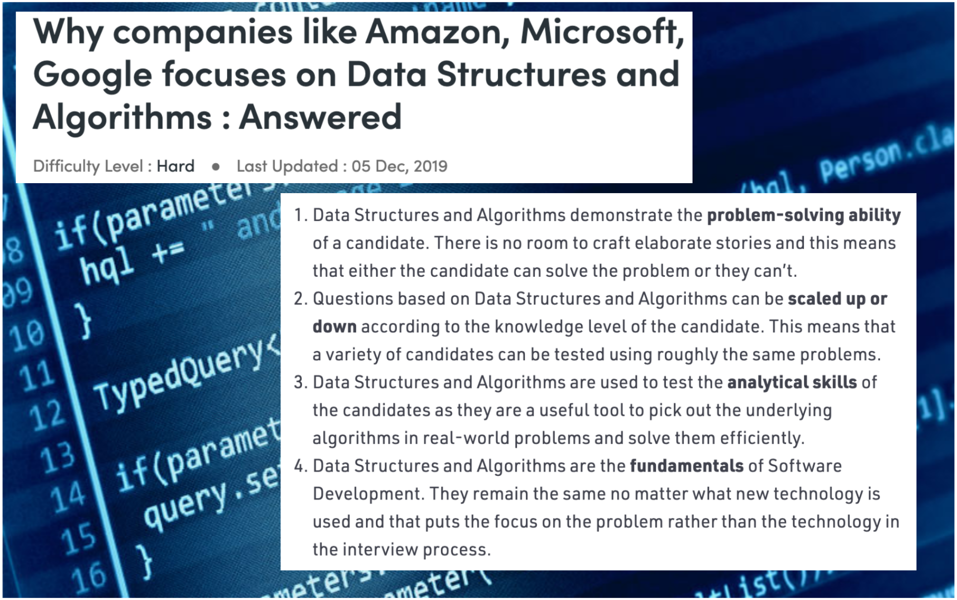
Data Structures

Augustin Cosse.

Spring 2021

January 29, 2021



Why companies like Amazon, Microsoft, Google focuses on Data Structures and Algorithms : Answered

Difficulty Level : **Hard** • Last Updated : 05 Dec, 2019

1. Data Structures and Algorithms demonstrate the **problem-solving ability** of a candidate. There is no room to craft elaborate stories and this means that either the candidate can solve the problem or they can't.
2. Questions based on Data Structures and Algorithms can be **scaled up or down** according to the knowledge level of the candidate. This means that a variety of candidates can be tested using roughly the same problems.
3. Data Structures and Algorithms are used to test the **analytical skills** of the candidates as they are a useful tool to pick out the underlying algorithms in real-world problems and solve them efficiently.
4. Data Structures and Algorithms are the **fundamentals** of Software Development. They remain the same no matter what new technology is used and that puts the focus on the problem rather than the technology in the interview process.

HACKERNOON

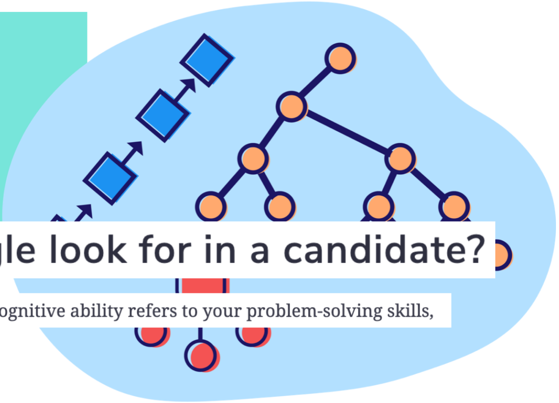
How I got a Job at Facebook as a Machine Learning Engineer

1. Telephonic Interview:

This was a very basic Data Structure interview and sort of a basic sanity check. I guess FB just wants to give you some more time to prepare for the coming rounds and also see whether it would be worth to call you for the onsite rounds. For me, this interview lasted 45 minutes on a Video Call. The interviewer started by telling me about his profile at Facebook, and in turn asking about my profile for the first 10 minutes or so.

I started my preparation by creating a list of topics I would need to prepare. You could prepare even more topics but these are the bare minimum for these interviews.

Data Structures: Array, Sets, Stack/Queue, Hashmap/Dictionary, Tree/Binary Tree, Heap, Graphs.
Algorithms: Divide-and-Conquer, DP/memoization, Recursion, Binary Search, BFS/DFS, Tree traversals.



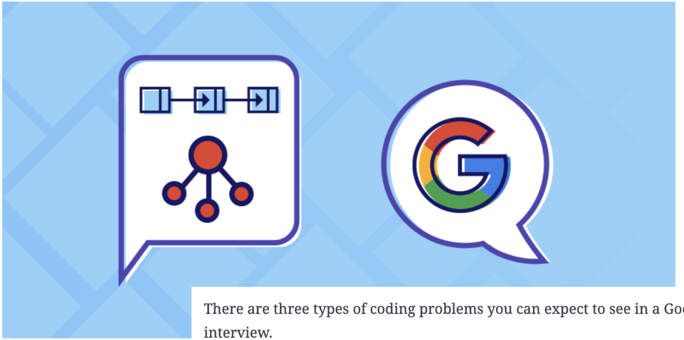
What does Google look for in a candidate?

Cognitive ability. General cognitive ability refers to your problem-solving skills,

Technical skills. Google hires candidates with the strongest coding abilities, and they assess technical skills mostly on *conceptual* understanding, not memorization. They assess coding skills on the following topics:

- Algorithms
- Sorting
- Data structures
- Graphs

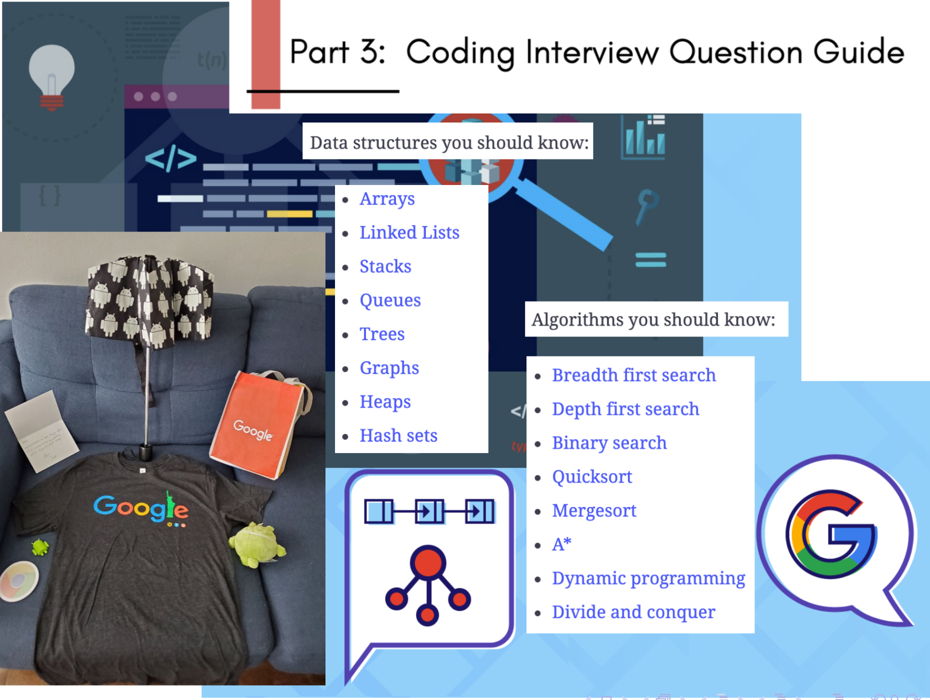
Cracking the Google Coding Interview: the definitive prep guide



There are three types of coding problems you can expect to see in a Google interview.

- **System design questions:** these questions gauge your ability to handle high-level system design with scalability in mind.
- **Coding interview challenges:** these questions gauge your knowledge of data structures and algorithms to optimize a solution to common problems.
- **General analysis questions:** these questions gauge your thought process through mathematical or opinion-based questions

Part 3: Coding Interview Question Guide



Data structures you should know:

- Arrays
- Linked Lists
- Stacks
- Queues
- Trees
- Graphs
- Heaps
- Hash sets

Algorithms you should know:

- Breadth first search
- Depth first search
- Binary search
- Quicksort
- Mergesort
- A*
- Dynamic programming
- Divide and conquer

LIVING GOLD
Microbes made Earth's biggest lode

NewScientist

WEEKLY February 7-13, 2015

EXPOSED:
THE ALGORITHMS THAT RUN YOUR

Music
Shopping
News
Airfares
Internet searches
Life
Friends
Family
Job
Finances
Human rights

SPECIAL REPORT
NO PLACE LIKE HOME
What your house gets up to when you're away

COSMIC DEFLATION
Big bang breakthrough bites the dust

FOWL PLAY
History's most persecuted bird

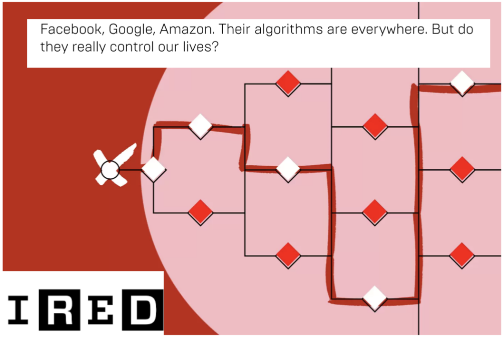
SAVE THE GIRLS
Fighting high-tech sex selection

No3007 US/CAN$5.95

Science and technology news www.newscientist.com US jobs in science

What happened when I let algorithms run my life for a week

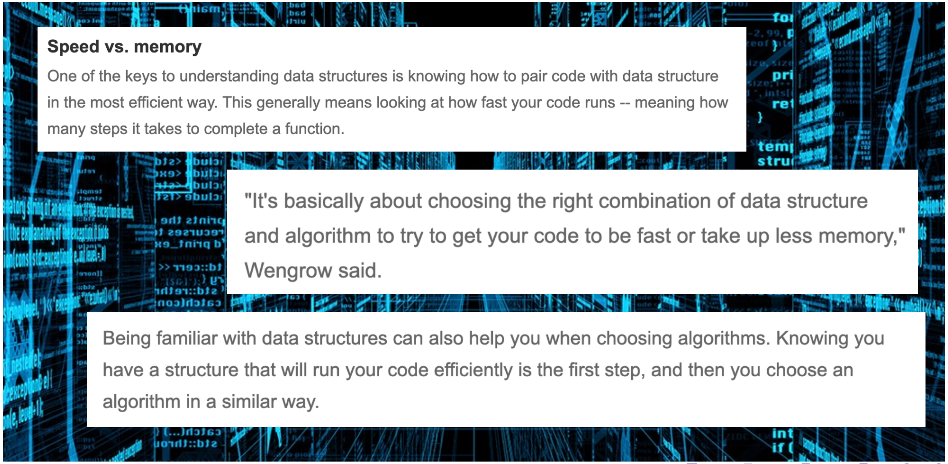
Facebook, Google, Amazon. Their algorithms are everywhere. But do they really control our lives?



WIRED

TechTarget SearchData Management

Why understanding data structures is so important to coders

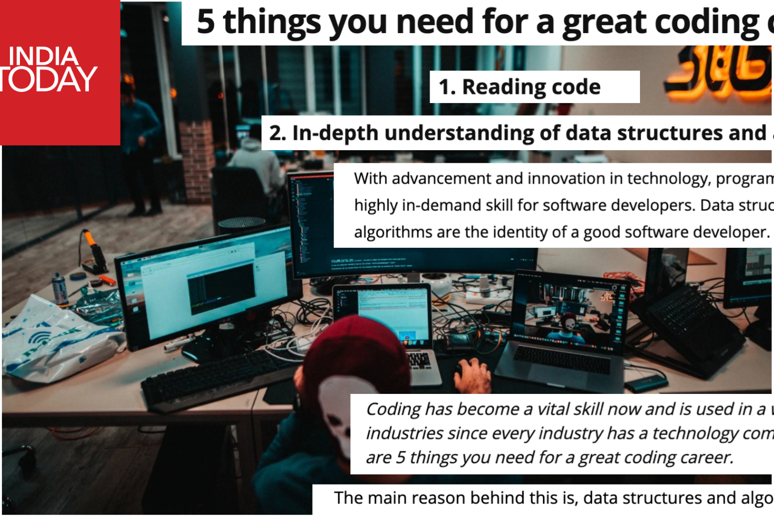


Speed vs. memory

One of the keys to understanding data structures is knowing how to pair code with data structure in the most efficient way. This generally means looking at how fast your code runs -- meaning how many steps it takes to complete a function.

"It's basically about choosing the right combination of data structure and algorithm to try to get your code to be fast or take up less memory," Wengrow said.

Being familiar with data structures can also help you when choosing algorithms. Knowing you have a structure that will run your code efficiently is the first step, and then you choose an algorithm in a similar way.



5 things you need for a great coding career

1. Reading code

2. In-depth understanding of data structures and algorithms

With advancement and innovation in technology, programming is becoming a highly in-demand skill for software developers. Data structures and algorithms are the identity of a good software developer.

Coding has become a vital skill now and is used in a wide range of industries since every industry has a technology component now. Here are 5 things you need for a great coding career.

The main reason behind this is, data structures and algorithms improve the problem-solving ability of a candidate to a great extent.

The interviews for technical roles in some of the tech giants like Google, Facebook, Amazon, Flipkart is more focused on measuring the knowledge of data structures and algorithms of the candidates as they want people who can think out of the box to design algorithms that can save the company thousands of dollars.

NO ONE'S DRIVING

The Modern World Has Finally Become Too Complex for Any of Us to Understand

Vast systems, from automated supply chains to high-frequency trading, now undergird our daily lives — and we're losing control of all of them

Tim Maughan Nov 30, 2020 · 10 min read ★

Schedule

- Lectures: Tuesday/Thursday, 12:30pm-1:45pm.
- Recitation (Mandatory): Tuesday, 2.15pm 3.45pm.
- Office hour : Thursday 2.15pm 3.45pm.
- Location: NYU Paris, 57 Boulevard Saint-Germain, Room 410
- Combination between programming sessions (Java) and lectures
- Exams: Midterm: 30%, Final : 30%
- Assignements throughout the semester: 30%
- Independent project: 10%

Course organization

- Notes + Sample exam questions will be posted (soon) on the course webpage
- Sample exam questions = help you with the study but not comprehensive
- If a section of the notes is not covered in class, you don't have to study it for the exam
- See http://www.augustincosse.com/teaching for details (select "Data Structures > Spring 2021" semester)

General Organization (I): Website + Project

- All the material will be posted on the website (including labs, slides and coursenotes)
- Current Password : Zion101
- One end of semester project: Implementation and/or reading of pioneering paper (see course website for some ideas)

Data Structures and Algorithms (NYU Paris, Spring 2021)

The Github page for the class will be hosted at https://github.com/acosse/DataStructures2021 and will be used for the lab and the assignments. You can also click on each "Lab" in the schedule below and this will re-direct you to the github page.

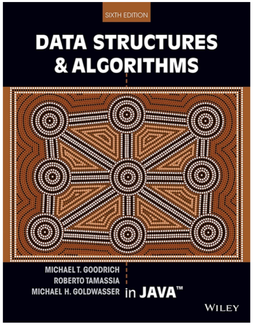
Tentative schedule:

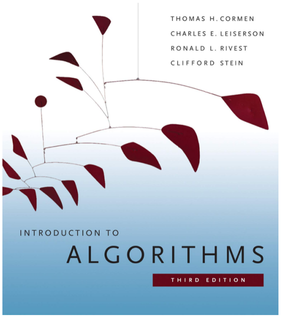
Legend: Lab sessions are in green, Homeworks are in red (right side of the table), dates related to the project are in orange.

Week #	date	Topic	Assignements
Week 1	01/26, 01/28	Reminders/Intro to Java, Elementary programming including loops, strings, arrays, objects, classes,... Lab 1 (Setting up Java, git,..)	
Week 2	02/02, 02/04	Object Oriented Programming and Design, Part I, Lab 2	Assign. 1
Week 3	02/09, 02/11	Object Oriented Programming and Design Part II including inheritance and Abstract classes, Lab 3	Assign. 1 due, Assignment 2

General Organization (I): Website + Project

- Main references for the course:
  - *Data structures and algorithms in Java*, Goodrich, Tamassia, Goldwasser
  - *Introduction to Algorithms*, Cormen, Leiserson, Rivest, Stein.


SIXTH EDITION
DATA STRUCTURES & ALGORITHMS
MICHAEL T. GOODRICH
ROBERTO TAMASSIA
MICHAEL H. GOLDWASSER
in JAVA™
WILEY


THOMAS H. CORMEN
CHARLES E. LEISERSON
RONALD L. RIVEST
CLIFFORD STEIN
INTRODUCTION TO
ALGORITHMS
THIRD EDITION

General Organization (II): Reference Books

- If you need additional references
  - *Introduction to Java Programming (and data structures) 10th edition (or more recent), Comprehensive version*, Yang.
  - *A Java Reference: Assorted Java Reference Material*, Hilfinger
  - *Algorithms*, Sedgewick and Wayne,
  - *Java Software solutions, foundations of program design*, Lewis and Loftus

This week

- General Introduction to Java
- No recitation $\Rightarrow$ replaced by setting up Java, Git, Gradescope,..

Intro to java

- The first step in coding is to write a program made of a series of statements whose syntax obeys well defined rules
- In order to obtain the results of the program, we will then want the computer to read this program
- For the computer to be able to read the program, we need to translate our program into a language that the computer can understand.
- Because of the electronic components, the information in a computer is transmitted through combinations of two states 0 (turned off) and 1 (switched on), also known as a sequence of bits

Intro to java

- Since the computer can only understand those two states, every type of information (even something as simple as a number) has to be encoded, through an appropriate scheme, as a sequence of 0 and 1
- In order words, when interacting with the computer, we need a specific step that will process every type of information and encode the information as a sequence of bits.
- Note that we will also need a step that will do the reverse, that is to say that will take the result of the execution (in bits) and turn it into the appropriate information.

Intro to java

- The computer is very good at executing tasks. In fact it can execute his tasks much faster than a human being
- A computer is however not intelligent. It cannot choose between two actions and because of its limited capabilities, it can only understand a task if it is given to him as a binary sequence.
- When designing programming language, the general idea is to encode the orders that will be submitted to the computer by using particular words and a particular syntax (i.e how the words can be combined with each other)
- C, C++, Pascal, Basic, Fortran, Cobol, Java, Python are all programming languages consisting of a collection of words and a syntax that is specific to each language

Intro to java

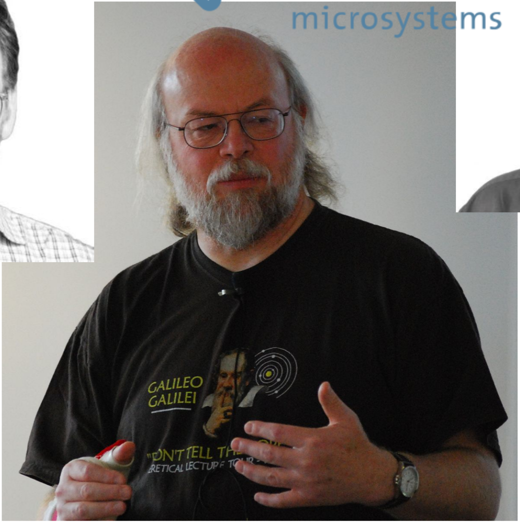
- To get the computer to execute a sequence of tasks, it is therefore necessary to know one of those languages
- The translation of the source code (i.e. the higher level code written in your favorite programming language) into the executable (lower level) machine language is done by a program known as the compiler
- Compiling consists in running a program that reads every instruction from the (higher level) source code, check if those are following the syntax of the language and then, provided there is no mistake, the compiler produces a binary code that can be executed by the computer

Intro to java

- A program written in Pascal is translated in machine language through a Pascal compiler. Similarly a program written in Java is translated in machine language through a Java compiler
- However, the binary code (machine language) that can be understood by each machine changes from one machine to another.
- The binary code associated with each operation (e.g. addition) is determined by the designer of each computer
- It is therefore impossible to run the same source code on two different machines without having to compile it twice

- The idea of Gosling was that it should be possible to execute a program written on a PC (IBM), just as on Mac (Apple) or on any Unix station (Sun type) without having the re-write or re-compile the source code, I.e introduce a language that would be independent of the computer.
- In order to do this, Gosling decided to create an intermediate code between the source code and the binary code
- This intermediate code is known as pseudo code or byte code

Patrick Naughton



Mike Sheridan

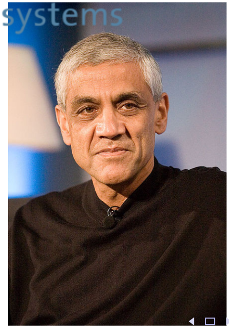
James Gosling

JAVA™ Java™ Java™ Java™

William Nelson Joy

Andy Bechtolsheim

Scott McNealy



Vinod Khosla

Ed Oates, Bruce Scott, Bob Miner

ORACLE®

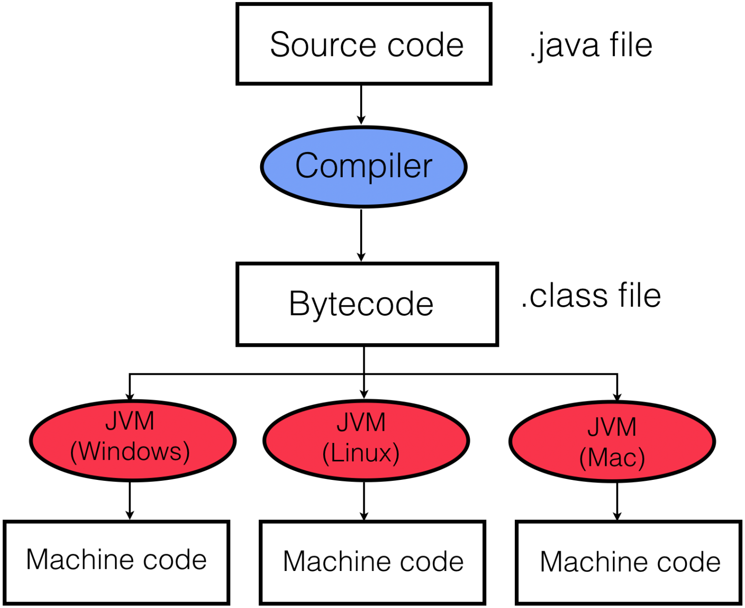
Larry Ellison.

Intro to java

- The byte code can then be generated once and used on every machine without having to re-compile it.
- Generating the machine code from the byte code is then done by a specific program which varies from one machine to the next and is known as Java interpreter (there are thus as many interpreters as there are computers)
- The collection of all interpreters makes up what is known as the Java Virtual Machine (JVM)
- Unlike other compilers such as C or C++, the Java compiler hence does not create binary code but produces the bytecode which is then interpreted by the JVM

Intro to java

- The advantage of this is that the developper is guaranteed that the program that he/she will design is perfectly compatible with any of the available computers


Source code
.java file
Compiler
Bytecode
.class file
JVM (Windows)
JVM (Linux)
JVM (Mac)
Machine code
Machine code
Machine code

Intro to java

- The first Java compiler was designed by Gosling for Sun around the 90's
- Today the compiler is provided as part of the Java Development Kit (JDK) or Standard Development Kit (SDK) which is available for Solaris, PC, Mac or Linux.
- The JDK is a toolbox designed for the development of applications which contains multiple tools/programs which can be executed as particular commands
- Two of the most important ones are the compiling (`javac`) and the execution (`java`) commands

Intro to java

- So what are examples of Java applications?

Java magazine

The 25 greatest Java apps ever written

From space exploration to genomics, from reverse compilers to robotic controllers, Java is at the heart of today's world. Here are a few of the countless Java apps that stand out from the crowd.

by Alexa Morales

22. Integrated Genome Browser.

Integrated Genome Browser

Visualization for genome-scale data

1. Maestro Mars Rover controller.

NASA SOFTWARE

21. NSA Ghidra.

MINECRAFT

10. Minecraft.

3. NASA WorldWind.

NASA WorldWind

elasticsearch

6. Wikipedia Search.

Lucene

Data Types

- Most commonly used data types in Java:

boolean	true or false
char	16 bit Unicode character
byte	8 bit signed two's complement integer
short	16 bit signed two's complement integer
int	32 bit signed two's complement integer
long	64 bit signed two's complement integer
float	32-bit floating point number (IEEE 754-1985)
double	64 bit floating point number (IEEE 754-1985)

- The two's complement of an N-bit number is defined as its complement with respect to 2^N. I.e. the sum of a number and its two complement is 2^N. For example 010 + 110 = 100 = 8. So the two complement of 010 is 110

- Two's complement is the most common method of representing signed integers on computers. If the binary number 010 encodes the signed integer 2 then its complement 110 encodes the inverse -2

- The IEEE 754-1985 standard consists of a sign bit (+1/-1), a sequence of 8 bits encoding the exponent and a sequence of 23 bits encoding the mantissa.

- As an example, consider the number 85.125 For this number

	85	= 1010101
	0.125	= 001
we have	85.125	= 1010101.001
		$= 1.010101001 \times 2^6$
	sign	= 0

- In single precision the biased exponent is obtained by adding 127 to the exponent
  biased exponent 127+6=133
  133 = 10000101
  Normalised mantisa = 010101001
- We then add 0's to complete the 23 bits
- The IEEE 754 Single precision representation of 85.125 is then given by

0 10000101 01010100100000000000000

Intro to java

- In Java the name of a class, method or variable is called an identifier which can be any string of characters as long as it begins with a letter and consists of letters, numbers and underscore characters (besides a couple of reserved words)

Intro to java

- In addition to executable statements and declarations, just as any other language, Java allows programmers to embed comments (i.e annotations that are not processed by the compiler) in the script.

- There are two types of comments: inline comments (ignoring everything subsequently on the line) and multiline comments (which open with /* and close with */)

```
// This is an inline comment
/*
* This is a block comment
*/
```

Intro to java

- A variable having any of the types **char, int, double,..** simply stores a value of that type
- Variables can be declared as follows

```
boolean flag = true;
boolean verbose, debug;
char grade = 'A';
int i, j, k = 257;
double e = 2.71, a = 6.22e23
```

Intro to java

- In more complex java programs the primary actors are objets
- Every object is an instance of a class which serves as the type of the object
- The critical members of a class in java are the following:
  - Instance variables (also called fields) which represent the data associated with an object of a class. Instance variables must have a type (base type such as int, float, double, or a class type)
  - Methods = blocks of code that can be called to perform actions (similar to functions and procedures). Methods can accept parameters as argument and their behavior depends on the object upon which they are called

Intro to java

- A Method that returns information to the caller without changing any instance variable is called an accessor method
- An Update method is a method that may change one or more instance variables when called.

Intro to java

- Consider the class below.

```java
public class Counter {
        private int count;
        public Counter() {}
        public Counter(int initial) { count = initial; }
        public int getCount() {return count; }
        public void increment() {count++; }
        public void increment(int delta) {count = 0; }
        public void reset() {count = 0; }
```

- This class contains one instance variable named 'count'
- It also contains two special methods known as constructors
- Finally it contains one accessor method and three update methods

Intro to java

- Here is another example of a class

```
public class CounterDemo {
        public static void main(String[] args){
                Counter c;
                c = new Counter();
                c.increment();
                c.increment(3);
                int temp = c.getCount();
                c.reset(); // value becomes 0
                Counter d = new Counter(5);
                d.increment();
                Counter e = d;
                temp = e.getCount();
                e.increment(2);
        }
}
```

Base type vs Reference Type

- There is an important distinction in java between base type variables and reference type variables.
- Classes in java are known as reference types and variables of that type are called reference variables
- A reference variable is capable of storing the location (memory address) of an object of the declared class
- In particular, we might use it to reference an existing instance or a newly created one
- A reference variable can also store the special value **null** that represents the lack of an object

Base type vs Reference Type

- From the class CounterDemo, we also see that in java, a new object is created by using the new operator followed by a call to a constructor to the desired class

```
c = new Counter();
Counter d = new Counter(5);
```

- A constructor is a method that always shares the same name as its corresponding class
- The **new** operator returns a reference to the newly created instance and the returned instance is then typically assigned to a variable for further use

Constructors

- A Constructor that takes no argument, e.g. **Counter()** is known as a default constructor
- When creating a new instance of a class, three events occur:
  - A new object is dynamically allocated in memory with all instance variables initialized with default values (null for reference types and 0 for base type). That is the object is not yet associated with the memory address
  - The constructor for the object is called, assigns more meaningful values to the variables and/or perform additional operations needed
  - The new operator returns a reference (memory address) to the newly created object. If the statement is of the form of an assignment, then the address is stored in the object variable.

The dot operator

- An object reference variable is useful to access the instance variables and the methods associated with the object
- This access is performed with the dot (.) operator
- We call the method associated with an object by using the reference variable name followed by the dot and then the method name and its parameters

```
c.increment();
c.increment(3);
c.reset(); // value becomes 0
d.increment();
temp = e.getCount();
e.increment(2);
```

The dot operator

- If the dot operator is used on a reference that is currently null, the Java Runtime environment will throw a NullPointer Exception
- If there are several methods that match a same name, the Java runtime system uses the method that matches the number of parameters
- A method's name together with the number and type of its parameters is called a method's signature
- the signature in Java does not include the type that the method returns. As a consequence Java does not allow tow methods with the same signature to return different types.

The dot operator

- A reference variable can be viewed as a pointer to the corresponding object.
- In other words the reference variable acts as a remote control that can be used to control the newly created object

Modifiers

- Before the definition of classes, instance variables or methods, we always find a series of keywords known as modifiers that can convey additional stipulations about the definition
- There are 4 main such modifiers:
    - Access control modifiers **public, protected, private**
    - The **static** modifier
    - The **abstract** modifier
    - The **final** Modifier

Modifiers

- Access control modifiers control the level of access (also known as visibility) that the definining class grants to other classes within the context of a larger Java program

- The three access control modifiers are the following:
  - **public**: If a member of a class has public visibility, it can be directly referenced from outside the object (for example through a call to the dot notation from an instance of the corresponding class)
  - When a member of a class has private visbility it can be used anywhere inside the class definition but cannot be referenced externally
  - Finally the **protected class modifier** restrict access to subclasses of the given class (i.e classes that inherit from the given class) and classes that belong to the same package (we will discuss this later)

Modifiers

- If no explicit control modifier is given, the defined aspect has what is known as package-private access level. This allows other classes in the same package to have access but not classes from other packages
- This idea of limiting access supports a key idea of object oriented programming known as encapsulation
- In particular, instance variables should be declared with private visibility to promote encapsulation

Modifiers

- Classes will usually be public except if they encode some implementation details.
- More specifically, we only declare a class private when it is nested within another class and it is meant to be an implementation detail
- Similarly constructors are usually public as well (aside from a couple of exception such as singleton class for which we want to limit the number of objects to one)
- Together those two aspects mean that the statement **public class** Counter means that all the other classes are allowed to construct new instances of the counter class.

Modifiers

- To illustrate Access control modifiers one last time, consider the example of the class **Counter()**. In this class, the variable **counter** is private which means that we are allowed to read or edit its value from within methods of that class

- The method getCount on the other hand is declared as public, which, since the class and constructor are public as well, means that any other class can make a call to that method (as an example of this, the class CounterDemo makes the call to c.getCount())

```
public Class Counter {
                public Counter() {}
                private int count;
                public int getCount() {return count; }
}
```

Modifiers

- Note that although it is recommended to declare your variables as private, you can always define public methods that grant access to those variables to oustide classes in a controlled manner. Such methods are known as accessors and mutators
- An example of an accessor method is the method is the getCount method from the Counter class

```
public class Counter {
        private int count;
        public int getCount() {return count; }
```

Modifiers

- One could then wonder why to set some members as private if we can still modify those members through accessors and mutators.
- When a member of a class is public it can be accessed from anywhere in the code. This also means that anybody can modify the value of the variable the way he/she wants anywhere in the code.
- Through the accesssors and mutators however, you can directly control how other programmers can interact with the members of your classes.

Modifiers

- As an illustration of this, consider the example of piece of code that enable a use to specify the number of passengers on a particular flight. If you set the variable in your class as public, any user can then specify any value for that number
- In particular a user could specify a number of passenger that exceed the number of seats in the aircraft.
- To avoid this, you could set the passenger variable of the aircraft private and create a method that makes it possible for another programmer to change its value provided that it does not exceed the plane size and return an error if it does
- Setting a variable as public can be understood as putting no restriction on what other programmers can do with the variable.

Modifiers

- If you later want to add some control on a particular member, you will then have to screen the whole program and manually control each of the spots where this member is used.

Modifiers

- The **static** modifier in Java can be used for any variable or method of a class
- When a variable is declared as **static**, its value is associated with the class as a whole an not with the individual instances of the class
- A static variable is shared among all instances of a class. I.e. there is only one copy of a static variable for all the objects of the class.
- As a consequence, changing the value of a static variable for one object changes it for all the others
- Memory space for a static variable is established when the class that contains it is referenced for the first time in a program. A local variable declared within a method cannot be static.

Modifiers

- When a method of a class is declared as static, it too is associated with the class itself.
- That means that the method is not invoked using an instance of the class through the dot notation
- Instead, it is invoked throught the class name

Modifiers

- To illustrate this, consider the following code snippets

```
public class SloganCounter{
  public static void main (String[] args) {
    obj = new Slogan("Yes we can");
    System.out.println(obj);

    obj = new Slogan("Make America Great Again");
    System.out.println(obj);

    System.out.println();
    System.out.println("Total slogans created: " +
                              Slogan.getCount());

 }
```

Modifiers

```
public class Slogan{
        private static int count = 0;
        public Slogan (String str){
                phrase  = str;
                count++;
        }
          public static int getCount () {
        return count
        }

 }
```

Modifiers

- In your opinion, what will the value of count be ?
- You see that the getCount method is static which allows it to be invoked directly through the class name.
- Also note that as a static method, getCount cannot reference any non static variable

Modifiers

- The **abstract** modifier. A method of a class may be declared as abstract. In this case its signature is provided but without an implementation of the method body. Abstract methods are usually combined with inheritance.
- Any subclass of a class with abstract methods is expected to provide a concrete implementation of the abtract methods

Modifiers

- The last modifier, the **final** modifier is used to declare data that will remain constant throughout the program. In Java a variable that is declared final can be initialized as part of a declaration but can never be assigned a new value.

- If a base type is declared as **final**, then it is considered a constant (and it is usually a good idea to write it in capital letters as shown below). If a reference variable (object) is declared as final, it will always refer to the same object

- When a member variable of a class is declared as **final**, it will typically be declared **static** as well. This is explained by the fact that it is unnecessary to have every instance of a class store the identical value when that value can be shared by the entire class

```java
final int MAX_OCCUPANCY = 427
```

Modifiers

- Declaring a method or class with the **final** modifier has a completely different meaning which really makes sense in the context of inheritance.
- A **final** method cannot be overriden by a subclass and a **final** class cannot be inherited.

Instance variables

- In Java, when defining a class, we can simultaneously declare any number of instance variables
- Each instance of a class maintains its own individual set of instance variables
- The general syntax for declaring one or more instance variabels of a class is the following

```
[modifiers] type identifier1(=initialValue1),
                        identifier2( = initialValue2);
```

- Note that when no initial value is provided, the variable is automatically assigned the value 0

Methods

- A method has two parts: the signature (which defines the name and parameters) and the body which defines what the method does when it is called

```
[modifiers] returnType methodName(type1 param1, ...
                                  typeN paramN){
// method body

}
```

- The returnType specifies the type of the value returned by the method
- The method name can be any valid Java identifier

Methods

- The list of parameters and their type declares the local variables that will store the values passed to the method when invoked. Those can be of any type and identified by any proper Java identifier. The bracket can also be empty (meaning no params are needed).
- Those parameters, as well as the instance variables and the methods of the class can be used within the body of the method.
- When a non static method of a class is called, it is invoked on a specific instance of the class and it can change the state of the this instance.

```java
public void increment(int delta){
        count +=delta;
}
```

Methods and return types

- A Method definition must specify the type of value that the method will return
- If a method does not return a value, then the keyword **void** must be used
- To return a value in java, the body of the method must use the **return** keyword followed by a variable of the appropriate type

```
public int getCount(){
        return count;
}
```

- Note that in Java methods can return only one value. To return multiple values you must return them in a single compound object.

Methods and parameters

- A method's parameters are defined in a coma separated list enclosed in parentheses
- the parameters declaration consists of two parts, the parameter type and the parameter name
- In Java, all parameters are passed **by value** which means that any time we pass a variable to a method, a copy of the value of that variable is made for use within the method body.
- If we pass an int variable, the variable's integer value is copied. If we pass an object, the reference of the object is copied as well

Methods and parameters

- Note that this in particular means that the method cannot change a variable associated to a base/primitive type (as it will only be able to act on the value itself) but it can change the object (as it can act on the copy of the pointer to the object)
- When a method is called, the value in each actual parameter is copied and stored in the corresponding formal parameters
- The formal parameter names in a method header serve as local data for that method. In particular, they don't exist until the method is created and they cease to exist when the method is exited.

A short Illustration (primitive type, Part I)

```
public class Counter{

public int count;
public Counter(){}

public int CountIncrease(int count){ count++;
return count; }

public void CountIncrease2(int count){ count++; }

public void CountIncrease3(){ count++; }

        }
```

A short Illustration (primitive type, Part II)

```
public class displayTest{
        public static void main(String[] args){
           Counter countObj = new Counter();
           int count2 = 0;
           int count3 = 0;
           int count4 = 0;
           count3 = countObj.CountIncrease(count2);
           countObj.CountIncrease2(count4);
```

A short Illustration (primitive type, Part III)

```
System.out.print(countObj.count);
              System.out.printf("%n");
              System.out.print(count2);
              System.out.printf("%n");
              System.out.print(count3);
              System.out.printf("%n");
              System.out.print(count4);
              System.out.printf("%n");
              System.out.print(countObj.count);
              System.out.printf("%n");
              countObj.CountIncrease3();
              System.out.print(countObj.count);
       }}
```

A short Illustration (reference type, Part I)

```
public class testObject{

        public int value0;

        public testObject(){}


        }
```

A short Illustration (reference type, Part II)

```
public class testObject2{
public static void increasetO(testObject to){
        to.valueO = to.valueO + 1;
}
public static void main(String[] args){

testObject to1 = new testObject();

System.out.print(to1.valueO);
System.out.printf("%n");

increasetO(to1);

System.out.print(to1.valueO);
System.out.printf("%n");

}}
```

A short Illustration (reference type, Part III)

- Why do I need the first method to be static ?

Another Illustration

- Recall the counter class

```
public class Counter {
  private int count;
  public Counter() {}
  public Counter(int initial){count = initial; }
  public int getCount() {return count; }
  public void increment() { count++; }
  public void increment(int delta) {count += delta; }
  public void reset() {count = 0; }
}
```

Another Illustration

- We then define a new class with the following two methods

```
public static void Method1(Counter c){
c = new Counter();
c.increment();
}

public static void Method2(Counter c){
c.increment();
}
```

- Imagine that we apply each method to an object, let's call it striker from the Counter class that has just been initialized (i.e striker.count = 0). What will be the value of striker.count after calling each method?

Constructors

- Constructors (which we have already encountered in the previous snippets) are a special kind of method that are used to initialize a newly created instance of the class
- Constructors initialize each instance variable of the object (note that we have seen that we can have multiple constructors that initialize part of the instance variables)
- The general syntax to call a constructor is the following

```
modifiers name(type0 parameter0, ..., typeN parameterN){
// constructor body
}
```

Constructors

- Constructors cannot be static, abstract or final so the only modifiers that are allowed are the visibility modifiers: **public protected, private**
- The name of the constructor must be identical to the name of the class it constructs
- We don't specify a return type for a constructor (not even **void**)
- When a user of class creates an instance using the syntax

  ```
  Counter d = new Counter(5)
  ```

  The **new** operator is responsible for returning a reference to the new instance. The constructor only initializes the state of the new instance

Constructors

- A class can have multiple constructors but each must have a different signature

```
public Counter() {}
public Counter(int initial){count = initial; }
```

- If no constructor is defined, java provides a default implicit constructor having zero argument and leaving all instance variables initialized to their default values

The keyword **this**

- Within the body of a non-static (recall that the word static refers to the class itself) method, the keyword **this** is defined as a reference to the instance upon which the method was invoked

- There are three main reasons why the keyword **this** may be useful:

    - To store the reference in a variable or pass it to a method that can take an instance of that type as an argument

    - To differentiate between an instance (i.e. class) variable and a local variable of the same name

```
public Counter(int count){
this.count = count;
}
```

The keyword **this**

- There are three main reasons why the keyword **this** may be useful:
    - To store the reference in a variable or pass it to a method that can take an instance of that type as an argument
    - To differentiate between an instance (i.e. class) variable and a local variable of the same name
    - To allow one constructor body to invoke another constructor body

```
public Counter(){
this(0) /* use one parameter constructor
with value 0 */
}
```

The main method

- Some Java classes are meant to be used by other classes but are not intended to serve as a self standing program
- The primary control for an application in Java must begin in some class with the execution of a special method named **main**.
- The **main** method is declared as follows:

```
public static void main(String[] args){
// main method body
}
```

The main method

- The args parameter is an array of `String` objects, that is args[0] is the first string, args[1] is the second string,...
- Those represent what are known as command line arguments that are given by the user when the program is executed (we will say more on this later)
- a Java program can be called from the command line using the java command followed by the name of the Java class whose main method we want to run plus additional arguments
- If we had defined our program to take an argument, we would have invoked the program as the second line below

```
java class1

java class1 string1
```

The main method

- In the case of an additional argument, it is up to the body of the main method to interpret string1 as the required information, just as in the example below

```java
for(int i = 0; i < args.length; i++) {
        System.out.println("Argument is: "+args[i]);
    }
```