JAVA

**JAVA CLASSES AND FILE NAMING CONVERSION**

A java file can contain any number of classes but only one class can be declared as public. This is because only one class in a file is allowed to be exposed to outside classes in different java files.

And also, if a file contains a public class, the name of the public class should be the name of the file by compulsory.

Eg: A.java

public Class A{

}

// this B public class will generate an error since only one public class is allowed per java file to be exposed outside

public Class B{

}

Class C{

}

Also if you save the above file as justice.java or c.java there will be an error. Since there is a public class “A” which java assumes it to be the file name.

But if a java file contains no public class, you can name the file whatever you like.

So, I can name bellow file as justice.java or books.java or B.java and there won’t be no error since no public class exist in it.

Class B{

}

Class C{

}

Class D{

}

=================================

**How To compile and execute a java file on the console**

To compile and execute a java file, you need to use the command prompt. Imagine I have this file:

Justice.java

class Justice{

    public static void main(String args[]){

       System.out.println("hello word");

    }

}

To compile the above file:

First open your cmd and compile the java file first with this synthess: javac filename.java (javac means => java compiler. It tells java compiler to convert all the code in the above file into machine codes)

So compile the above file as: javac Justice.java

When you type the above, the java compiler will compile all the classes in the file to different files with each single class name (it will save it class as .Class not .java).

Then you need to execute the file with this synthess: java javafilaname.java or java className

So, execute the above as: java Justice.java

The above will execute the main method in the file.

// result: hellow words

=================================

**Main Method**

All classes in a single java file can contain main method.

But When the file is executed, the compiler will not run each main method. But it will rather initialize the main methods in all the classes. You would then need to **run each class separately.**

**Eg. Durga.java**

class A {

public static void main(String[] args) {

System.out.println("A class");

    }

}

class B{

public static void main(String[] args) {

System.out.println("B class");

    }

}

class C{

    public static void main(String[] args) {

    System.out.println("C class");

        }

    }

You can run this on console with:

Javac Durga.java // this will just initialize all the main methods in each class without printing anything to the console

So you then need to call each class in the console with:

Java A // result = A class

Java B // result = B class

Internal Concept of the Main Method

public static void main(String[] args) { }

The main method is invoked by the JVM. And below is what it details mean:

Public: means the main method can be accessed any where

Static: The main method is run by the JVM. The static keyword helps not to create object of the class before invoking it. This saves time. Because jvm will call it direct like A.main(args);

Void: means this method does not return anything.

String [] args: these are command line argument that needs to be passed into the program at run time not when you are firstly compiling it. Eg.

Javac A.java

Java A.java Justice //justice represent the arugment passing into the program

Then grab it in your java program as System.out.println(args[0]);

=================================

Difference between compile Time and Run Time

Before you can run a java program that you have coded already. You need to first:

1. Compile the program with javac filename.java (This is called compile time) Here the java compiler will only convert your code into byte codes without running it.



1. Run the program with java filename.java (This is run time) Here is where your operating system will run your program byte code.



=================================

**JAVA IMPORT PACKAGE CONCEPT**

Java has a lot of already defined classes and interface that does specific things. These classes and interfaces are mainly saved in java.util packages. (Is just like modules in angular and node.js)

So any already defined method by java you can use in your class, you have to import it first before you can get access to all the methods that package class has.

Eg. Main.java

First I need to import the ArrayList package before I can use it.

// Import the arraylist package

import java.util.ArrayList;

public class Main {

    public static void main(String[] args) {

// create an object of above imported Arraylist

  ArrayList al =new ArrayList();

  //use it

  al.add("book");

  System.out.println(al.get(0));

    }

}

So from the above if I omit the import java.util.ArrayList; the compiler will throw an error because it won’t know What an ArrayList is and where it comes from.

So the package java.util.ArrayList contains all the methods and properties of ArrayList. So if you want to see all methods and properites of java arrayList, you just need to check java documentation for java.util.ArrayList.

This applies to every package in java. And it’s also one concept of java (You import packages/modules and access the methods and properties in it)

So for example if you want to perform MySQL dB operation. All the methods and properties are grouped in java.util.mysql package.

Remember, you should also create your own packages and group your classes and interface instead of using folder.

All your package name should be a reverse order of your internet domain. Example successismoney.com should be com.successismoney.

And remember only one package statement is allowed per java file and it should be the first line of code in the file. But if comment is the first line, there will be no error. Eg.

// comment com.justice is the package name of the Main class

package com.justce // this will not give error but if any other code comes first except comment, there will be error

**import java.util.ArrayList;**

public class Main {

    public static void main(String[] args) {

// create an object of above imported Arraylist

  ArrayList al =new ArrayList();

  //use it

  al.add("book");

  System.out.println(al.get(0));

    }

}

=================================

**CLASS LEVEL Access MODIFIES**

Access modifies represent the level of a class or method to it outside files. Is a mechanism to either allow or restrict other class or methods from accessing a specific class.

**DEFAULT** => represent a class with no access modifies. It is...

Package level Restriction === Meaning it is accessible to only the class files that are in the same package only.

**PUBLIC** => It represent a class or method with public access modifier. There is no restriction on it. It is…

Global Level => Meaning there is no restriction on it, it can be accessed anywhere.

**PRIVATE** => It is only accessible in that particular class example like it inner class. And also if a method is defined as private, it is only accessible to that class only. It is…

Class level Restriction === Meaning no other class can access it except only the inner class…

=================================

**Non-Primitive variables Or Object Reference Variables**

Non-primitive variables are variables that have a type of class object and interface object.

That’s every variable that has a data-type of class is an object reference variable.

The Trick is this:

You can't see object in your code, they are not visible. They are stored in the application memory,

So in below code, jvm will create an invisible object of the Dog class and it properties and methods in the application memory. And the address to this memory location is "d"

So "d" is a reference to the Dog object in the application memory. That's why it called (object reference variable or reference variable)

Dog d = new Dog();

REference Variable: Declaration, create, and assignment

1. Dog d; (tell the jvm to allocate space for the d variable)

2. new Dog(); (tell the jvm to allocate space for the Dog object in heap memeory)

3. Dog d = new Dog(); (assign the Dog object memory address location to the "d" variable)

There is no such thing as object variable.

There is only an object reference variable. The "d" variable below is an object reference variable.

It does not contain or hold the properties and methods in the Dog class itself. But rather it a pointer or address to access all the properties and methods in the "Dog" Object.

public class Main {

    public static void main(String[] args){

       Dog d = new Dog(); // d is an object reference variable that holds the addres of the Dog object in your application memory

  }

}

// Dog class

class Dog{

  void bark(){System.out.println("Dog is barking");}

}

The difference is this: The below "age" primitive variable holds a value of 20. But what is the value of the above "d" reference variable?

It value is the address of the "Dog" object in the application memory EG. Myapplication.Dog@3fee733d. That's because each class object has a different address in your application memory.

So the "d" reference variable is just a way to get to the Dog object in the application memory.

int age = 20;

So if am to print the above variables it will be:

System.out.println(age); //result will be 20

System.out.println(d); // This will print the address of the Dog object in the application memory eg. Myapplication.Dog@3fee733d

We use the dot (.) operator on the object reference variable to access the properties and the methods in that class object.

For example below means execute the "bark" method in the Dog object.

d.bark();

=================================

**Singleton pattern**

Singleton pattern helps you to have one instance of the class in your entire app instead of allowing other classes to create separate object of it.

You need 3 things to make it work:

1. Private static instance variable of the class
2. Private constructor of the class
3. Public static method with the return-type of the class

**Example:**

// inside School.java

public class School {

    private static School schoolInstance;

    // make it private so that other class can't create object of this class

    private School() {

        System.out.println("today is a good day");

    }

    // the only way to get the class constructor to execute is to call this method

    public static School getSchool() {

        // check if the schoolInstance is null and assign the class instance to it

        if (schoolInstance == null) {

            // because you have created, object of the class here, it will call the

            // constructor

            schoolInstance = new School();

        }

        // return the schoolInstance

        return schoolInstance;

    }

}

// inside Main.java

class Main {

    public static void main(String[] args) {

        // just call the public static method that returns the class Instance because you can’t create the object of the class to access it

        School.getSchool();

    }

    /\*

      RESULTS: today is a good day

     \*/

}

=================================

**The concept of garbage collection**

When a memory is allocated to an object of a class and later the address get removed, it's called garbage collection.

public class Main {

    public static void main(String[] args){

       Book a = new Book();    // Book object 1

       Book b = new Book();   // Book object 2

       Book c = a; // c holds the same address as “a” (object 1 above) to the Book object 1 in the application memory

       /\*

        below "b" reference object of Book object 2 now has no address. Meaning a space has

        been allocated to the  Book object 2 but there is no reference object to access it.

        This is called "cabbage collection"

        in short, This is becuase a memeory was allocated for book object 2, with an address of "b".

        But the address has been removed when set b = null below

        So now Book object 2 is garbage collected. And if you get more of it let say 10 object with no reference address,

        it will decrease the performance of your application.

        Your application will take more memory from the user system, but one small portion of the memory will be used, the rest will be used to

store garbage object.

        but "b" is still a reference variable that can be assigned b = new Book(); or b = a

        \*/

       b=null;

    }

}

// the book class

class Book{

}

=================================

**ABSTRACT CLASS AND METHODS**

Abstract class is a partial or incomplete implemented class. This is because it a class that some of it methods are not complete.

Example is a method that has only declaration but no body

You can't create object of abstract class because it members are incomplete. And if a class create an object to access it incomplete

methods, then what result will the abstract method give?

A class can be declared abstract to prevent object creation even if the class doesn't contain any abstract method.

You can create both abstract and non-abstract method in an abstract class

A class can also be defined as abstract without any abstract method to prevent object creation of the class.

If a class contains even one abstract method, the class should be declared abstract.

Abstract method is a method that contains only declaration but no implementation. (it declared with no curry braces )

Note == You can’t create an object of abstract class. You can only inherit it. And if you inherit it, you must provide body declaration to all it abstract methods without leaving a single one.

Example 1.

Inside ageExcemption.java

// Abstract class. You can not create object of it. You can only inherit it in different class

public abstract class ageException {

  // abstract method does not contain body. It body is provided by the class that inherit it

  public abstract String getFruitTaste();

}

Then inside Main.java extend the above abstract class and provide body implementation for its abstract method.

// inherit the above abstract calss

public class Main extends ageException {

  public static void main(String[] args) {

  }

  // implement it abstract method

  public String getFruitTaste() {

    return "the taste of orange is salty";

  }

}

**Example 2.**

Abstract class allow other class to perform different operation with it abstract methods…

// inside School.java

public abstract class School {

  abstract void getTotals(int num1, int numb2);

}

// inside Main.java

public class Main {

    public static void main(String[] args) {

        Friend1 friend1 = new Friend1();

        friend1.getTotals(5, 5);

        // Result: 10

        Friend2 friend2 = new Friend2();

        friend2.getTotals(5, 5);

        // Result: 25

    }

}

// create a class and extends the abstract class do something different with it

// method

class Friend1 extends School {

    @Override

    void getTotals(int num1, int numb2) {

        // here I just want to add the two numbers

        System.out.print(num1 + numb2);

    }

}

// create a class and extends the abstract class do something different with it

// method

class Friend2 extends School {

    @Override

    void getTotals(int num1, int numb2) {

        // Here I want to multiply the two numbers

        System.out.print(num1 \* numb2);

    }

    /\*So from the above you can see that Friend1 and Friend2 class are doing different things with the same implemented abstract class

"getTotals()" method. one is performing addition and the other is performing multiplication operation.

 That’s the main purpose of abstract class:

 1. to prevent object creation of a class

 2. to allow different classes to perform different implementation to it abstract methods.

    \*/

}

=================================

**Iterface**

Interface is just like an abstract class, the difference is that, whiles abstract class can have non-abstract methods, all methods in interface are abstract.

You can’t create an object of interface, it needs to be implemented.

It provide full data abstraction (hiding some data and showing only relevant once to it users)

When you create an interface, the java compiler by default adds public static final to all interface variables. And it adds public abstract to all interface methods.

So when you create a variable or method in interface, the jvm will add those to it by default. Example:



Note:

Class extends class

Class implements interface

Interface extends interface



Interface is also termed as any service requirement specification. Interface will give you the requirement but you have to decide what you are going to use that requirement for. Example. Your father will give you money (father = interface, money = requirement) But either you will buy food, water, cloth or fish with that money, is up to you.

* Note = if a class implements an interface method, it should not forget about it access modifier (private, public, protected) it must be provided.
* Note = When a class implements an interface, it needs to provide implementation to all the interface methods without living a single one. If the class does not want to provide implementation for all the interface methods, then the class itself should be declared abstract. But if another class extends this Abstract class (chain class), then it needs to provide implantation for the remaining interface methods.

**Example:**

// Interface

interface Animal {

    public void animalSound(); // interface method (does not have a body)

    public void sleep(); // interface method (does not have a body)

  }

  // Pig "implements" the Animal interface

  class Pig implements Animal {

    public void animalSound() {

      // The body of animalSound() is provided here

      System.out.println("The pig says: wee wee");

    }

    public void sleep() {

      // The body of sleep() is provided here

      System.out.println("Zzz");

    }

  }

  class Main {

    public static void main(String[] args) {

      Pig myPig = new Pig();  // Create a Pig object

      myPig.animalSound();

      myPig.sleep();

    }

  }

HOW A THIRD PERSON CAN USE INTEFACE

// inside DrawingInterface.java

interface DrawingInterFace {

  // public static final

  String myName = "justice";

  // public abstract method by default

  void drawSomething();

}

// inside Main.java

public class Main {

    public static void main(String[] args) {

        // use the CircleClass class to create and object of the interface

        DrawingInterFace drawingInterFace = new CircleClass(); // THIRD PERSWON USING THE INTERFACE

        drawingInterFace.drawSomething();  // result; drawing Circle

        System.out.println(drawingInterFace.myName);  // result: justice

        // this will be error, the "DrawingInterFace" doesn't have "circleColor" variable

        System.out.println(drawingInterFace.circleColor);

    }

}

// create a class that implements the interface and implementation for it methods

class RectangleClass implements DrawingInterFace {

    // provide implementation for the interface method

    @Override

    public void drawSomething() {

        System.out.println("drawing rectangle");

    }

}

// create a class that implements the interface and implementation for it methods

class CircleClass implements DrawingInterFace {

    String circleColor = "Red";

// provide implementation for the interface method

    @Override

    public void drawSomething() {

        System.out.println("drawing Circle");

    }

}

Multiple Inheritance

Multiple inheritance in java through class is not possible but it can be achieved with interface.

When a class implements multiple interface, its multiple inheritance.

When an interface extends multiple interface, its multiple inheritance.



**Example:**

interface Man1 {

    void printman1();

}

interface Man2 {

    void printman2();

}

// multiple inheritance

class person implements Man1, Man2{

    @Override

    public void printman1() {

        System.out.println("man 1");

    }

    @Override

    public void printman2() {

        System.out.println("man 2");

    }

}

// Multiple inheritance

// the class who will implement this interface must provide implementation for all the exteded interfaces

interface Girl extends Man1, Man2{

}

=================================

## Encapsulation

Encpsulation is a way of making sure that some important datas are hidden from the user.

To Achieve Encapsolulation, you must:

1. Declare class variables as private
2. You must provide public set and get method to access and update values of private variables

The key is to make your class variables private so that when someone create an object of the class he can’t access it with the object. You only have to provide public set and get method to access the private variables.

**Example:**

class Person {

    // private variable access only in this class

    private String firstName;

    // setter for setting value for the "firstName"

    public void setfirstName(String firstName) {

        this.firstName = firstName;

    }

    // getter for getting the value of "firstName"

    public String getFirstName() {

        return this.firstName;

    }

}

class Main {

    public static void main(String[] args) {

        // create object of the class

        Person person = new Person();

        System.out.println( person.firstName); // error, can't access private variable in outside class

       // set value for the private variable

        person.setfirstName("justice");

        // get the value of the private variable

       System.out.println(person.getFirstName());

       // result: justice

    }

}

=================================

# Method Signature

Method signature in java represent, the method-Name followed by all its Parameters Datatypes.

Method signature is used by compilers for deciding the right matching method in a particular class to execute (known as method resolution). Eg.

public class Main {

    public int addition(int num1, int num2) {

        return num1 + num2;

    }

    // The method signature is: addition(int, int)

    public void printFirstName(String firstName) {

        System.out.println("your first Name is: " + firstName);

    }

    // The method signature is: printFirstName(String)

}

/\*  So now, if you create an object of the above class and you call one of it method,

Main main = new Main();

main.printFirstName("Justice")

The compiler will look for the method with signature "printFirstName(String)" in the "Main" class and execute it. If it doesn't find it, then it will throw a compile time error

\*/

Note = Two or more methods with the same signature In the same class is not allowed. Eg.

public class Main {

    public void printFirstName(String firstName) {

        System.out.println("your first Name is: " + firstName);

    }

    // The method signature is: printFirstName(String)

    public void printFirstName(String lastName) {

        System.out.println("your first Name is: " + firstName);

    }

    // The method signature is: printFirstName(String)

}

/\*  You can see that the method name in the above two methods are all the same. And they also have one single String parameter.

    Meaning they all have the same method signature: "printFirstName(String)"

The compiler will throw an error if you try to call any of the above method. since they have the same signature, the compiler will be confused note knowing which one to call

Main main = new Main();

main.printFirstName("Justice")

\*/

=================================

# Polymorphism

Polymorphism is a way of doing the same thing in different ways. It mainly occurs when a class has the same method name with it extended class.

Polymorphism can be achieved in two ways.

1. Method Overriding => The sub class has to has to override it extended class method.
2. Method overloading => This is achieved by changing number of method parameters or by change method parameters data-type

**Example 1:** Method overriding

class Parentclass {

    public void dancing() {

        System.out.println("Parent is dancing");

    }

}

class child extends Parentclass {

    // the extended class also has dancing() method, so Override it so that this will be the method to be called

    @Override

    public void dancing() {

        System.out.println("child is dancing");

    }

}

class Main {

    public static void main(String[] args) {

        child child = new child();

        child.dancing();

        // resuts: child is dancing

    }

}

**Example 2:** Method Overloading

// ===== Method overloading by changing method parameters data-types ====

class Parentclass {

    public void operation(int num1, int num2) {

        System.out.println(num1 + num2);

    }

}

class child extends Parentclass {

    // the extended class also has operation() method, so Overload by changing the

    // parameters data-type it so that this

    // will be the method to be called

    public void operation(double num1, double num2) {

        System.out.println(num1 + num2);

    }

}

class Main {

    public static void main(String[] args) {

        child child = new child();

        // this will call the parent operation() method becuase the dataType is int

        child.operation(2, 2); //result: 4

           // this will call the child operation() method becuase the dataType is double

        child.operation(2.3, 2.4);   // resuts: 4.699999999999999

    }

}

// ===== Method overloading by changing number of method parameters ====

class Parentclass {

    public void operation(int num1, int num2) {

        System.out.println(num1 + num2);

    }

}

class child extends Parentclass {

    // the extended class also has operation() method that takes only 2 parameters, so change the number of parameters for this method

    public void operation(int num1, int num2, int num3) {

        System.out.println(num1 + num2);

    }

}

class Main {

    public static void main(String[] args) {

        child child = new child();

        // this will call the parent operation() method becuase it has 2 parameters

        child.operation(2, 2); //result: 4

           // this will call the child operation() method becuase it has 3 parameters

        child.operation(2, 2, 2);   // resuts: 6

    }

}

=================================

GENERICS

Generics helps you to use different data-types to do the same thing. For example, You can define a single method that will print a list of String, int, double and Object values.

Format:::

The HashMap<K, V> “K” and “V” are not already defined data-types. This allows you to define your own data-types when creating a HashMap, and that’s the whole concept of generics.

Eg.

HashMap<String, String> HashMap = new HashMap<>();

HashMap<int, int> hashMap1 = new HashMap<>();

You can see from the above that I have been able to create 2 different HashMap with different data-types. That’s because, the HashMap class is generic.

Note 🡺 All your custom generics can’t take primitive datatypes like int, double. They take Object reference like Integer, Double. You can check online for more of it.

Generic Class

Generic class helps you to pass different data-types of parameters to the class for further operation. Example:

// the "<T>" represent the generic type. it means I don't want to be strict, allow classes that will create object of me to come up with their

own dataTypes

class Books<T> {

    private T bookName;

    public void setBookName(T t) {

        this.bookName = t;

    }

    public T getBookName() {

        return this.bookName;

    }

}

class Main {

    public static void main(String[] args) {

        // create object of the class and pass in a data-Type of “String” to the generic block based

        // on the your data

        Books<String> book1 = new Books<String>();

        book1.setBookName("Aki ola");

        System.out.println(book1.getBookName());

        // result: aki ola

        // pass in Integer if the setBookname you want to pass is of type int

        Books<Integer> book2 = new Books<Integer>();

        book2.setBookName(5);

        System.out.println(book2.getBookName());

        // result: 5

        HashMap<String, String> hashMap = new HashMap<>();

    }

}

Generic Method

class Books {

    // Generic Method

    // <T> makes this method generic. "T" is the return type of it

    public <T> T getSomething(T t) {

        return t;

    }

class Main {

    public static void main(String[] args) {

      // create object of the class

        Books book1 = new Books();

        // because the method is generic, You can pass any value of whatever data-type into it

        System.out.println(book1.getSomething("justice"));

        System.out.println( book1.getSomething(44));

        System.out.println(book1.getSomething(5.5));

        /\*result:

        justice

        44

        5.5

        \*/

    }

}

}

=================================

**How** To Cast Int to String

Use Integer.toString(Number\_To\_Convert\_To\_String) to convert int values to string;

int data = 12;

String string =  Integer.*toString*(data);

How To Check The Data-Type Of A Property

Use property\_Name.getClass().getName() to check the data-type of any property: Eg

System.*out*.println("the date type is: " + data.getClass().getName()); // Result= the data type is: java.lang.Integer

=================================

**NOTE: STRING, INT, Interface HASHMAP, CLASS, OBJECT, ARRAYS, COLLECTIONS, LIST**

See, string, int, collections, class, list, arraylist, hashmap, object and array. They are all ways of storing values in memory in different format. Int store only numbers, haspmap store key value pair values, array store set of values, class store properties, methods and inner class. The same is true with interface.

So when you create an object of a class, you are actually creating a variable that will store the methods, properties and other inner classes of that particular class in the application memory. Example:

User user\_obj = new User();

So above “user\_obj” is just a variable that will store the properties, inner classes and methods of User class.

So whenever you are using any of the above package, you are just creating a variable that will store different data types.

So they are just variable data-types. Data-type is just ways of storing values in different format, that’s either to save it as a key value pair, string, list, or int.

=================================

**Java File IO**

In every project, all the variables values are available during when the program is running only. When the project or app is terminated, all your variables values stored in memory will be lost.

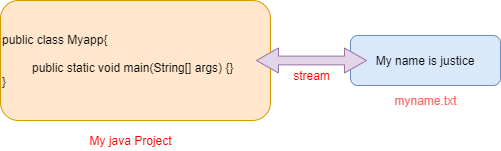
So you need to figure out ways to store some important values permanently, you can choose to use database or store your values into text-files or deal with images and videos.

That’s where the java IO comes in to help you write some text to a file or read a text from a file. Not only files, Java IO can help you to write and read data from a keyboard, camera, pdf, image, monitor, videos and many more.

Stream

You have your project source code, you also have some text-files on your computer hard-disk. You need a connection between your project and the text-files on your computer hard-disk before you can read or write some text onto these files using your project. That is where streams comes in. Stream is the connection between your project and files you want to interact with.

(So if you want to read or write to a file, you need a connection and the connection is called STREAM)



Remember => you cannot see a stream. Because for example: when you ON your phone Wi-Fi, you will see all list of available Wi-Fe’s you can connect to. But you cannot see that connection that helps you to see all these available Wife’s on your phone. So stream is a connection that you cannot see and because of that it is called logical connection.

Types of Stream

You can’t use the same stream for both reading and writing to file, no. There is a separate stream for doing these two jobs. Output and input stream.

Output Stream 🡺 send/write value from your javaApplication to a text file.

Any Data going out of your java Program is called output.

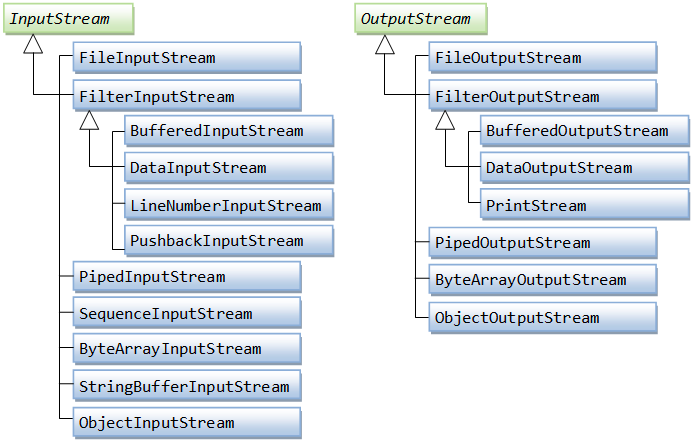
Input stream 🡺 fetch/read value from a text file to your java application.

Any Data coming into your java Program is called Input.

NOTE =Your view should be on your java Project not the text-file if you want to understand the meaning of the above two types.

Sub classes of Input and Output Stream

Input and output stream are both abstract class and they have below sub classes which are meant for different purpose. So if you are working with files, then you should use fileInputstream and fieloutputstream.



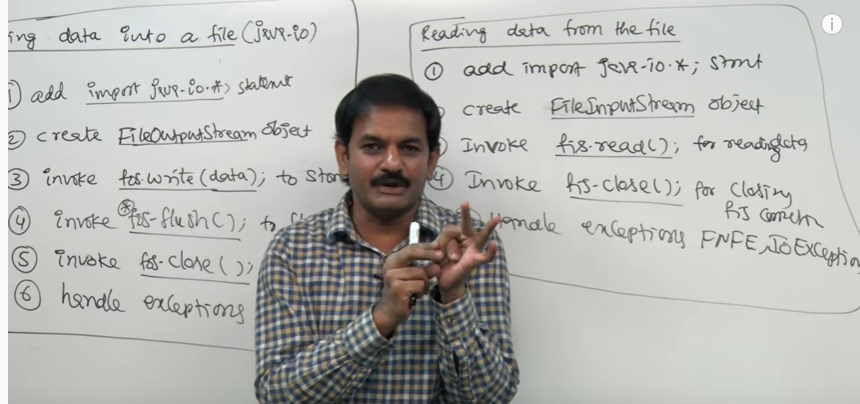
Stream Data-types for Reading and Writing to files

Whenever you want to use stream to read or write to a file, we use these data-types:

Binary called byte (0 and 1) 🡺 For reading and writing to a file in a binary format. (mainly int values)

Character called Char (alphabet eg. “A”, “B”) For reading and writing to a file in a character format. (mainly String values)

Steps to read and write file using fileOutputstream and fileinputstream



fileOutputstream

Remember 🡺 every SINGLE string character is equal to some bytes, including space. Example “A” has a byte of 65. “ ” has a byte of 40. And all bytes are int Types

So always if you want to read or write a file, You have to either convert:

From byte/ **byte**[] to string/char => when reading

String/char to byte/ **byte**[] => when writing

Example: Writing to a file using fileOutputstream

public static void main(String[] args) {

          try{

 // if the file "D:\\testout.txt" is not available, it will create it (connection)

                 FileOutputStream fout=new FileOutputStream("D:\\testout.txt");

                 String s="Welcome to Justice";

                 byte[] b=s.getBytes();//Get the bytes value of above string and convert it into byte[] array

                 fout.write(b);   //write() write the byte of above string to a file

                 fout.close();    // close the connection

                 System.out.println("success...");

                }catch(Exception e){

                    System.out.println(e);  // print the error

          }

 }

fileInputestream

Remember 🡺 fileInputestream can read only one byte at a time. That’s when you have the world “justice” it will read only the bytes of the first alphabets “J” and leave the rest.

So if you want to read all the text in a file or all the alphabet “justice”, you need to read each single bytes including spaces by using a loop. Mainly While loop.

Also remember.

When there is text/data in the file, fileInputestream.read() will return the byte of the first alphabet in the file.

When there is no text/data in the file, fileInputestream.read() will return -1.

Example:

public static void main(String[] args) {

    // the file may not be found, thats why it surrounded in try catch

         try{

               // create fileinputstream connection for reading data

                FileInputStream fin=new FileInputStream("D:\\testout.txt");

                int i=0;  //set variable for looping through each single byte

                /\*

                 When there is no text in the above file fileinputStream.read() will return -1.

                 So read each single char/byte and assign it to "i" untill it reaches the -1

                 (-1 which represent the end of the file)

                 \*/

                while((i=fin.read())!=-1){

                 System.out.print((char)i);  //convert each byte to a char and display it.

                }

                fin.close();// close the fileinputstream connection to free memory

              }catch(Exception e){

                  System.out.println(e);

                  }

}

// the text in the file "D:\\testout.txt" is "Welcome to Justice"

// result==> Welcome to Justice

=================================

How to Create A Method That Accepts Any Number Of Arguments

You can annotate the 3 dot “…” to a method parameter datatype. This will mark such method to accept any number of arguments of the same datatype

public class Main {

  // "..." means “I” should accept any number of int arguments

 public void name(int... i) {

   // Loop through the above data as an array

   for(int b=0; b<=i.length; b++ ) {

        System.out.println(i[b]);

   }

   System.out.println(i[2]);

 }

 public static void main(String args[]) {

  // pass different argument length to it

    //result: 4, 5

  Main main = new Main();

  main.name(4,5);

  // pass different argument length to it

  //result: 4, 5, 3

  Main main2 = new Main();

  main2.name(4,5,3);

 }

}

=================================

GSON package/library

Gson is a package/library for working with json-data and java object/class.

In All programming Languages what we do is that:

1. When we want to post data to server: We convert a Java-Model-class to json and send it to the server. This process is Known as Serialization or Encoding.
2. When we want to fetch data from the server: We convert the json-data from the server to a Java-Model-class. This process is known as Deserialization or Decoding.

**Note:::** The Gson class, has a method called toJson() and fromJson() for performing Serialization and Deserialization.

**Example:**

First Create a maven or gladle project and add the Gson package.

implementation 'com.google.code.gson:gson:2.8.9'

Inside UserModel.java

// Create a custom Model class (this should be in a seperate file)

public class UserModel {

    public String firstName;

    public String lastName;

    public int age;

    // create a constructor to initialize the properties

    public UserModel(String firstName, String lastName, int age) {

        this.firstName = firstName;

        this.lastName = lastName;

        this.age = age;

    }

    // you can Override the toString method here if you want

}

Inside Main.java

public class Main {

    public  static  void main(String[] args){

        //First Create a new object of Gson

        Gson gson = new Gson();

         // ======SERIALIZATION/ENCODING

        // Create object of <UserModel> class

        UserModel userModel1 = new UserModel("Justice", "Ankomah",68);

        // Convert the above <userModel> object to json (serialization)

        //The Gson class has a method called toJson(object) which takes in any object/class and convert it to json

        // the return type is a String, (use the "var" if you want)

        String userModelToJson =  gson.toJson(userModel1);

        System.out.println(userModelToJson);

        // result: {"firstName":"Justice","lastName":"Ankomah","age":68}

        // Now you can Post the above <userModelToJson> to the server

        // ======DESERIALIZATION/DECODING

        // userModelToJson is a json String which looks like this

        // userModelToJson = {"firstName":"Justice","lastName":"Ankomah","age":68}

        // in real work: above userModelToJson will be a json from the server

        // The Gson class has a method called fromJson(jsonString\_here, Model\_class\_here) which takes in the

        // json and the Model\_class

        // desirilize or decode the jsonString to Java class

        UserModel userModel2 =  gson.fromJson(userModelToJson, UserModel.class);

        System.out.println(userModel2.firstName);

        // result: Justice

    }

}

=================================

How To Make Http Request with Retrofit With Clean Architecture?

Retrofit is a nice package for making http request with java. The logic is the same as you would implements in android native development.

To make a request with Retrofit you need 4 things.

1. An Instance of retrofit
2. An interface to define all the endpoint of your request
3. A service class to make different http calls
4. A model class to serialize/deserialize your server json data

**Example:**

**Create a new gladle project and install these packages**

dependencies {

    implementation 'com.google.code.gson:gson:2.8.9'

    implementation group: 'com.squareup.retrofit2', name: 'retrofit', version: '2.9.0'

    implementation group: 'com.squareup.retrofit2', name: 'converter-gson', version: '2.9.0'

}

// Inside ArticleModel.Java

// Create an articleModel class

public class ArticleModel {

    // @SerializedName("author") means use the key in the anotation and not

    // the below property <author> when doing encoding/decoding

    @SerializedName("author")

    // Allow Gson to encode/serialize & decode/deserialize this <firstName> property

    // If you avaoid encode or decode, you can use:  @Expose(serialize = false, deserialize = false)

    @Expose

    private String author;

    @SerializedName("title")

    @Expose

    private String title;

    @SerializedName("description")

    @Expose

    private String description;

    @SerializedName("urlToImage")

    @Expose

    private String urlToImage;

    public ArticleModel(String author, String title, String description, String urlToImage) {

        this.author = author;

        this.title = title;

        this.description = description;

        this.urlToImage = urlToImage;

    }

// Override the toString method

    @Override

    public String toString() {

        return "ArticleModel{" +

                "author='" + author + '\'' +

                ", title='" + title + '\'' +

                ", description='" + description + '\'' +

                ", urlToImage='" + urlToImage + '\'' +

                '}';

    }

}

// Inside MyretrofitInstance.java

// Create A Custom Class with a method that returns instance of retrofit

public class MyretrofitInstance {

    // create a string var that will represent the base url of all your network request

    public static String baseUrl= " https://newsapi.org/v2/";

    // create a var of type retrofit but make it null

    public static Retrofit retrofit = null;

    // Create a Method that returns an Instance of Retrofit

    public static Retrofit getRofitInstance(){

        if (retrofit==null){

            // You will need  implementation group: 'com.squareup.retrofit2', name: 'converter-gson', version: '2.9.0'

            // package for GsonConverterFactory

            // it means use Gson with Retrofit to perform serialization & Deserialization

            retrofit = new Retrofit.Builder().baseUrl(baseUrl).

                    addConverterFactory(GsonConverterFactory.create()).build();

        }

        return  retrofit;

    }

}

// Inside MyApiEndpoint.java

// Inside this interface is where you define the list of each Request url & and the method type thus (get,post,put, delete)

// Actually, this is just a DAO (data access object), you can google java DAO for the meaning

public interface MyApiEndpoint {

    String rUrl = "everything?q=tesla&from=2022-01-12&sortBy=publishedAt&apiKey=9a284a31566545e3bd747b33f1cf4b3f";

     @GET(rUrl)

     Call<HashMap<String, Object>> getArticles();

}

// Inside Service.java

// This is where you make different request to the server with the

// retrofit instance and the api endpoint interface methods

public class Service {

    // Use the instance of Retrofit, and link it to create an object of all your Api request/endpoint are

    static MyApiEndpoint myApiInterface = MyretrofitInstance.getRofitInstance().create(MyApiEndpoint.class);

    //Service Method to Fetch List Of Articles

   static List<ArticleModel> getAllArticles(){

        // Create a list of <ArticleModel> that will be assigned value and return when the server response successfully

        List<ArticleModel> allArticles = new ArrayList<>();

        //Try to make a get request to the server

        Call<HashMap<String, Object>> getArticles = myApiInterface.getArticles();

        // the call the engueue method and implements it onResponse & onFailure method

        // to check if th above getArticles request is successful or fails

        getArticles.enqueue(new Callback<HashMap<String, Object>>() {

            // If request successfully

            @Override

            public void onResponse(Call<HashMap<String, Object>> call, Response<HashMap<String, Object>> response) {

                // response object

                // System.out.println("response object is:"+ response.toString());

                // response body

                //System.out.println("Response body is:" + response.body().toString() );

                // Get the list of Articles object

                Object serverData = response.body().get("articles");

                // convert it to a list

                List articles = (ArrayList) serverData;

                // Loop through Each list and execute this the code in this block on it

                for (Object bb: articles){

                    //Create a new object of Gson

                    // it just a package for working with json

                    Gson gson = new Gson();

                    // Convert each single article toJsonString

                    String tojson = gson.toJson(bb);

                    // Decode/desirialise this articles to the ArticleModel class

                    // So all you are doing is creating an object of articleModel with the json data from the server

                    ArticleModel articleModel = gson.fromJson((String) bb, ArticleModel.class);

                    // Add the ArticleModel to this list

                    allArticles.add(articleModel);

                }

                System.out.println("All articles is:" + allArticles.toString());

            }

            // If request unsuccessful

            @Override

            public void onFailure(Call<HashMap<String, Object>> call, Throwable t) {

                // Print the error

                System.out.println(t.getMessage());

            }

        });

        // return the list of articles from the server to the calling code

        return allArticles;

    }

}

// Inside main.java

public class Main {

    public  static  void main(String[] args){

        // Call the service method to fetch all articles

        Service.getAllArticles();

    }

}

=================================

**RXJAVA**

Rxjava is a way of making asynchronous operation.

It’s all about two key components: **Observable** and **Observer**. In addition to these, there are other things like **Schedulers**, **Operators** and **Subscription**.

*Observable:* Observable is a data stream that do some work and emits data. An Observable emits items or sends notifications to its observers by calling the observers’ methods.

*Observer:* Observer is the counter part of Observable. It receives the data emitted by Observable.

*Subscription:* The bonding between Observable and Observer is called as Subscription. There can be multiple Observers subscribed to a single Observable.

*Operator / Transformation:* Operators modifies the data emitted by Observable before an observer receives them.

*Schedulers:* Schedulers decides the thread on which Observable should emit the data and on which Observer should receives the data i.e background thread, main thread etc.,

**Example:**

public class Main {

    public static void main(String[] args) {

            // ============ USING OBSERVABLE.JUST()

                // Create an observable that returns  Observable<String>

                Observable<String> BooksObservable = Observable.just(

                    // This method returns "Aki Ola"

                    // So this observable will just emit it

                    // so at the end it will be

                    // Observable<Aki Ola> BooksObservable();

                    // Or you can simply create the whole observable as

                    // Observable<String> BooksObservable = Observable.just("aki ola", "english", "flamingo");

                    getbooks()

            )

                    // Map is one of the operators used to modify items

                    // all operators returns an Observable

                    // it will modify each single emitted value by the Observable before returning it to the observer

                    // It just like how the map() function works in javaScript and dart list

                    // it will simple execute the code in below apply(String s) on the emited value

                    // emited by observables before getting to the observer

                    .map(new Function<String, String>() {

                        // s below represesent "Aki Ola" from the getbooks() method

                @Override

                public String apply(String s) throws Throwable {

                    // Add "justice" and return it

                    // so it will be:  Observable<Aki Ola justice> BooksObservable();

                    return s + " justice";

                }

            });

            // Then subscribe to the above observer and get the data

            BooksObservable

    //                .observeOn(Schedulers.newThread())

    //                .subscribeOn(Schedulers.io())

    //                .subscribeOn(Schedulers.io())

                    .subscribe(new Observer<String>() {

                        @Override

                        public void onSubscribe(@NonNull Disposable d) {

                            // you can add a Displosable here so that you can unsubscribe when needed

                            System.out.println("subsribed");

                        }

                        @Override

                        public void onNext(@NonNull String s) {

                            System.out.println("onNext: " + s);

                        }

                        @Override

                        public void onError(@NonNull Throwable e) {

                        }

                        @Override

                        public void onComplete() {

                        }

                    });

    }

    // Create a method that returns a String

    static String getbooks() {

        return "Aki Ola";

    }

}

**Note= you can find the whole documentation from the official source here::** [**https://reactivex.io/languages.html**](https://reactivex.io/languages.html)

**For Java only = http://reactivex.io/RxJava/3.x/javadoc/overview-summary.html**

**You should easily use the documentation above if you encounter any bugs because it really easy.**

=================================

**RECOMMENDED THINGS BY TOP JAVA PROGRAMMERS**

Use package to group your classes and interfaces instead of folder. And every class or interface you create should belong to a package

Method Overloading 🡺 Methods with the same name but different Parameter Datatypes. E.g. Void m1(int g){}; void m1(String gg){};+