Ch.01

Data Structure and Algorithm (Introduction to Data Structure)

(kmitl) cs-department

Data Structures and Algorithms

Learning Outcomes

CLO-1 Understand and use the process of abstraction in problem solving.

CLO-2 Analyze step by step and develop algorithms to solve real world problems.

CLO-3 Implementing basic data structures and algorithms.

CLO-4 Understanding various searching and sorting techniques

Teaching Plan		
Week	Topic	CLO
1	Data, Algorithms, and JAVA review	CLO-1, CLO-2
2	Analysis of data structures and algorithms	CLO-1
3	Array, Linked List	CLO-2, CLO-3
4	Stack	CLO-2, CLO-3
5	Queue	CLO-2, CLO-3
6	Heap and Priority Queue	CLO-2, CLO-3
7	Recurrent relation	CLO-2
9	Tree concept and Binary Search Tree	CLO-2, CLO-3
10	Balanced Binary Search Tree	CLO-2, CLO-3
11	Sorting I	CLO-3, CLO-4
12	Sorting II	CLO-3, CLO-4
13	Graph representation and traversal	CLO-4
14	Graph algorithms	CLO-3, CLO-4
15	Hashing	CLO-2, CLO-3

Outline

- Overview
  - Introduction
    - Why Data Structures & Algorithms?
    - What is data?
    - Data vs Information
    - What is Algorithms?
    - What is computer?
  - What to learn in DSA class?
  - An example of Data Structure in real life
- Java Review
  - Java Programming
  - Built-in Data Types
  - Operators
  - String and Math Library
  - Type Conversion
  - Flow Control
  - 1-D Array, 2-D Array
  - Object Oriented Programming
  - Class and Object
    - Instance variables
    - Instance methods
  - OOP principles

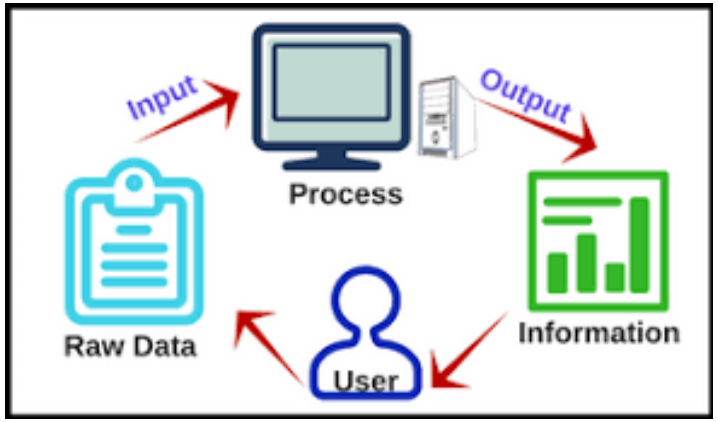
Why Data Structures & Algorithms?

Data structures + algorithms = program

What is data?

- Data is facts
- There are abundant data in the world, we can only make use some of them
- With computer, have capability to make use more and more of them.
- Basic Operations – CRUD
  - Create
  - Retrieve
  - Update
  - Delete

- Data vs Information
  - In some definition, data mean raw data where information mean data ready to be used
  - We process data to make it useful



What is Algorithms?

- Can be viewed as “ways to achieve a certain goal”.
  - Processing data is one of them.
- An algorithm is a recipe which leads to the required output.
- Recent development: data can be mined, algorithms can be learnt.

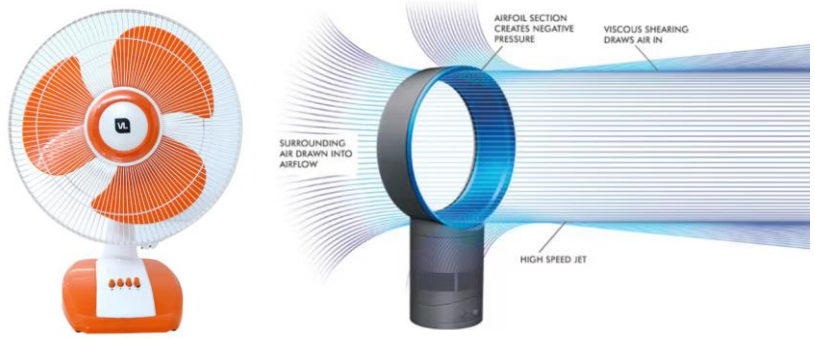
What is computer?

- Information processing machine
- Help us solving problem by processing information faster

What to learn in DSA class?

- Characteristics of good data structures / algorithms and tool for analyzing them.
- Basic data structures
  - Array, Linked List
- Concept of Abstract Data Type (ADT) and some simple ADT
  - Stack, Queue, Heap, Tree, Graph
  - A mathematical model of data types.
    - We only care about what it can do and sometimes its efficiency, but not how it implement.

- Abstraction (Daily life perception)



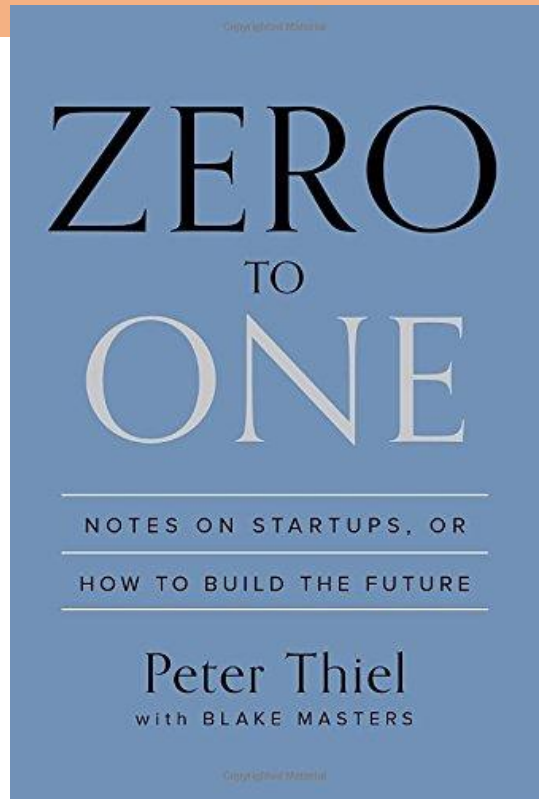
- More example → speaker

What to learn in DSA class? (cont.)

- Fundamental algorithms
  - Sorting and searching
- Some advance data structures and algorithms
  - Binary Search Tree (BST), AVL Tree, Splay Tree
  - Minimal Spanning Tree, Shortest Path
  - Hashing

Imagine if you have only one book.

- You don’t need any structure or algorithm for your data (book)

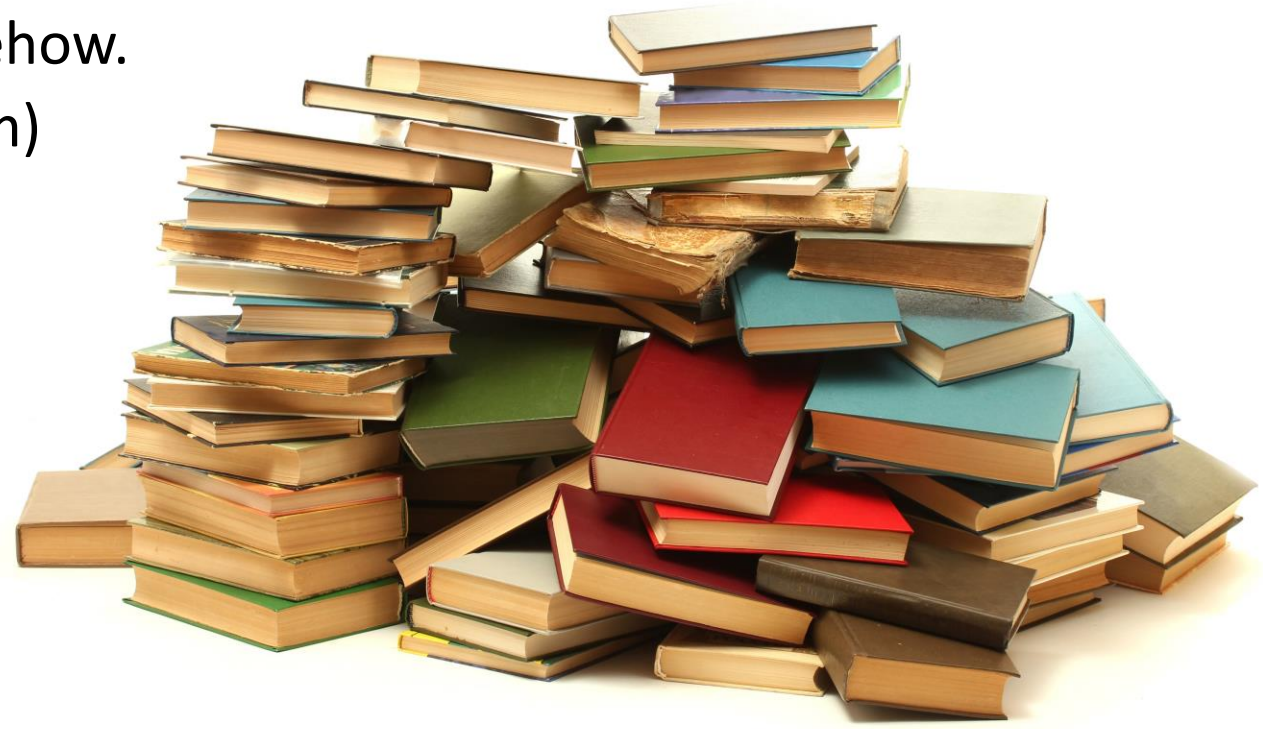


How about seven books

- Well, still manageable.

Make it a hundred

- Now, I think we need some order!
  - We need to structure them somehow.
  - We also need a way (an algorithm) to deal with them.



Structured them

- Alphabetical order seems to be a smart choice
- Binary search can be used to find a book quickly
- Data structure → ordered array
- Algorithm → binary search

- What other algorithms are involved?

Let keep going..

- Is alphabetical order still enough?
- Any good idea?

Let structure them!

- Dewey Decimal Classification – DDC
  - a.k.a. Library System
  - Using hierarchy classification
- For example

500 Natural sciences and mathematics

510 Mathematics

516 Geometry

516.3 Analytic geometries

516.37 Metric differential geometries

516.375 Finsler geometry

*** Cuypers Library, Amsterdam, Netherlands*

Beyond Data Structures & Algorithms

- Where to physically store the books?
- How to connect an alphabetical order to the DDC?
- How to add/take out/return/remove books?
- What is the function of librarian, comparing to computer system?

Summarize

- Data Structures + Algorithms = Program
- There are abundant data in the world
  - Some of them are useful to us.
  - We have capability to make use more and more of them.
- We create, retrieve, update, and delete useful data.
- Data can be processed to make it even more useful.
- Algorithms can be thought as ways to process data
- Computer makes DSA necessary
  - although DSA are useful even before the age of computer
- In Data Structures and Algorithm class, we will learn
  - Some basic data structures and algorithms
  - How to recognize good ones
  - Also, some advanced ones

Lecture 1

JAVA Review

References:

https://introcs.cs.princeton.edu/java/11cheatsheet/

https://www.upgrad.com/blog/types-of-literals-in-java/

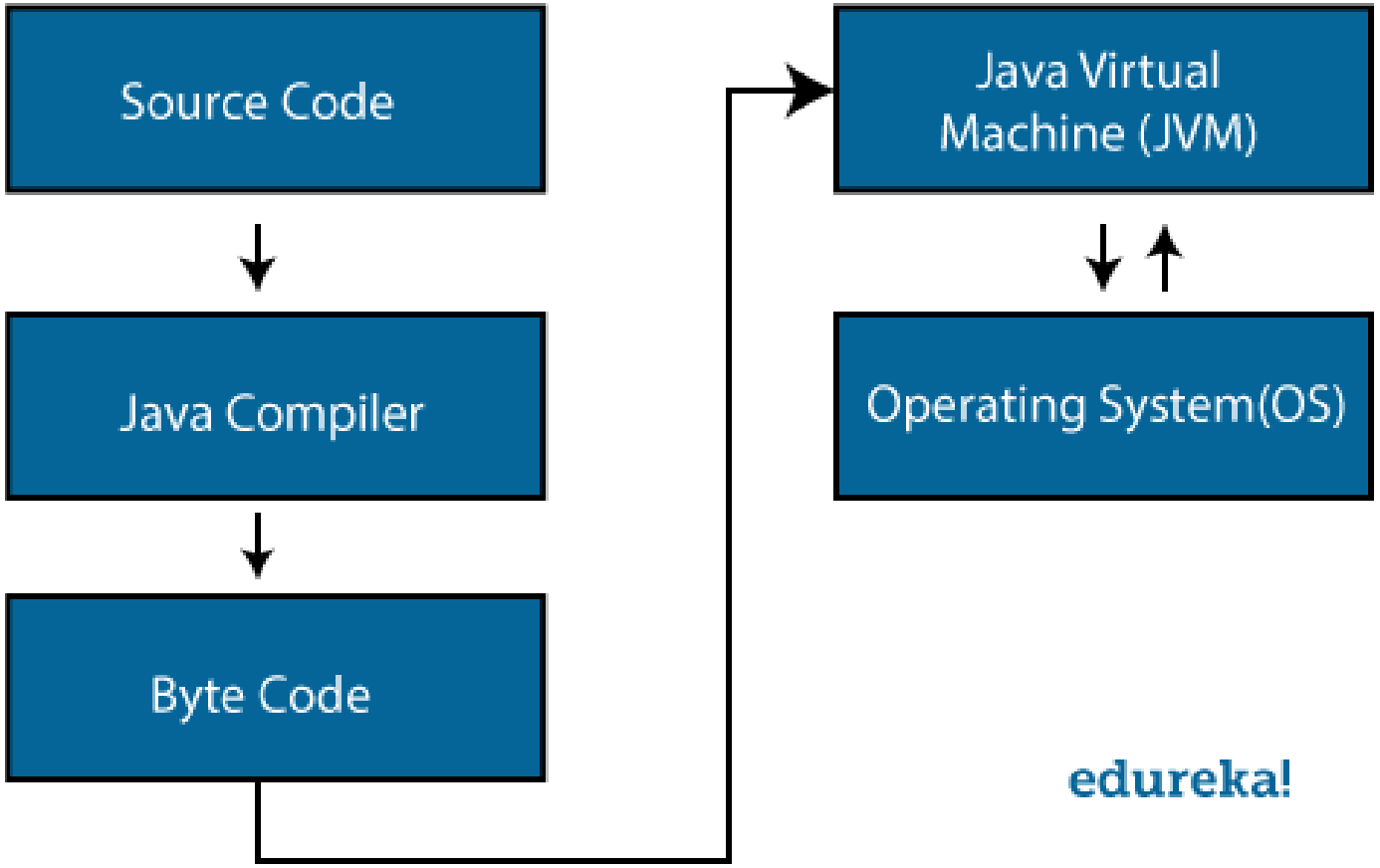
http://www2.hawaii.edu/~tp_200/lectureNotes/review_of_some_java_basics.htm

http://comet.lehman.cuny.edu/sfakhouri/teaching/cmp/cmp338/lecturenotes-3rdEdition/Chapter-01.pdf

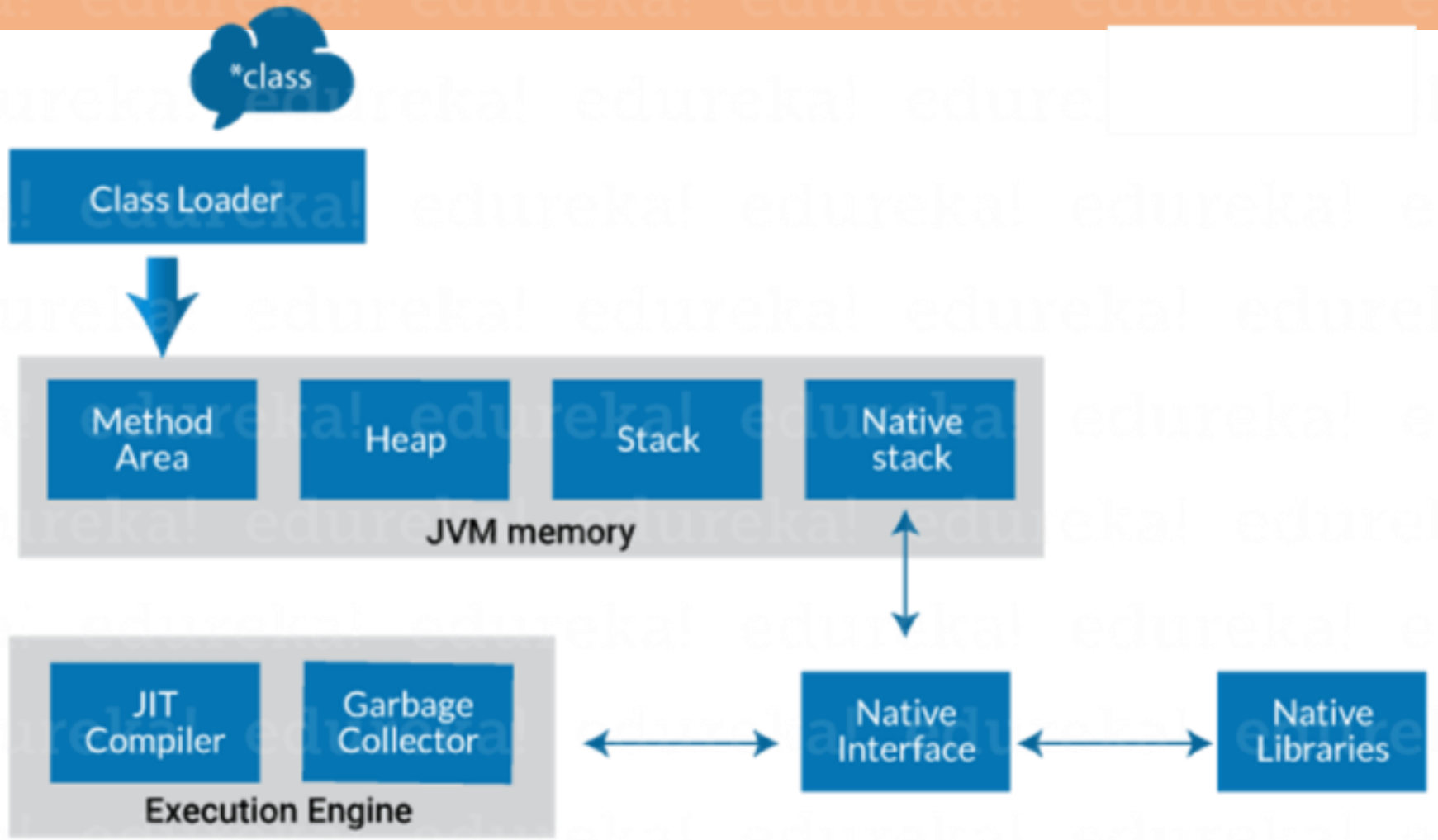
JAVA Application

- Collection of classes
  - One of them have a designated main method.
  - Can be packed into JAVA archive (.jar) and run as an executable (.exe) in JAVA ready machine
- Create by SUN Microsystem
  - Slogan from 1995: Write (and compile) once, run anywhere
  - Bought by Oracle in 2010,
    - Thus, Oracle's implementation (licensed) is the de facto standard.
    - OpenJDK is one of the notable free implementation.

JAVA Programing



JAVA Virtual Machine – JVM



HelloYou.java

```
1  import java.util.Scanner;
2
3  public class HelloYou {
4      public static void main(String args[]) {
5          Scanner in = new Scanner(System.in);
6          System.out.print("Enter your name: ");
7          String yourName = in.nextLine();
8          System.out.println("Hello "+yourName+"!");
9      }
10 }
11
12
```

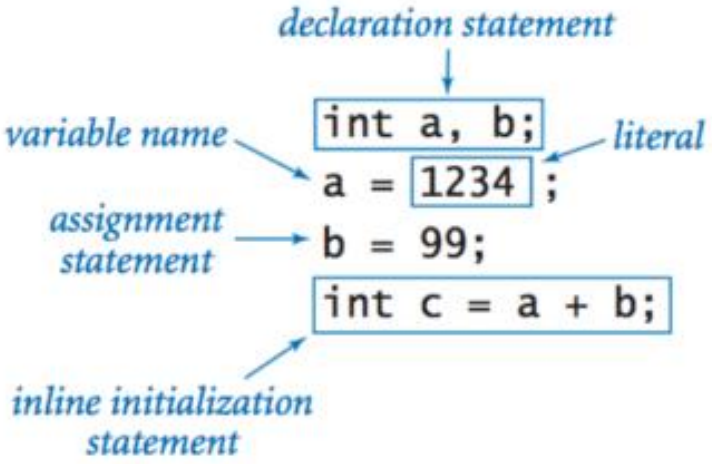
Some Reserves Words

boolean	default	for	private	switch
break	double	if	*protected*	this
case	else	import	public	throws
catch	extends	int	return	try
char	final	new	static	void
class	float	*package*	super	while

Built-in Data Types

type	*set of values*	*common operators*	*sample literal values*
`int`	integers	`+ - * / %`	`99 12 2147483647`
`double`	floating-point numbers	`+ - * /`	`3.14 2.5 6.022e23`
`boolean`	boolean values	`&& || !`	`true false`
`char`	characters		`'A' '1' '%' '\n'`
`String`	sequences of characters	`+`	`"AB" "Hello" "2.5"`

- Declaration and assignment



Literals

- boolean
  - {true, false}
- char
  - {'a','b',...}
- int
  - Decimal: 232
  - Octal: 0231 *** leading zero*
  - Hexadecimal: 0X1A *** capital X*
  - Binary: 0b1101
- String
  - "String" *** not primitives*
- Object
  - null

Primitives and Wrapper Classes

- Boolean type
  - boolean → Boolean
- Character type
  - char → Character ***Unicode*
- Integer types
  - byte → Byte
  - short → Short
  - int → Integer
  - long → Long
- Floating point
  - float → Float
  - double → Double

Operators

Integers.

values	integers between -2^{31} and $+2^{31}-1$					
typical literals	`1234`	`99`	`0`	`1000000`		
operations	*sign*	*add*	*subtract*	*multiply*	*divide*	*remainder*
operators	`+ -`	`+`	`-`	`*`	`/`	`%`

Floating-point numbers.

values	real numbers (specified by IEEE 754 standard)			
typical literals	`3.14159`	`6.022e23`	`2.0`	`1.4142135623730951`
operations	*add*	*subtract*	*multiply*	*divide*
operators	`+`	`-`	`*`	`/`

Booleans.

values	*true or false*		
literals	`true`	`false`	
operations	and	or	not
operators	`&&`	`||`	`!`

Comparison Operators

op	*meaning*	`true`	`false`
==	*equal*	`2 == 2`	`2 == 3`
!=	*not equal*	`3 != 2`	`2 != 2`
<	*less than*	`2 < 13`	`2 < 2`
<=	*less than or equal*	`2 <= 2`	`3 <= 2`
>	*greater than*	`13 > 2`	`2 > 13`
>=	*greater than or equal*	`3 >= 2`	`2 >= 3`

String

Special Characters

Special characters	*Display*
\'	Single quotation mark
\"	Double quotation mark
\\	Backslash
\t	Tab
\b	Backspace
\r	Carriage return
\f	Formfeed
\n	Newline

public class String

	`String(String s)`	*create a string with the same value as* `s`
	`String(char[] a)`	*create a string that represents the same sequence of characters as in* `a[]`
`int`	`length()`	*number of characters*
`char`	`charAt(int i)`	*the character at index* `i`
`String`	`substring(int i, int j)`	*characters at indices* `i` *through* `(j-1)`
`boolean`	`contains(String substring)`	*does this string contain* `substring`*?*
`boolean`	`startsWith(String prefix)`	*does this string start with* `prefix`*?*
`boolean`	`endsWith(String postfix)`	*does this string end with* `postfix`*?*
`int`	`indexOf(String pattern)`	*index of first occurrence of* `pattern`
`int`	`indexOf(String pattern, int i)`	*index of first occurrence of* `pattern` *after* `i`
`String`	`concat(String t)`	*this string, with* `t` *appended*
`int`	`compareTo(String t)`	*string comparison*
`String`	`toLowerCase()`	*this string, with lowercase letters*
`String`	`toUpperCase()`	*this string, with uppercase letters*
`String`	`replace(String a, String b)`	*this string, with* `a`*s replaced by* `b`*s*
`String`	`trim()`	*this string, with leading and trailing whitespace removed*
`boolean`	`matches(String regexp)`	*is this string matched by the regular expression?*
`String[]`	`split(String delimiter)`	*strings between occurrences of* `delimiter`
`boolean`	`equals(Object t)`	*is this string's value the same as* `t`*'s?*
`int`	`hashCode()`	*an integer hash code*

Output

Printing.

`void`	`System.out.print(String s)`	*print s*
`void`	`System.out.println(String s)`	*print s, followed by a newline*
`void`	`System.out.println()`	*print a newline*

Parsing from String to number

Parsing command-line arguments.

`int`	`Integer.parseInt(String s)`	*convert s to an `int` value*
`double`	`Double.parseDouble(String s)`	*convert s to a `double` value*
`long`	`Long.parseLong(String s)`	*convert s to a `long` value*

Math Library

public class Math

double abs(double a)	*absolute value of a*
double max(double a, double b)	*maximum of a and b*
double min(double a, double b)	*minimum of a and b*
double sin(double theta)	*sine of theta*
double cos(double theta)	*cosine of theta*
double tan(double theta)	*tangent of theta*
double toRadians(double degrees)	*convert angle from degrees to radians*
double toDegrees(double radians)	*convert angle from radians to degrees*
double exp(double a)	*exponential* (e^a)
double log(double a)	*natural log* ($\log_e a$, *or* $\ln a$)
double pow(double a, double b)	*raise a to the bth power* (a^b)
long round(double a)	*round a to the nearest integer*
double random()	*random number in* [0, 1)
double sqrt(double a)	*square root of a*
double E	*value of e (constant)*
double PI	*value of π (constant)*

Type Conversion

expression	*expression type*	*expression value*
`(1 + 2 + 3 + 4) / 4.0`	`double`	`2.5`
`Math.sqrt(4)`	`double`	`2.0`
`"1234" + 99`	`String`	`"123499"`
`11 * 0.25`	`double`	`2.75`
`(int) 11 * 0.25`	`double`	`2.75`
`11 * (int) 0.25`	`int`	`0`
`(int) (11 * 0.25)`	`int`	`2`
`(int) 2.71828`	`int`	`2`
`Math.round(2.71828)`	`long`	`3`
`(int) Math.round(2.71828)`	`int`	`3`
`Integer.parseInt("1234")`	`int`	`1234`

Implicit type casting

- `int` → `long` → `float` → `double`
- Ex:

```
double d = 10;   // cast int to double.
                 // do not confuse with
                 // auto-boxing.
```

Flow Control

- Condition statement
  - If, If else
  - Switch
  - Terary expression (Ternary operator)
- Interative statements
  - while
  - do – while
  - for
- Control Transfer
  - return, break, continue

Array

Inline array initialization.

```
String[] SUITS = { "Clubs", "Diamonds", "Hearts", "Spades" };

String[] RANKS = {
   "2", "3", "4", "5", "6", "7", "8", "9", "10",
   "Jack", "Queen", "King", "Ace"
};
```

Typical array-processing code

create an array with random values	`double[] a = new double[n];`
	`for (int i = 0; i < n; i++)`
	`  a[i] = Math.random();`

print the array values, one per line	`for (int i = 0; i < n; i++)` `  System.out.println(a[i]);`

find the maximum of the array values	`double max = Double.NEGATIVE_INFINITY;`
	`for (int i = 0; i < n; i++)`
	`  if (a[i] > max) max = a[i];`

compute the average of the array values	`double sum = 0.0;`
	`for (int i = 0; i < n; i++)`
	`  sum += a[i];`
	`double average = sum / n;`

reverse the values within an array	`for (int i = 0; i < n/2; i++)`
	`{`
	`  double temp = a[i];`
	`  a[i] = a[n-1-i];`
	`  a[n-i-1] = temp;`
	`}`

copy sequence of values to another array	`double[] b = new double[n];`
	`for (int i = 0; i < n; i++)`
	`  b[i] = a[i];`

2D Array

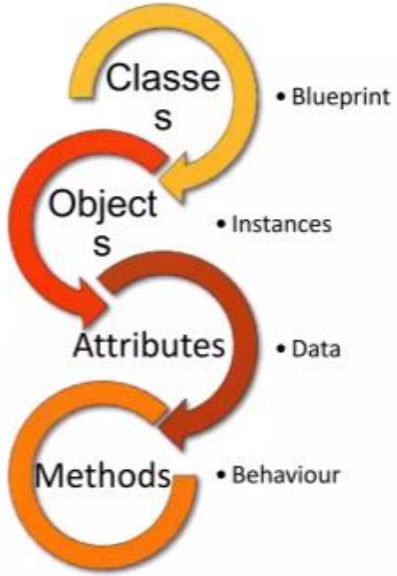
Inline initialization.

```
double [][] a =
{
   { 99.0, 85.0, 98.0,  0.0 },
   { 98.0, 57.0, 79.0,  0.0 },
   { 92.0, 77.0, 74.0,  0.0 },
   { 94.0, 62.0, 81.0,  0.0 },
   { 99.0, 94.0, 92.0,  0.0 },
   { 80.0, 76.5, 67.0,  0.0 },
   { 76.0, 58.5, 90.5,  0.0 },
   { 92.0, 66.0, 91.0,  0.0 },
   { 97.0, 70.5, 66.5,  0.0 },
   { 89.0, 89.5, 81.0,  0.0 },
   {  0.0,  0.0,  0.0,  0.0 }
};
```

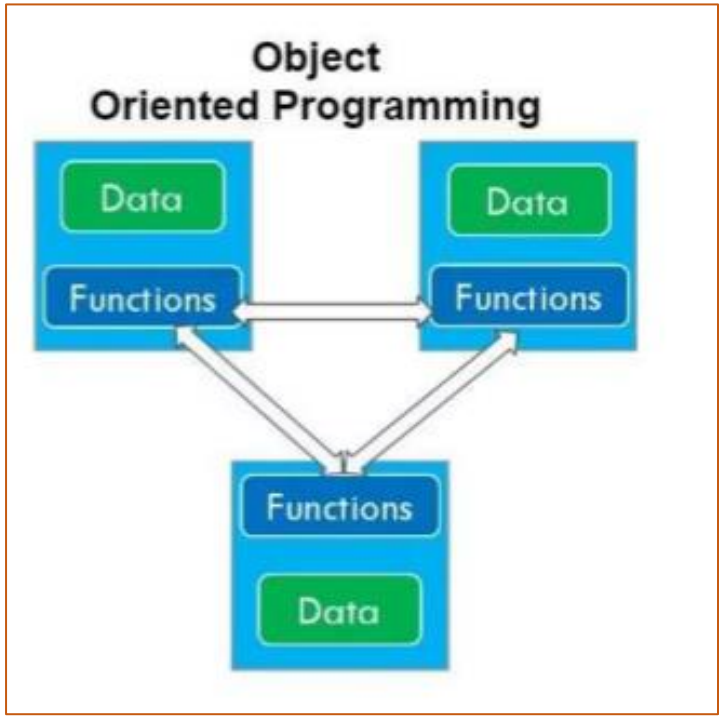
Building blocks of an OOP program

The fundamental building blocks of an OOP program are :

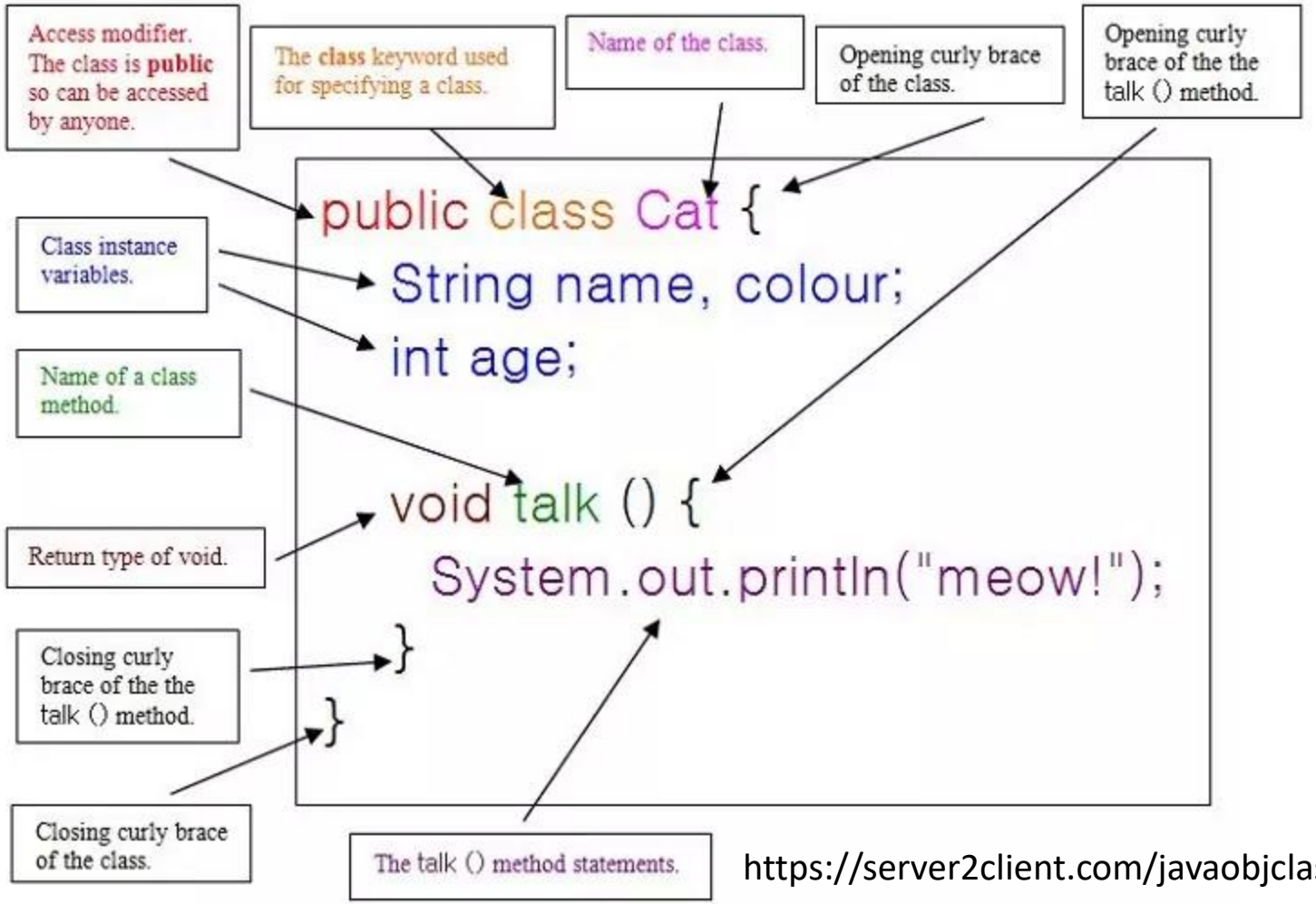
- Classes
- Objects
- Methods
- Attributes



https://www.slideshare.net/AnushkaGupta763558/oops-252140976

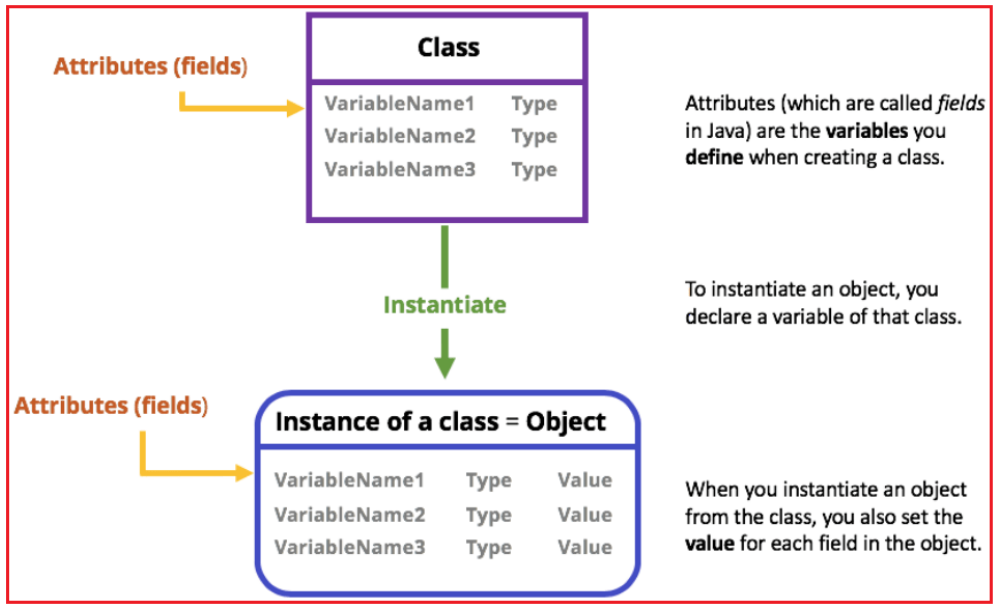


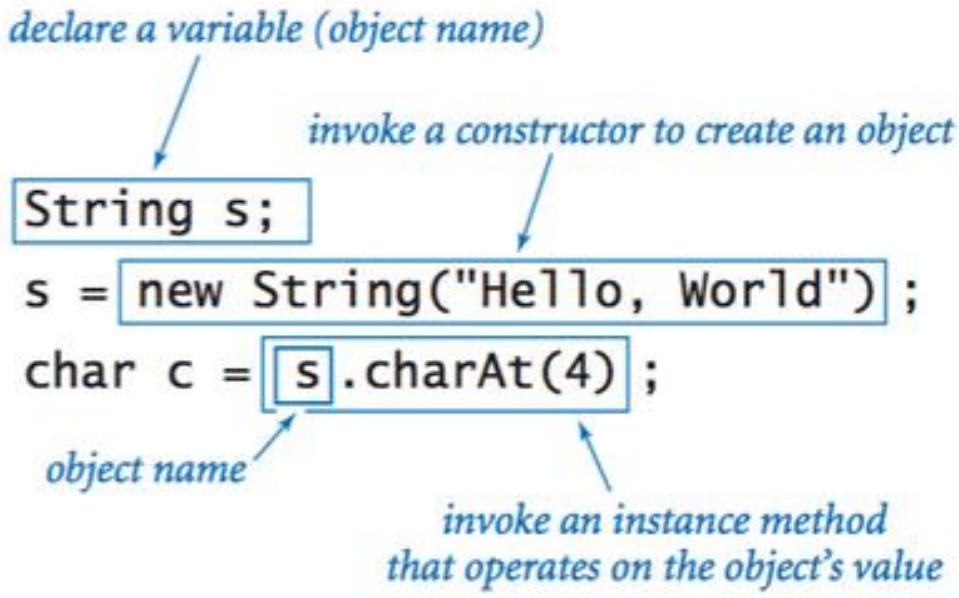
Classes



https://server2client.com/javaobjclasses/classstructure.html

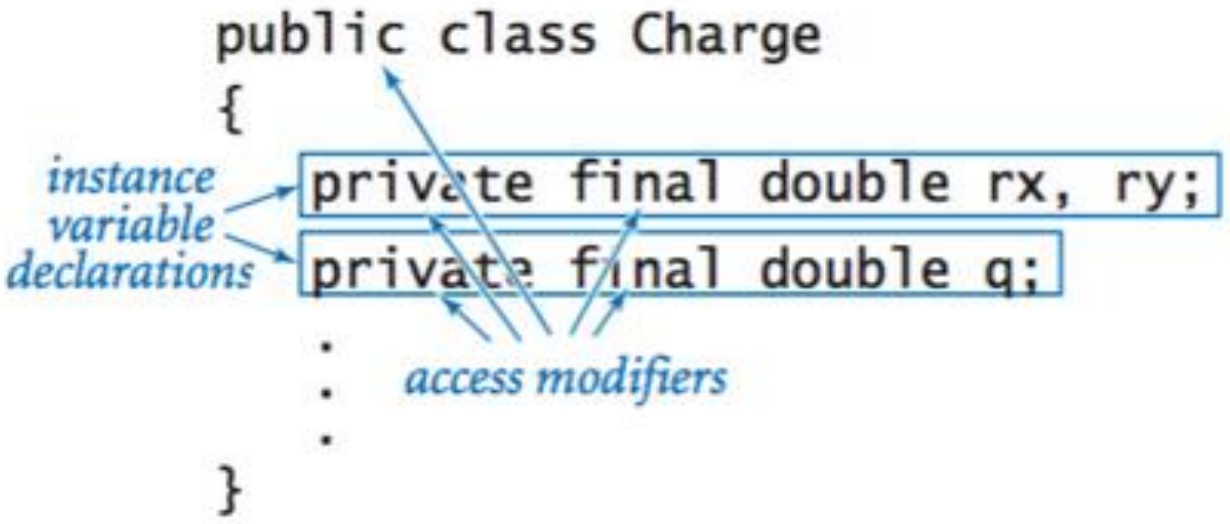
Using an Object



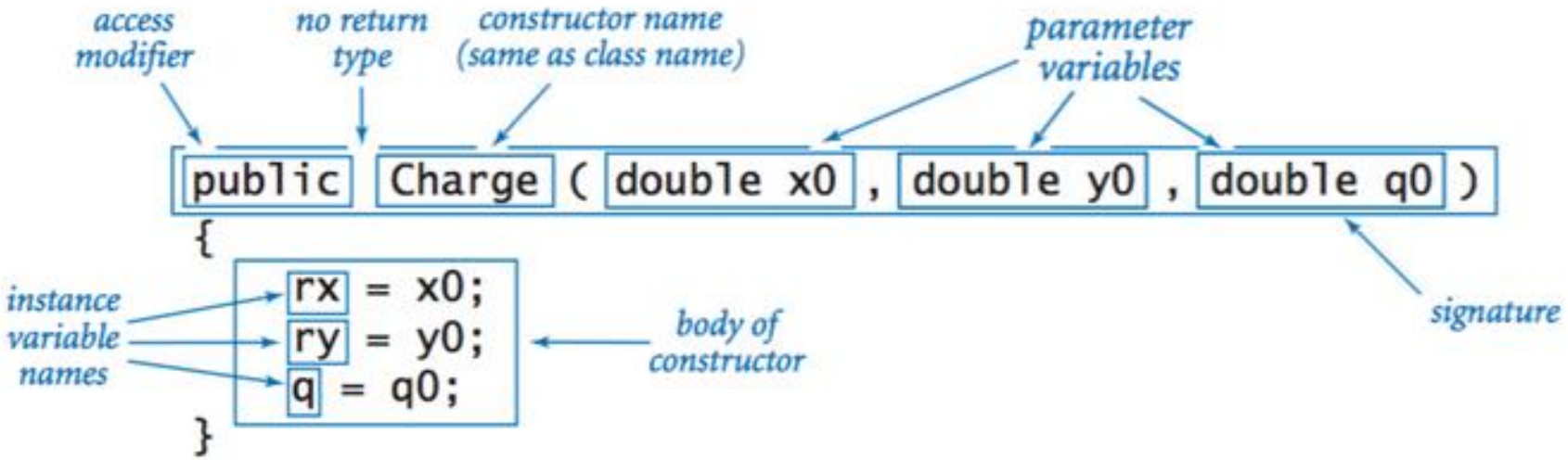


https://dotnettutorials.net/lesson/object-oriented-programming-in-java/

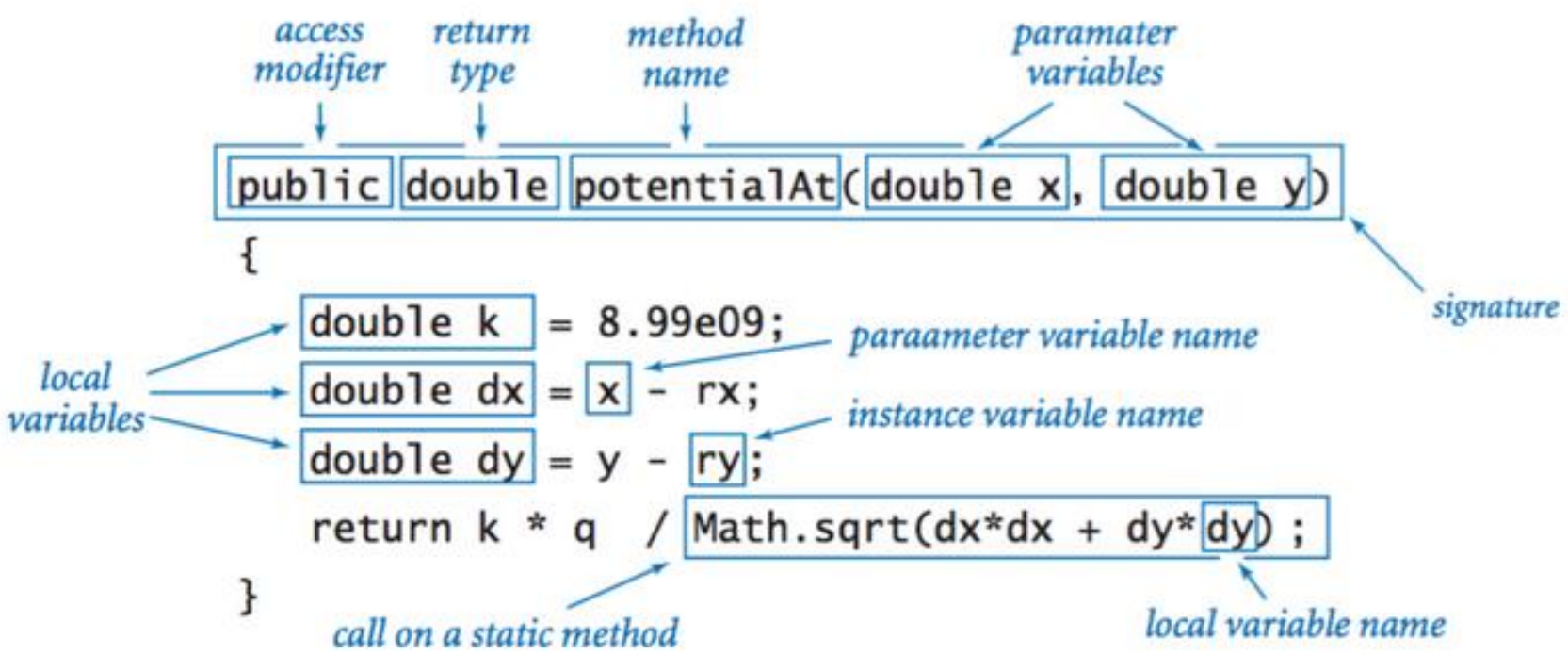
Instance Variable



Object Constructors



Instance Methods



Static vs. Dynamic

- It's important to understand the distinction between static and dynamic behaviors in classes.
  - You can invoke a class static attribute or method ***directly by naming the class***.
  - Whereas, in a (dynamic) non-static class: you ***need to create an instance*** with the ***new*** operator.
- (Again) A class is just a packaging of methods and data.
  - Static: There is only ONE copy of the data, and only ONE copy of the methods in memory.
  - You can create any number of instances of a dynamic class.
    - each instance has its own copy of variables
    - It helps to also think: each instance has its own copy of methods

https://www2.seas.gwu.edu/~simhaweb/java/objects.html

Static vs. Dynamic

```
static void demo1() {
  println("-demo1---");
  A.method_a1(890);
}
static void demo2() {
  println("-demo2---");
  B b1 = new B(22);
  B b2 = new B(33);
  b1.method_b1();
  b2.method_b1();
}
```

```
public class A {
  static int val_a1 = 1;
  static void method_a1(int x) {
      print("From method_a1() " + val_a1 );
      println(" local argument value
                                is " + x);
  }
}
public class B {
  int val_b1;
  B(int v) { val_b1 = v;  }
  void method_b1() {
    print("From method_b1() " + val_b1 );
    println(" There is not argument received.");
  }
}
```

Static vs. Dynamic

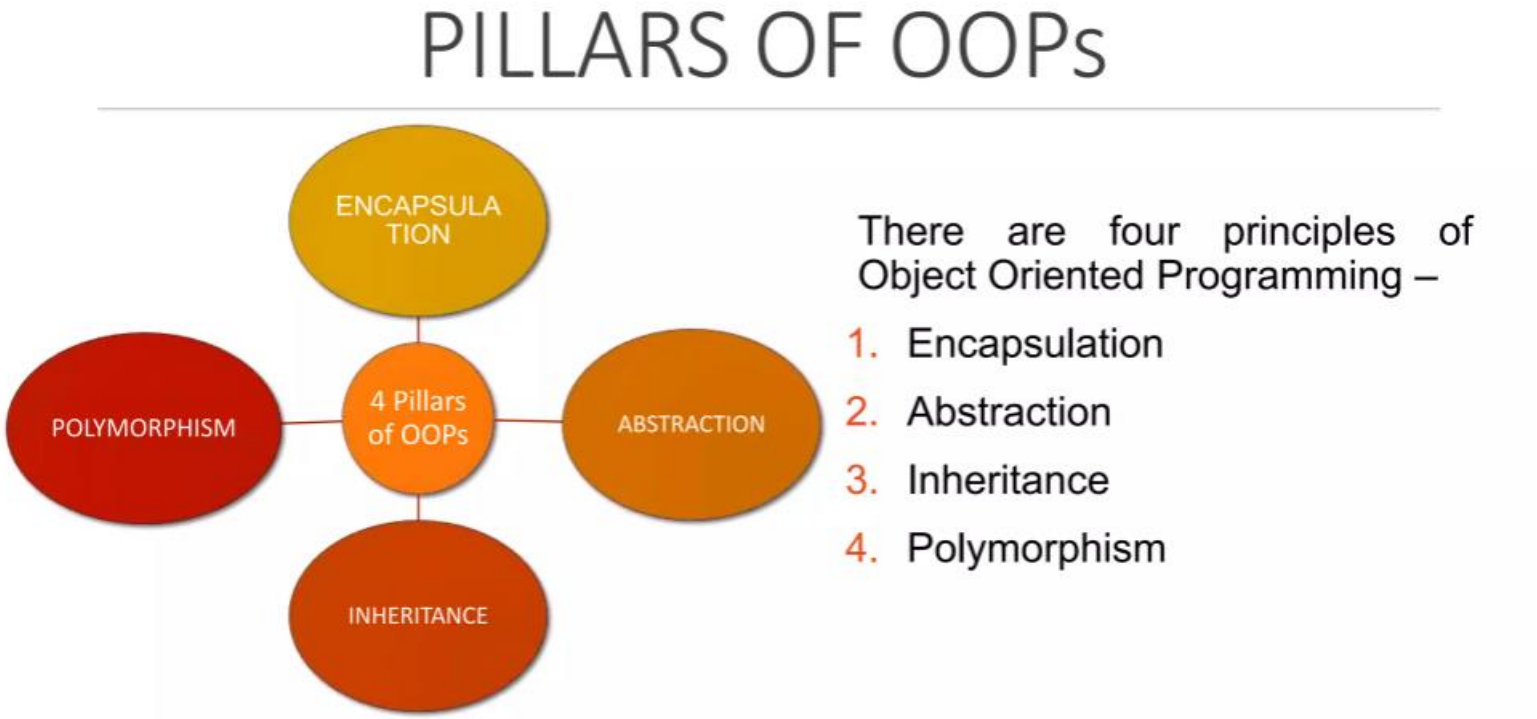
```
static void demo3() {
  println("-demo3---");
  C c1 = new C(444);
  C c2 = new C(555);
  c1.method_c1(908);
  c1.method_c1(908);
  C.method_c2();
}
static void demo4() {
  println("-demo4---");
  D d1 = new D(6666);
  D d2 = new D(7777);
  D.method_d1();
  D.method_d2();
  d1.method_d3();
  d2.method_d3();
  D.method_d4();
}
```

```
public class C {
  int val_c1;
  static int val_c2 = 3;
  C(int v) {  val_c1 = v;  }
  void method_c1(int x) {
    print("From method_b1() "
                              + val_c1 );
    println(" local argument value
                               is " + x);
    println("** non static method may
    access static attribute " + val_c2);
  }
  static void method_c2() {
    println("static method cannot access
             non-static attribute "
                    /* + val_c1 */ );
      // methed_c2 does not knows
      // **whose** val_c1 is being
      // referred to.
    println("non static method
      **may**access static attribute "
      + val_c2);
  }
```

```
public class D {
  int val_d1;
  static int val_d2 = 4;
  D(int v) {
      val_d1 = v;
  }
  static void method_d1() {
    println("this is my only assigned task");
  }
  static void method_d2() {
    // method may be called from another method
    method_d1();
    println("leaving method_d2()");
  }
  void method_d3() {
      // method may be called from another method
      // static method may be called from non-static
      method_d1();
      println("leaving method_d3()");
  }
  static void method_d4() {
      // but not the other way around (again
      // think of non-static method of **which**
      // instance)
    // method_d3();
    println("leaving method_d4()");
  }
}
```

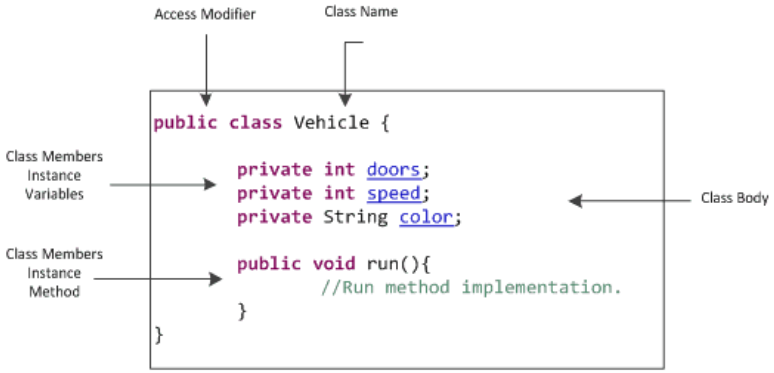
- non-static attribute / method may not be accessed / invoked in static method method

Java OOP



https://www.slideshare.net/AnushkaGupta763558/oops-252140976

Encapsulation (Access Modifier)

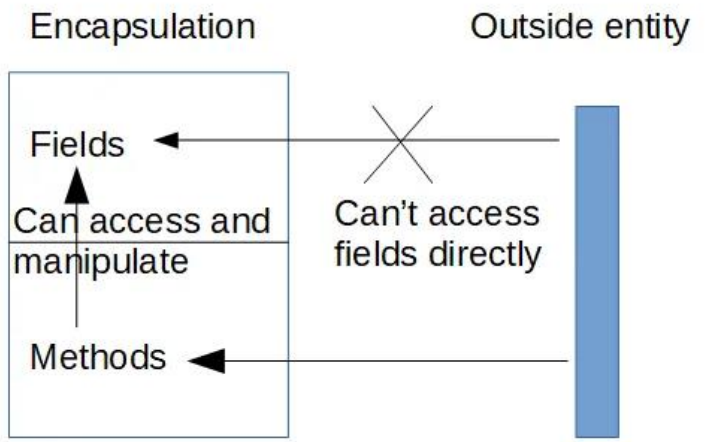


https://www.w3resource.com/java-tutorial/java-class-methods-instance-variables.php

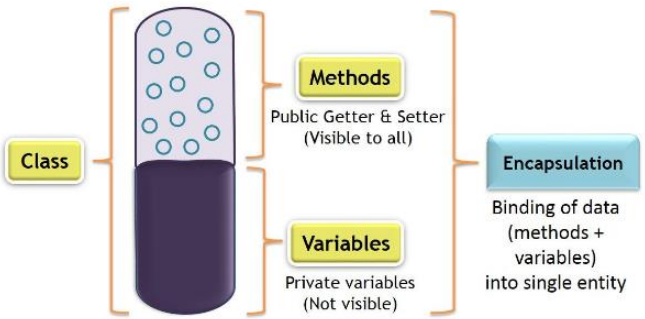
Access Modifier	Within the Class	Other Classes [Within the Package]	In Subclasses [Within the package and other packages]	Any Class [In Other Packages]
public	Y	Y	Y	Y
protected	Y	Y	Y	N
default	Y	Y	Same Package – Y Other Packages - N	N
private	Y	N	N	N

https://www.startertutorials.com/corejava/access-control.html

- Used for classes, variables, and methods

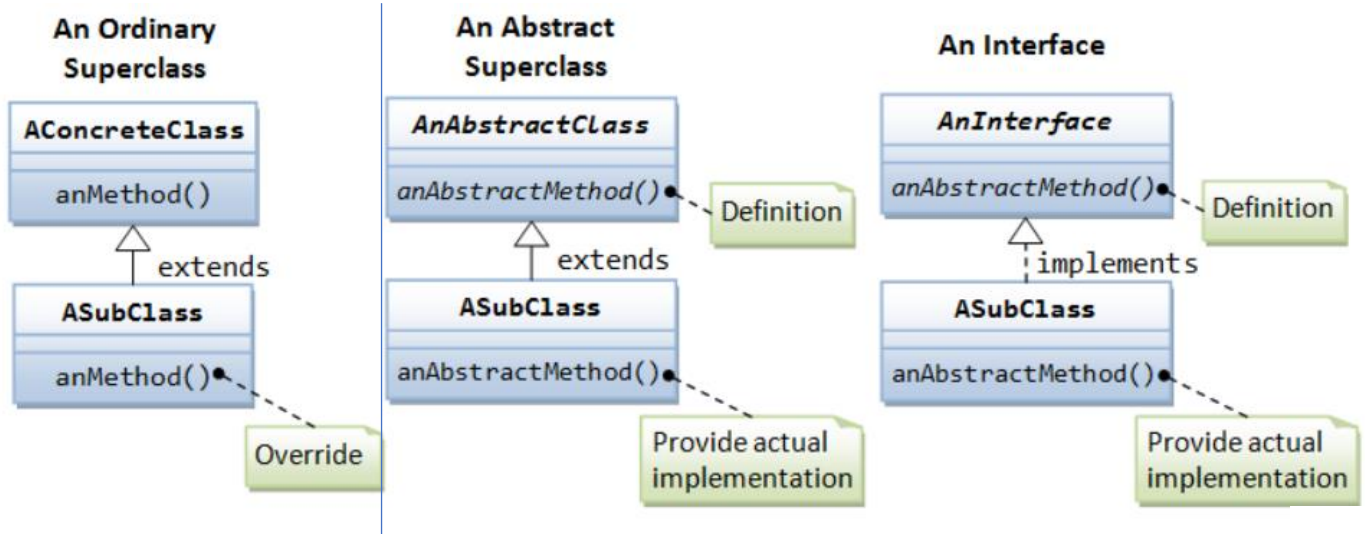


https://www.netjstech.com/2015/04/encapsulation-in-java.html



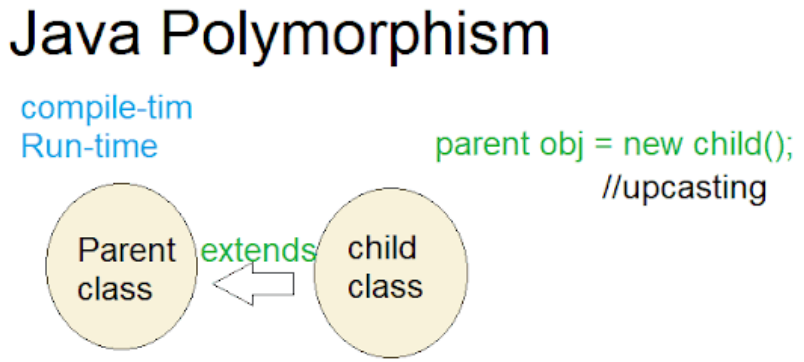
https://www.pinterest.com/pin/why-encapsulation--733805333044557573/

Inheriance vs Abstraction



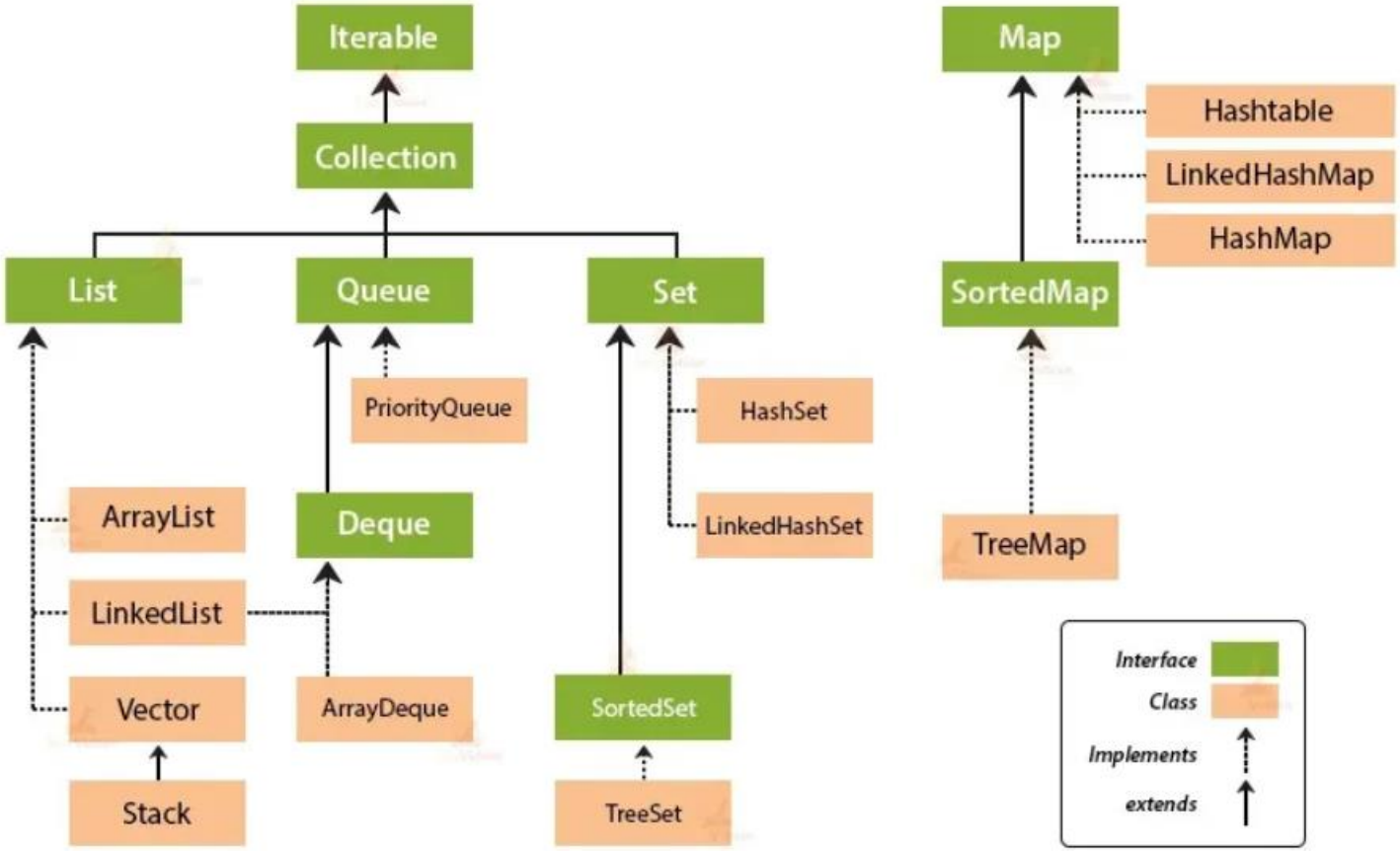
https://www3.ntu.edu.sg/home/ehchua/programming/java/J3b_OOPInheritancePolymorphism.html

polymorphism



https://www.thecodingshala.com/2019/07/java-runtime-polymorphism-coding-shala.html

Collections Framework Hierarchy in Java



https://www.freecodecamp.org/news/java-collections-framework-reference-guide/

Java Collections Efficiency

Method	Array List	Linked List	Stack	Queue	TreeSet /Map	[Linked] HashSet /Map	Priority Queue
`add` or `put`	O(1)	O(1)	O(1)*	O(1)*	O(log *N*)	O(1)	O(log *N*)*
`add` at index	O(*N*)	O(*N*)	-	-	-	-	-
`contains`/ `indexOf`	O(*N*)	O(*N*)	-	-	O(log *N*)	O(1)	-
`get`/`set`	O(1)	O(*N*)	O(1)*	O(1)*	-	-	O(1)*
`remove`	O(*N*)	O(*N*)	O(1)*	O(1)*	O(log *N*)	O(1)	O(log *N*)*
`size`	O(1)	O(1)	O(1)	O(1)	O(1)	O(1)	O(1)

- * = operation can only be applied to certain element(s) / places

https://slideplayer.com/slide/9739625/

ArrayList

- Java ArrayList is the resizable-array implementation of the List interface.

- The size, isEmpty, get, set, iterator, and list iterator operations run in constant time. The add operation runs in amortized constant time, that is, adding n elements requires O(n) time. All of the other operations run in linear time (roughly speaking). The constant factor is low compared to that for the LinkedList implementation.

HashSet

- Java HashSet is the basic implementation the Set interface that is backed by a HashMap. It makes no guarantees for iteration order of the set and permits the null element.

- This class offers constant time performance for basic operations (add, remove, contains and size), assuming the hash function disperses the elements properly among the buckets.
- We can set the initial capacity and load factor for this collection. The load factor is a measure of how full the hash map is allowed to get before its capacity is automatically increased.

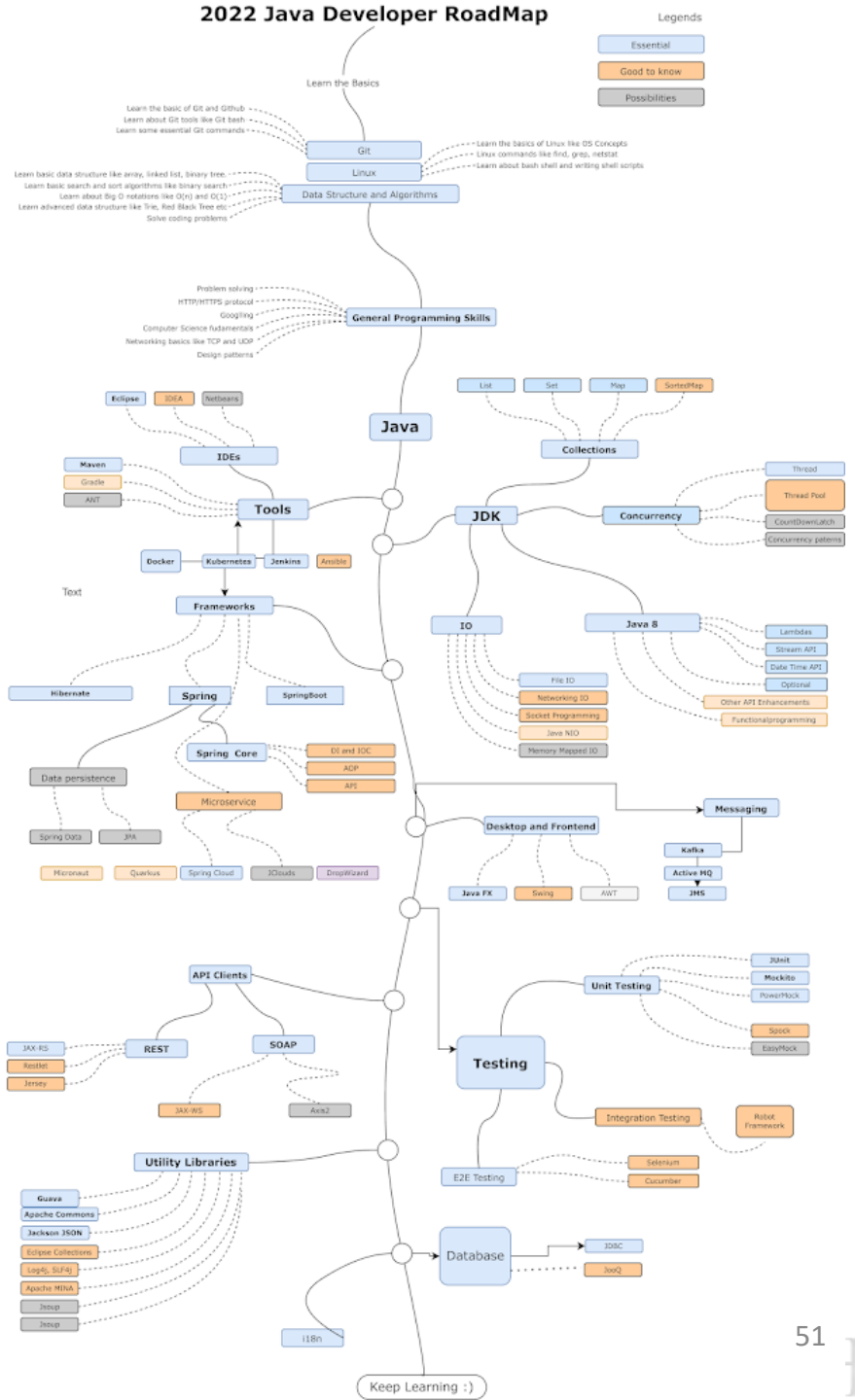
HashMap

- HashMap class is roughly equivalent to Hashtable, except that it is unsynchronized and permits null. This class makes no guarantees for the order of the map. It is implemented on the Map interface.

- This implementation provides constant-time performance for the basic operations (get and put).
- It provides constructors to set initial capacity and load factor for the collection

Summary

- Required for this course
  - General Programming Skills
  - Java Basics
  - OOP concepts
  - Collections (a little)



https://javarevisited.blogspot.com/2019/10/the-java-developer-roadmap.html#axzz81BEe6xrA