**FULL STACK JAVA**

**JDK JRE IDK**

### Java: The Programming Language

<https://howtodoinjava.com/series/java-versions-features/>

* JAVA 8 JAVA 11 JAVA 17 JAVA 12

Run the Java file in command line by >>javac Main.java

>> java Main

Java is compiler programming language so, need to run it explicitly

If the java class created under the package >> java com.package.name.Main

Comments in Java

Line comment>> //int n=0;

Block comment>> /\* Statements \*/

Packages

Naming convention – [use reverse domain name notation] com.lma.learn;

Variables

Final Variables – once it’s final can’t change the value of it.

One byte is equal to 8 bits

Byte 8; short 16; int 32; long 64; float; double

All Arithmetic handled data types(integer), Character, Boolean and floating point types are primitive data types.

Basic Operators: [BODMAS]

Prefix and Postfix Operators

i++; i--; ++i; i--;

Compound Assignment Operators

i -= 3; i +=3; i /=3; i %=3; etc….

Type Conversions

Implicit type conversion >> automatically converted by compiler [only widening happens automatically]

Explicit type conversion >> Done by casting

Conditional Logic Statements and Block Statements

if; else; else if; - conditional ==, <=, >=, >, <. Logical &&, ||, ^, !.

ternary operator >> n==m ? n++ : n-- ;

switch(n){ // only int,short, byte, long, char are supported from primitive types  
case 1: break;

case 2: break;

}

while; do-while; for; for-each;

Array: Collection of elements

Int array[]={1,3,4,5};

Int array;

array =new int[size];

for(int i; array){ // best suite for iteration

System.out.println(i);

}

### Methods

A work done by object can implement here in a class.

By making Method static, methods can call without creating a object.

### Strings

Using StringBuilder strings can be modified. append(), insert()

String methods > equals();concat(); intern() to copy a string

Creating new strings from exiting > concat,replace, toLowerCase, toUpperCase, trim, split

Extract substring: charAt, substring

Test substring: contains, endsWith, startsWith, indexOf, lastIndexOf

Comparison: equals, equalsIgnoreCase, isEmpty, comapreTo, comapreToIgnoreCase

Formatting: format >>int n=1323; S= String.format(“%,d”,n) > S=1,323.

String from non-string; valueOf

**String.format(**format, value**)**

**%02d – if value is fewer than two digits it pads the 0 to the value. Similarly if %.2f – if value float is lessa than the two digits it adds the zero to value on right side**

Date and Time

LocalDate date= LocalDate.Now();

DateTimeFormatter datef= DateTimeFormatter.ofPattern(“MM-dd-YYYY”);

Date.format(datef)

Using Generics, it is possible to create classes that work with different data types.

The **var reserved type name (not a Java keyword)** was introduced in Java 10. Type inference is used in var keyword in which it detects automatically the datatype of a variable based on the surrounding context. The below examples explain where **var**is used and also where you can’t use it.

# **Collections**

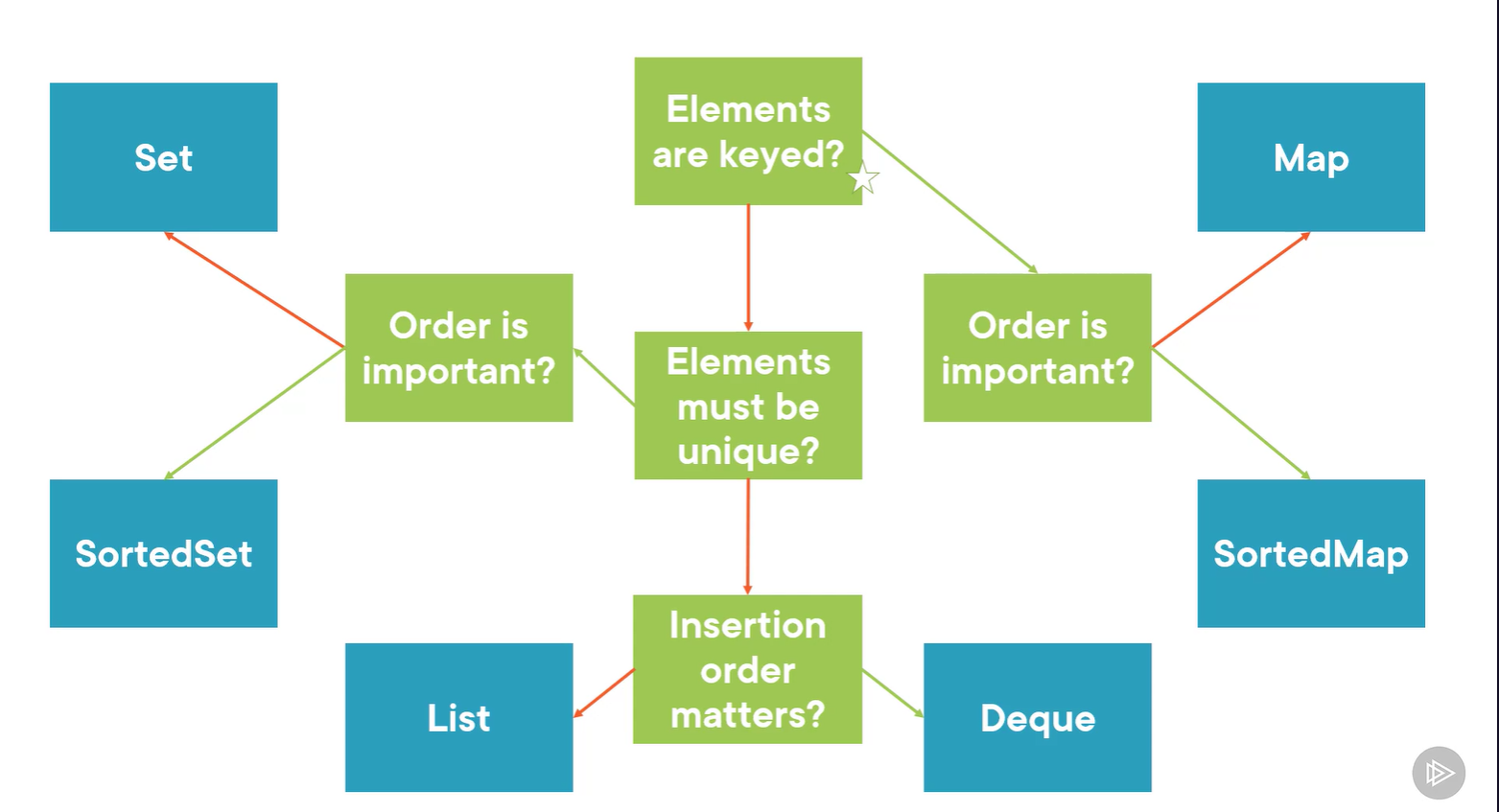
Data Structure are diversified in in nature:

* Ordering - Lists
* Pairs - Maps
* Uniqueness – Sets

Java Streams

Collection Operations and Factories

**List, Set and Queue implements the Collection interface, where Map is separate interface.**

****

When **elements are keyed** – must use **SortedMap**, if not can use **Map**.

When **elements are not keyed** and **not be unique** – can use **List** if **insertion order** is matter must use **Deque**

When elements needs **to be unique** and must be in **order** the must use **SortedSet** and if not can use **Set**

**Common Behaviour of Collection**

**size() – to get size**

**isEmpty() – returns true if collection is empty**

**add(element) – adds an element to collection at specific index**

**addAll(Collection) – appends new collection elements to old collection**

**set(index,element)**

**get(index)- retrieves the target index element**

**remove(element) – removes the element from collection if present**

**removeAll(collection) – remove all elements**

**Col.retainAll(collection) – remove all elements from collection which are not matched with the argument collection**

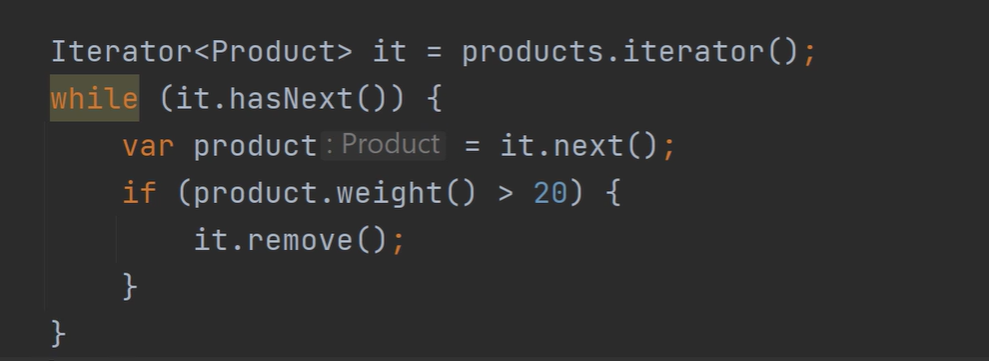
**contains(element) – returns true if element presents in collection**

**conatainsAll(collection) – if all elements contained in source collection returns true.**

**clear()**

**Loop over** - Collection extends to another interface Iterable. It provides the Iterator object.

For Each – cannot remove element form the collection. So alternative approach is iterable.



subList(0,5) – to extract the subset of an element

### Comparator interface

Multiple sorting based on object properties

Comparator provides **compare() method** to sort elements.

**Collections.sort(List, Comparator) or Arrys.sort(array,comparator)**

**More like object comparision**

new Comparator<Integer> (){

                public int compare(Integer a, Integer b){

                    return a-b;

                    }}

### Comparable Interface

Comparable provides **compareTo() method** to sort elements.

We can sort the list elements of Comparable type by **Collections.sort(List)** method.

More like object specific comparesion

## **Arrays**

Arrays can store objects, typically it store homogeneous data and static in size;

toString method is helps to print the array without need of iterating

int[] array = {2,3,4 ,4};

System.out.println(Arrays.toString(array));

# Stream API

JAVA -8

Same as ARRAY Callbacks functions in JAVASCRIPT

<https://howtodoinjava.com/java/stream/java-streams-by-examples/>

java.util.Stream

*There uses of Stream in Java are mentioned below:*

**Designed for**[**lambdas**](https://howtodoinjava.com/java8/lambda-expressions/)**or functional programming**

**Do not support indexed access**

1. ***Stream API is a way to express and process collections of objects.***
2. ***Enable us to perform operations like filtering, mapping,reducing and sorting.***

Stream in JAVA is same like .methods in JS – map, filter, sort, reduce

## Features

* A stream is not a data structure instead it takes input from the Collections, Arrays or I/O channels.
* Streams don’t change the original data structure instead returns the new data structure as per underlined menthod.
* **Stream operations are either intermediate or terminal.**

**1. Create a Stream** => Stream<Integer> stream = Stream.of();

Use below methods to create a stream:

Stream.of() **or** Stream.of(array) **or** List.stream()

Stream<Integer> stream = list.stream();

stream.forEach(p -> System.out.println(p));

Stream.generate() or Stream.iterate()

Stream of String chars or tokens

**2. Stream Collectors**

1. **Collect Stream Elements to a List**

Stream<Integer> stream = list.stream();

List<Integer> evenNumbersList = stream.filter(i -> i%2 == 0)

.collect(Collectors.toList());

1. **Collect Stream Elements to an Array**

Integer[] evenNumbersArr = stream.filter(i -> i%2 == 0).toArray(Integer[]::**new**);

**3. Stream Operations**

**1. Intermediate Operations**

Intermediate operations return the stream itself so you can chain multiple methods calls in a row.

**Stream.filter()**

memberNames.stream().filter((s) -> s.startsWith("A"))

.forEach(System.out::println);

**Stream.map()**

memberNames.stream().filter((s) -> s.startsWith("A"))

.map(String::toUpperCase)

.forEach(System.out::println);

**Stream.sorted()**

memberNames.stream().sorted()

.map(String::toUpperCase)

.forEach(System.out::println);

**2. Terminal Operations**

Terminal operations return a result of a certain type after processing all the stream elements.

**Stream.foreach() –** helps iterate the loop

memberNames.forEach(System.out::println);

**Stream.collect() – used to retrieve elements from stream list**

List<String> memNamesInUppercase = memberNames.stream().sorted()

.map(String::toUpperCase)

.collect(Collectors.toList());

**Stream.match() –** to compare if matched returns results

**boolean** matchedResult = memberNames.stream()

.anyMatch((s) -> s.startsWith("A"));

System.out.println(matchedResult); *//true*

matchedResult = memberNames.stream()

.allMatch((s) -> s.startsWith("A"));

System.out.println(matchedResult); *//false*

matchedResult = memberNames.stream()

.noneMatch((s) -> s.startsWith("A"));

System.out.println(matchedResult); *//false*

**Stream.count() -** The count() is a terminal operation returning the number of elements in the stream as a long value.

**long** totalMatched = memberNames.stream()

.filter((s) -> s.startsWith("A"))

.count();

System.out.println(totalMatched);

**Stream.reduce() -** The reduce() method performs a reduction on the elements of the stream with the given function

**4. Short-circuit operations**

**Stream.anyMatch() and Stream.find()**

**5.Parallel Streaams**

To enable parallelism, all we have to do is to create a parallel stream, instead of a sequential stream.

List<Integer> list = **new** ArrayList<Integer>();

**for**(**int** i = 1; i< 10; i++){

list.add(i);

}

*//Here creating a parallel stream*

Stream<Integer> stream = list.parallelStream();

Integer[] evenNumbersArr = stream.filter(i -> i%2 == 0).toArray(Integer[]::**new**);

System.out.print(evenNumbersArr);

### 7.2 Intermediate Operations

* [filter()](https://howtodoinjava.com/java8/java-stream-filter-example/)
* [map()](https://howtodoinjava.com/java8/stream-map-example/)
* [flatMap()](https://howtodoinjava.com/java8/stream-flatmap-example/)
* [distinct()](https://howtodoinjava.com/java8/java-stream-distinct-examples/)
* [sorted()](https://howtodoinjava.com/java8/stream-sorted-method/)
* [peek()](https://howtodoinjava.com/java8/java-stream-peek-example/)
* [limit()](https://howtodoinjava.com/java8/java-stream-limit-method-example/)
* [skip()](https://howtodoinjava.com/java8/stream-skip-example/)

### 7.3. Terminal Operations

* [forEach()](https://howtodoinjava.com/java8/java-stream-foreach/)
* [forEachOrdered()](https://howtodoinjava.com/java8/java-stream-foreachordered/)
* [toArray()](https://howtodoinjava.com/java8/convert-stream-to-array/)
* reduce()
* collect()
* [min()](https://howtodoinjava.com/java8/java-stream-min/)
* [max()](https://howtodoinjava.com/java8/java-stream-max/)
* [count()](https://howtodoinjava.com/java8/stream-count-elements-example/)
* [anyMatch()](https://howtodoinjava.com/java8/stream-anymatch-example/)
* [allMatch()](https://howtodoinjava.com/java8/stream-allmatch-example/)
* [noneMatch()](https://howtodoinjava.com/java8/stream-nonematch-example/)
* [findFirst()](https://howtodoinjava.com/java8/stream-findfirst-findany/)
* [findAny()](https://howtodoinjava.com/java8/stream-findfirst-findany/)

# Lambda Expressions in JAVA

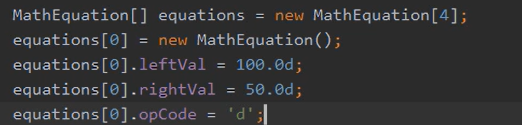
IS ANONYMOUS UNCTIONS to a achieve functional interface type. ()->{};

(1,2)-> return 1+2;

# **Classes And Interfaces**

Fields Constructor Methods

Object[] oops=new Object[4]; >> Aray of Classes

Now using above object we can store multiple object fields here, 

Encapsulation >> implemented by private access specifiers (Accessor > Getter and Mutator>Setter)

This keyword is used to reference a variable belongs to aclass instance

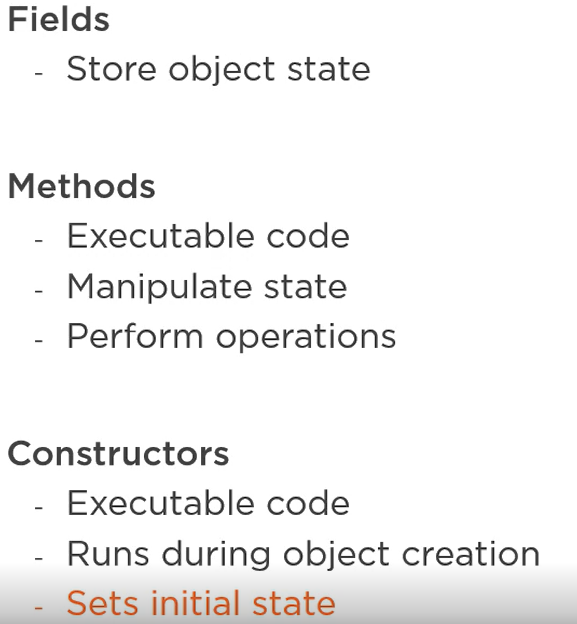
**Access Specifier**:- This can be understood as the access you provide to your code in Java whether other classes can access your code or not.

E.g. public, private, protected and default.

**Access Modifier**:- Java provides both Access Specifier and Access Modifiers for creating access to your Java code for other classes. Here modifier is also used to do the same task but there are limitations.

1. **Class Modifier**:
   * **abstract** :- This defines the restriction such that objects cannot be created.
   * **final**:- This restricts a class from being inherited.
   * **strictfp**:- it is related to the checking of floating point values irrespective of OS.
2. **Variable and Method Modifier**:
   * **static**:no object creation required
   * **final**: cannot be reassigned
   * **transient**: it is not serialized
   * **volatile**: the values are liable for change

Class:



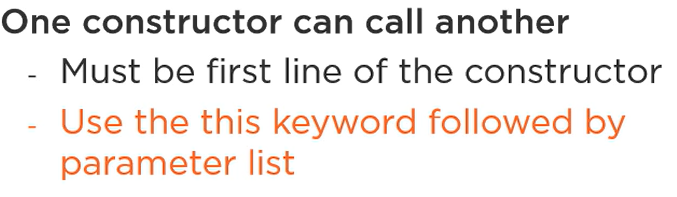
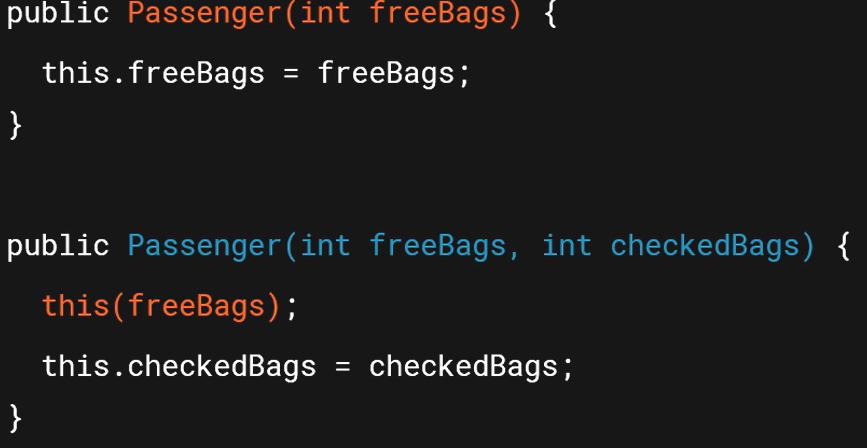
*Class Constructor and Initializers*

Code that runs when object is creating

Implicit and explicit constructors(user defined)

Constructor Chaining >>

This keyword at first line of the second constructor calls the first constructor, this linking process is called Constructor Chaining.

If Constructor visibility set to private, the object of that class can’t create outside of that.

Static variables or methods are not tied any specific class instance can access class wide, can use it outside the class by calling with classname. Can import these static fields.

Static methods access only static members.

Method Overloading, methodName(Parameter….)>> to send multiple arguments

# GUI based applications

To create GUI based applications, JAVA provides javax.swing and java.awt packages:

/\*

\* Task1 : Command Line Application vs GUI (Inversion of Control)

\* Underlying code builds the GUI and responds to events, not determined by

\* the programmer.

\* The programmer tells the framework what to call, but does not invoke the method directly.

\*/

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

import java.util.Scanner;

public class SwingIOC {

private static void createGUI() {

// Create the Window Frame to add components to..

JFrame frame = new JFrame("SwingIOC");

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setLayout(new GridLayout(5, 2));

frame.setBackground(Color.BLUE);

// Add the GUI components

JLabel label = new JLabel("Name:");

JTextField text = new JTextField();

JButton button = new JButton("Submit");

JButton buttonExit = new JButton("Exit");

JLabel labelOut = new JLabel();

frame.getContentPane().add(label);

frame.getContentPane().add(text);

frame.getContentPane().add(button);

frame.getContentPane().add(buttonExit);

frame.getContentPane().add(labelOut);

buttonExit.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

System.exit(0);

}

});

// Display the window.

frame.pack();

frame.setVisible(true);

}

public static void main(String[] args) {

// Scanner sc = new Scanner(System.in);

// System.out.println("Please enter your name: ");

// String name = sc.nextLine();

// System.out.println("you entered "+ name);

// sc.close();

javax.swing.SwingUtilities.invokeLater(new Runnable() {

public void run() {

createGUI();

}

});

}

}

## To make API calls using JAVA:

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.net.HttpURLConnection;

import java.net.MalformedURLException;

import java.net.URL;

 class restAPI {

    public static void main(String[] args) throws MalformedURLException, IOException {

        // create url

        URL = new URL("https://crio-xflix.herokuapp.com/v1/videos/602d228e672f010020e5e95d");

        // Send Get request and fetch data

        HttpURLConnection conn = (HttpURLConnection) url.openConnection();

        conn.setRequestMethod("GET");

        BufferedReader br = new BufferedReader(new InputStreamReader(

            (conn.getInputStream())));

        // Read data line-by-line from buffer & print it out

        String output;

        while ((output = br.readLine()) != null) {

            System.out.println(output);

        }

        conn.disconnect();

    }

}

# Spring Boot Application Development:

* @SpringBootApplication is spring boot application processor. It support all annotations
* SpringApplication.run(mainApplocation.calss,args); is main method for Spring boot applications.
* Inversion of Controller and Dependency Injection are used to implement Loose coupling and runtime object creation by Beans and interfaces
* Model is responsible for navigation and data sharing to different models in application, here Spring MVC come to picture by mapping annotations:

MVC is used to serve the **Controller** > HTTP request by sending **View** > HTTP response by model to model through DispatcherServlet

@Controllers

@Getmappping()

@Requestmapping

@Postmapping

@RequestParam --> it take data from web page

@Controller

**public** **class** HomeController {

@GetMapping(value="/home")

**public** String getHome() { #one model

**return** "Home.jsp";

}

@PostMapping(value="/registration")

**public** String addCustomer(@RequestParam String firstName, @RequestParam String lastName, @RequestParam String email, Model model) { # second model

model.addAttribute("firstName",firstName);

model.addAttribute("lastName", lastName);

model.addAttribute("email", email);

**return** "userdata.jsp"; #this jspm file holds the attribute name to render attribute value

}

}

* The src folders is where we write our web content, from here IDE Spring Boot renders web pages, by auto configuration.

**Spring JPA and Hibernate:**

Spring JPA (JAVA PERSISTANCE API) helps to implement functions related to data controlling from relational data bases , CRUD operations. It provides the h2 database internally.

Hibernate is used to create a connection between database to spring boot application.

<http://localhost:8080/h2-console> where the local database created by JPA

@Autowired