets you have access to all functionalities held within the java.util package. The \*at the end is known as a wildcard and it essentially means to import all. So, in the event you forget a specific import like example 1, you can make an import statement like example 2 and you will be all good to go. The third example is one you will almost never write but will always be there. java.lang is the default java package that contains all the functionality of the core Java experience, that being said you will never need to import it yourself, making it almost always redundant.

3.4 Writing Hello World

You almost learned everything you need to write your first java program. The last two key things you will want to know is that to print things to the termina; you will need to use:

System.out.println(“What I want to Print”);

Notice at the end there is a semicolon. Almost all Java lines end in semicolons, the exceptions being when you are opening and closing code blocks with the curly brace. Now that you know about the structure of a Java file and how to write a simple print statement, you can finally write the famous “Hello World” program.

Text

Description automatically generated

Figure Java Program that Prints "Hello World" to the terminal

3.5 Compiling and Executing Code

Now that you have successfully written your first java program, it is time to compile and execute your code. To do this – assuming you are using codio – go to the navigation bar on the top of the page, click on tools then terminal. Once the terminal window opens you will type the following command to compile your code:

javac FileName.java

Once you do that, one of two things will happen either you will get a result like in Figure 11 or like in Figure 12. If you get a result like in Figure 11, congratulations your code has successfully compiled, and you may proceed to execution. If you got a result like in Figure 12, then you have received compiler errors that you will need to fix prior to executing your code. Most often you can just read the output provided and do what the compiler tells you to do. We will talk more about reading the output the compiler gives you in Chapter 9.

Text

Description automatically generated

Figure Screenshot of Codio Terminal with Successfully compiled code for a file with the name "Card.java"

Text

Description automatically generated

Figure Screenshot of Codio Terminal with unsuccessfully compiled code for a file with the name "Card.java" that has a single compiler error on line 13

Compiling your code gives you a .class file of the same name, we will talk more about the differences between .class and .java files in the next section but for now let’s execute our code. To execute a java file, you will need to write the following command in the terminal:

java FileName

After you enter this command and press enter, anything in the main method of the file will be executed, doing this on our Hello World program should yield the following output.

Text

Description automatically generated

Figure Screenshot of the Codio Terminal showing the successful compilation and execution of MyFirstProgram.java

While it is fundamentally important that you know how to manually compile and execute your code. There is a chance your professor has enabled the “Compile and Run” button on the navigation bar on the top of the window. If this is the case, I recommend using that once you know you can remember how to manually compile and execute your code.

4 Java Building Blocks

To understand programming in Java, you must understand the basic building  
blocks that make up the language itself. The first thing that needs to be  
understood is the concept and application of variables.

4.1 Variables

Variables are how you store data in Java, when dealing with primitive data types – which will be discussed shortly – variables directly hold the data that they are pointing to. When dealing with non-primitive object types, variables hold references to the data they are pointing to. Variables are often referred to as fields, so do not get confused when you hear the term field.

There are two distinct ways to declare and assign values to variables in Java and the methodology is dependent on the type of the variable, whether they are primitives or objects.

For primitive types it is as follows:

**datatype variableName = valueOfExpressionOfSameType;**

For object types it is as follows:

**ObjectType variableName = valueOfExpressionOfSameType;**

Later on we will see concrete examples of the preceding two statements.

4.2 Primitive Data Types

In Java when you are dealing with data, there are different forms that the data can take, and these forms determine what type of data can be stored. While there are many primitive data types, programmers typically use only a handful of them. Since it is required of you to know all the primitive data types in Java, below is a list of the types and the specific attributes about them that may be worth jotting down:

1. byte: stores 8 bits of data and can hold integer values ranging from -128 to  
   127
2. char: stores 16 bits of data and can hold single characters ex: ‘a’ must use  
   single quotes
3. boolean:  stores 8 bits of data and is used to hold one of two values true or  
   false
4. short: stores 16 bits of data and can hold integer values ranging from                   -32768 to 32767
5. int: stores 32 bits of data and can hold integer values ranging from                         -2147483648 to 2147483647
6. float:  stores 32 bits of data and can hold floating-point values
7. double: stores 64 bits of data and can hold floating-point values
8. long:  stores 64 bits of data and can hold integer values -9.22E+18 to  
   9.22E+18

The following is an example of how to declare a variable named myFirstInt as type int and we will assign it the value 1004:

**int myFirstInt = 1004;**

We can then reassign myFirstInt to have a different value the following way:

**myFirstInt = 3134;**

Notice that we don’t have to put the dataType when we are reassigning the variable to a different value, this is because the variable already exists. You only need to put the dataType when you first declare the variable. To illustrate this the following code segment are identical in terms of functionality:

int mySecondInt; //declaring the variable  
mySecondInt = 2021; //assigning a value to the variable  
int mySecondInt = 2021; //declaring and assignment

4.3 Objects

4.3.1 Constructors

4.4 The Math and String Class

4.5 Overloading Methods

5. Iteration and Conditionals

When programming, you are going to want to repeat certain segments of code or run code only when a certain condition is met. That is where the idea of Iteration and Conditionals come into play. I will first discuss structures used for repeating commands but then I will transition into talking about structures used for conditionals.

5.1 The While Loop

The While Loop is a very simple structure with a simple premise:

while(someConditionIsTrue){

//execute code

        }

The above is exactly how you construct a while loop too, inside the parentheses is a boolean which is either true or false, if the boolean is true the loop will execute otherwise it will be skipped, if you want the reverse to occur use the ‘!’ operator in front of the boolean condition.  An important part of the while loop to remember is to be making progress within the loop so you will eventually exit the loop, otherwise you will create an infinite loop and cause yourself a headache trying to resolve it.

5.2 The For Loop

The For Loop allows you to repeat an operation x times. There are two key types in Java, the traditional for loop and the enhanced for loop and we will cover each individually.

5.2.1 Traditional For Loop

The traditional for loop is what you will most often use to repeat code segments and if you want to loop through indexed collections and use the index at some point while doing so. The traditional for loop looks like the following:

for(int i = 0; i < someValue; i++){

//code to run

}

The first thing you will need to do is initialize the counter variable, it is the first third of the for loop statement “int i = 0;” Secondly you’ll need the terminating condition, this tells the loop when it should stop executing the code block inside “i < someValue” finally you’ll need the incrementing condition which tells the loop how much to increment or decrement the counter variable by “i++” this simply means increase i by 1.

5.2.2 Enhanced For Loop

The other type of for loop is the enhanced for loop. This is used for loop through collections of items seamlessly. The look as follows:

for(someCollectionType x : someCollection){

//code to run

}

You will use these less frequently, but they may be helpful when dealing with collections of items.

5.3 If Statements

The main conditional structure for this course if the if statement:

if(someCondition){

//code to run

}

That is exactly how an if statement looks if you only have one line where the “code to run” is then you actually do not need the curly braces.

5.3.1 Else if Blocks

If you wanted to add another condition to check then you will need to use an “else if” block and it must be attached to an if statement as follows:

if(someCondition){

//code to run

}else if(someOtherCondition){

//some other code to run

}

This allows for multiple separate conditions to be checked within an if statement.

5.3.2 Else Blocks

If you want some default action to occur in an if statement, the best way to achieve this is with an else block which must be attached to either the end of an if block or an else if block but cannot come between an if and else if.

if(someCondition){

//code to run

}else if(someOtherCondition){

//some other code to run

}else{

//default code to run

}

 6. The Java Style Guide

Now that you have a bit of Java under your belt, and you can write simple programs it is time to discuss another important part of programming: Style.

It is good to get into the practice of writing code that is neat and concise. In the files section there is a style guide present that you can refer to. I will cover the contents of the style guide here.

The first principle deals with the essentials of your source file. 1.1 is a trivial concept but your filename should be a copy of the class name with the .java extension attached. 1.2 is not trivial in terms of the usual grading for programming assignments, usually points will be deducted in the event you forget to include 1.2 which states that you should include your name, UNI, filename, and a succinct description of the file. The following is an example of a proper file header as expected from the style guide.

[insert example of file header]

I will skip over 1.3 as it really is directed towards people with past CS experience and to be fair, I do not believe it is beneficial to mention terms that aren’t directly necessary for this course’s applications.

Section 2 deals with the meat of the style guide which is formatting. Improper formatting can make your code unreadable and may prevent you from locating the source of errors when it comes time for debugging your code. 2.1 talks about the style for braces which is either K&R or Horstmann, I will show examples for each below:

[insert example of brace style]

2.2 just states that you should use braces wherever optional to decrease the number of potential errors to be caused by not doing as such. 2.3 mentions indentations which states you can either indent using tab, 2 spaces or 4 spaces. For convenience just click tab to indent your code. 2.4 talks about line length, which should never be over 80 characters (obviously if you have 82 no one is going to dock you points for it) in terms of method length, this is dependent on the assignment but averages to about 25 lines per method. 2.5 is about whitespace, simply put: between each logical block of code add a space to improve readability for the user and yourself.

Section 3 deals with all things naming. 3.1 ClassNames shall be written in UpperCamelCase as it was just demonstrated. Variables shall be written in lowerCamelCase as it goes with methods as well. 3.2 Constants shall be ALL\_CAPITAL\_LETTERS with underscores in between each part of the name. You should also give your variables and methods meaningful names which give a clue to their purpose in the program. 3.4 In the event you see yourself reusing variables of the same value and purpose, avoid hardcoding any potential changes by turning that variable into a constant, a popular example is gravity.

7.  Java Operators and Precedence

In Java there are lots of operator types, there are arithmetic operators, logical operators, and bitwise operators. Just like you learn in elementary school, these operators must be executed in the proper order to achieve the desired result. I will skip over the bitwise operators as they are not needed for 1004 but in the future, you may wish to revisit them as they can be useful for bit manipulation problems. The precedence of Java operators from highest to lowest is as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| Precedence | Operator | Type | Associativity |
| 11 | () [] . | parentheses, array subscript, member selection | Left to Right |
| 10 | ++ -- | Unary post increment/decrement | Right to Left |
| 9 | ++ --  + -  ! (type) | Unary pre-increment  Unary pre-decrement  Unary plus, Unary minus  Unary logical negation  Unary type cast | Right to Left |
| 8 | \*, /, % | multiplication, division, modulus | Left to Right |
| 7 | * - | addition and subtraction | Left to Right |
| 6 | <  <=  >  >=  instanceof | Relational less than  Relational less than or equal  Relational greater than  Relational greater than or equal  Type comparison (objects only) | Left to Right |
| 5 | == != | Relational is equal to  Relational is not equal to | Left to Right |
| 4 | && | Logical AND | Left to Right |
| 3 | || | Logical OR | Left to Right |
| 2 | ?: | Ternary Operator | Right to Left |
| 1 | =  +=  -=  \*=  /=  %= | Assignment  Addition assignment  Subtraction assignment  Multiplication assignment  Division assignment  Modulus assignment | Right to Left |

Associativity is simply the order in which you would execute operations of the same precedence.

8.  Introduction to Boolean Logic

Booleans are an integral part of any programming language. They are the building blocks of many control structures described earlier. In Java we have the && operator and the || operator. They mean AND and OR, for booleans with && within them to evaluate to true, both conditions must be true, while with the || only one needs to be true.

Let’s go through a couple of examples using the && operator and the || operator.

Assume I have two booleans, “isDay” and “isSunny” and I want to go outside if and only if it is both daytime and sunny outside, we can set a new boolean called “canGoOutside” equal to the following boolean:

canGoOutside = isDay && isSunny;

//only equates to true if both conditions are true

The && makes sure to enforce the fact that both need to be true, in Java as well if the first condition is false it won’t bother to evaluate the other condition as if the first condition is false, then there is no chance for the expression to evaluate to anything other than false.

With the same assumption as before, let's change the condition for going outside to be if it is daytime or if it is sunny - in this case sunny simply means no storms. Our boolean “canGoOutside” can now equal the following boolean:

canGoOutside = isDay || isSunny;

//equates to true if either condition is true

The || makes sure to enforce the fact that only one condition needs to be true, just like with the && operator, if the first condition evaluates to true then it won’t bother to evaluate the other condition since only one condition needs to be true.

9. Reading the Compiler

10. Type Casting

11.  Class Design

11.1 Top-Down Design

11.2 Bottom-Up Design

12. The String Class Reprise

13.  Introduction to Arrays

14.  Using Arrays

15. The Mutable Array

16. Using ArrayLists

17 Autoboxing and Unboxing

18  Inheritance and Polymorphism

18.1 Extends Keyword

18.2 Super and Sub Class Relationship

18.3 Overriding Inherited Methods

18.4 Statics in Java

19 Interfaces in Java

20 Files and Exceptions

21 Introduction to Algorithm Analysis and Big O

When it comes to designing Algorithms, there are multiple things that you should consider. The first is how long it takes your algorithm to run and finish, this is known as Time Complexity. The second is how much space your algorithm takes up, this is known as Space Complexity. The third and most important, does your algorithm do what it needs to do at all? When it comes to designing algorithms, getting a solution that works is the first step. Afterwards you can review design etiquette and derive a solution that is more optimal. Just like in Math there is a certain notation used to describe the space and time complexities of an algorithm. The most common is Big O notation, there are notations used such as Big Theta notation and Big Omega notation but for the purposes of 1004 you only need to concern yourself with Big O notation and particularly when it comes to Time Complexity, but you should also note the Space Complexity of any algorithm you write as well. Big O is an approximation of an algorithm’s complexity. Every action is a unit, so if you had an array of n items and for each item you performed an action then the complexity would be O(n). If for the same array of n items you performed n actions per item then the complexity would be O(n^2). Typically, the expression within the parentheses is a polynomial expression, you will learn more about the significance of that in 3134. For 1004 purposes, that is what you should know.

22 Introduction to Recursion

There is this neat little easter egg on Google if you type in “recursion” into the search bar and click enter, it will prompt you with “did you mean: recursion” and this is a loop that goes on forever if you keep clicking on it. That gives a hint of what recursion is but we’ll need to go deeper for the purposes of 1004.

Recursion is when a function or method calls itself within its scope, this is a useful and almost trivial programming technique used when dealing with data structures like linked lists and especially trees. When it comes to recursive methods there are a few key things that you should be aware of:

1. Always include a base case

* Base cases allow for you to exit out of the method, if you do not provide a base case, your program will crash with a Stack Overflow Error

1. Always make progress towards that base case

* See bullet above for reasoning.

1. Do not repeat work!

* Repeating work can increase time complexity and runtime for a recursive method, for small scale recursion exercises this is not necessary to consider but in the real world you will have to properly deal with it
* If interested in learning more, the process is called “dynamic programming” feel free to Google it to learn more!

23 Linked Lists

After Arrays and ArrayLists, typically Linked Lists enter the chat when talking about Data Structures. To put it simply, a linked list is a list of objects that contain links to the next object in the list (singly linked lists) and possibly to the object before it in the list (doubly linked lists). When you concern yourself with data structures, you must talk about their complexities when it comes to performing certain tasks, specifically: accessing an element, searching for an element, insertion, and deletion. For both singly and doubly linked lists the time complexities for these operations are, in the worst case:

1. Insertion:  O(1) - constant time
2. Deletion:  O(1) - constant time
3. Accessing: O(n) - linear time
4. Searching: O(n) - linear time

See if you can correctly identify the time complexities for these operations for an Array check this website for the answer once you take a minute to think about it ([Big-O Algorithm Complexity](https://www.bigocheatsheet.com/)) Linked Lists are an appropriate data structure to use when you want to make insertions and deletions from either end really efficiently and are often used as the backbone for other data structures like Stacks and Queues.

24 Stacks and Queues

To wrap up the course notes - because in my earnest opinion I do not see a proper scenario where you can get past Stacks and Queues and have a solid foundation of the rest of the material that has been previously discussed - we will talk about Stacks and Queues.

To start, Stacks and Queues are similar data structures, a Stack is a LIFO (Last In First Out) and a Queue is a FIFO (First In First Out), but they share the same methods and time complexities.

1. Insertion:  O(1) - constant time
2. Deletion:  O(1) - constant time
3. Accessing: O(n) - linear time
4. Searching: O(n) - linear time

For a Stack the relevant methods are: push(), pop(), and peek(); All push does is put the object you pass in as an argument onto the Stack. Pop removes the latest item added to the stack. Peek returns the latest item added to the stack but does not remove it from the stack. The prime applications of a stack include:

1. evaluation of postfix expressions
2. executing calls to methods
3. checking to see if a string is “balanced”

For those who are interested in majoring in CS and are already familiar with leetcode here is an easy stack problem [Valid Parentheses - LeetCode](https://leetcode.com/problems/valid-parentheses/)

For Queues it is a very similar story the relevant methods are enqueue(), dequeue(), and peek(): Enqueue is the analog to push , dequeue is the analog to pop and peek is the exact same between stacks and queues.  You can use queues to keep track of items in a particular order, any music app or video streaming service often uses queues to keep track of what the next song or video is.