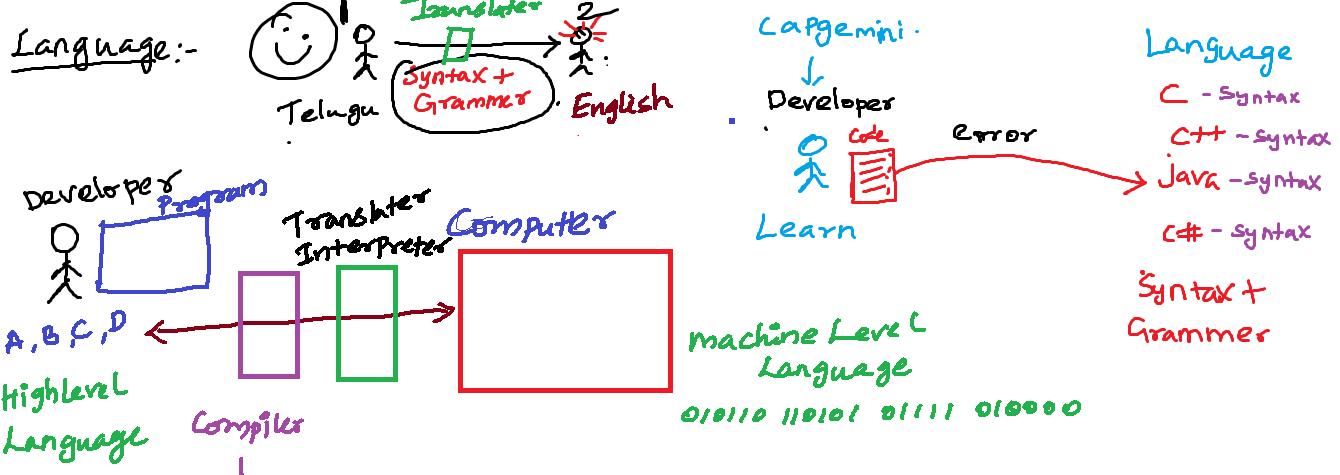
Basics of Language:-



Eclipse IDE:-

Workspace:

It is a folder contains one or more related java projects.

Project:-

It is a folder. It contains one more files related to that project

One or more java files

JRE System Library:-

Has all predefined java classes

Src :-

All user defined .java file must be place inside the src folder

Build -> javac filename.java

Run -> java filename

JDK

JRE

JVM

Sample.class 🡪 JRE 🡪 Machine code(output)

Sample.class 🡪 JDK(JRE), JRE 🡪 Machine code(output)

Sample.java🡪 JDK 🡪 Sample.class🡪 JRE🡪 Machine code(output)

.apk(.classes)

|

Phone (JRE)

|

Machine (Output)

Identifiers:-

All words in java. It is called as identifier

Reserved Keyword or User Defined Keyword

package name, class name, method name, variable name

Package Name:

* All letters must be in lower case
* Should not contains any space

Variable Name:-

* A java variable can start contain (a-z A-Z 0-9). Special character \_ $
* Variable name should starts with a character(a-zA-Z \_ $).
* It should not start with a number
* Variable name should be meaning full name
* If the variable is constant variable then variable name must be capital letter
* If variable name is single word all letters must be smaller case
* If variable name contains more than one word then first word all letter smaller second word onwards each word first letter must be capital

Method name:-

* If method name is single word all letters must be smaller case followed by bracket()
* If method name contains more than one word then first word all letter smaller second word onwards each word first letter must be capital then followed by ()

Class names:-

* Class name each word starting letter must be in capital all the remaining letters in lower case

Interface Name:-

* interface name each word starting letter must be in capital all the remaining letters in lower case

class Syntax:-

package <packagename>;

import <packagename>;

accessmodifier class <classname>{

constructor

variable

methods

}

Class in java:-

A class contains related members

Members – Member variable, Member Functions

Class is template or blueprint of an object

Class is a container in which we write related variable and methods (Encapsulation)

Object Oriented concept:-

Class and Objects:-

Why Functions?

Execute some logic or task

1. Function without arg without return value

**public** **void** multiply() {//perform task or logic

**int** a=29;

**int** b=45;

**int** c=a\*b;

System.***out***.println(c);

}

1. Function with arg without return value

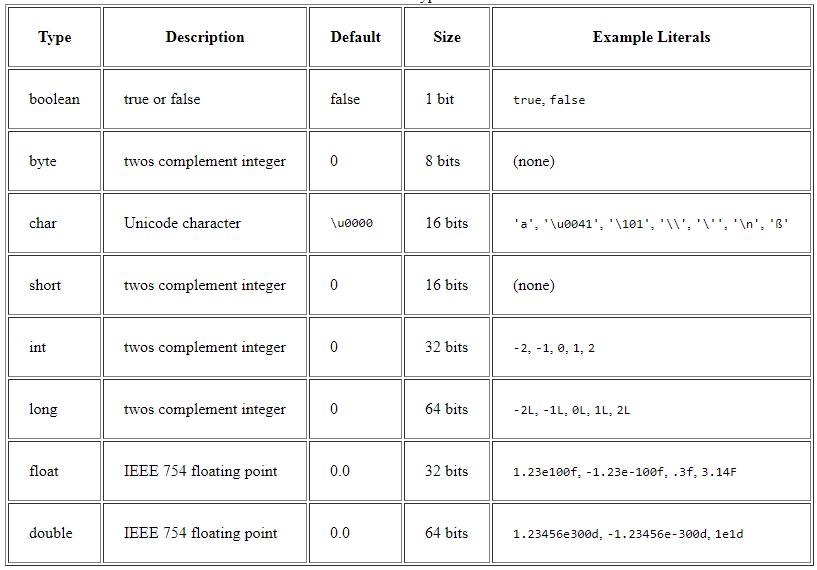
**public void add(int a,int b) { //perform task or logic**

**int c=a+b;// logic**

**System.out.println(c);**

**}**

1. Function without arg with return value
2. Function with arg with return value



Variable in java:-

Variable is an identifier for the memory location

Variable declaration

Datatype variablename;

Variable initialization

Variablename=value;

String name= “rajesh”

String rollno;

String year;

String sem;

int phone;

String email;

float cgpa;

rajesh, 3555SF, IV, VIII, 445454, [rajesh.kit@gmail](mailto:rajesh.kit@gmail), Chennai, 6.4

Types of variables:-

1. Instance variable
   1. Declared inside the class and outside of any method in the class without static keyword
2. Local variable
   1. A variable declared inside the method or block
3. Static variable
   1. Declared inside the class and outside any method with static keyword
4. Reference variable

Types of methods:-

1. Instance method OR non static method
2. Static method OR non instance method

Packages:-

Collections of related java classes, interfaces and enums.

Avoid the naming conflict

Variable declare:-

---------------------------

Accessmodifier Nonaccessmodifier datatype variable;

Class declaration:-

---------------------------

Accessmodifier Nonaccessmodifier class classname{

}

Function:-

Accessmodifier nonaccessmodifier returntype functionName(arg){

-----------------;

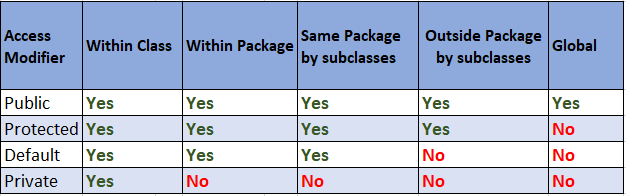
-----------------;

}

Access modifiers:-

private, default, protected, public

* Can be added before variable, method, class , interface



Non-Access Modifiers:-

Non access modifiers changes the default behaviour

Final, static, abstract, synchronized, volatile, native, transient

int mark=100;

Variable initialization:-

* Local variable -> while declaring a local variable we must initialize the value
* Instance or static variable
  + During variable declaration you can initialize

Flow Control

Conditional control statement

If, if else, if elseif, switch

If:-

if(condition){

statements;

}

Un-Conditional Control Statement

Break, label

{

}

Block – grouping set of statements

switch(condition or choice){

case label:

statements;

case label:

statements;

case label:

statements;

default

statements;

}

Java Operators:-

Assignment Operator

Int a+10;

Relational Operator -> comparing between values. It always returns a Boolean true or false

10>20 -> false

10>=20 -> false

10<20 -> true

10<=20 -> true

<,<=,>,>=,==,!= or <>

Arithmetic Operators

+,-,/,\*,%

Conditional Operator:-

(condition) ? value1 : value2; if(condition){

Statements

Else{

statements }

logical operators:

&&, ||, !

Control Statements:-

You can control statement or flow of statement execution

Conditional Statements:- if, if..else, if..elseif..else, switch

Un Conditional Statement:- break, continue, label

Looping statement:-

One or more java statement can be executed finite no of times. You have to loops

For, while , do..while, advanced for loop

For loop:-

for (initialization;condition;increment/decrement){

statements;

}

Advanced for loop:-

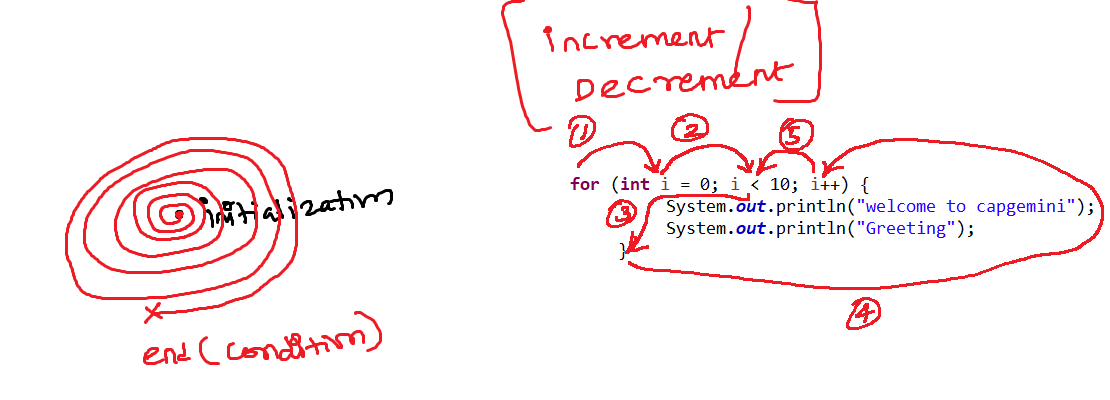
-----------------------------

for(datatype variablename:arrayname){

statements;

}

Instance of Operator:-



while loop:-

while(condition){

statements;

}

do while:-

initiliaztion

do{

statements;

increment/decrement;

}while(condition);

break and continue

Unlabelled break and Unlabelled continue

labeled break and labeled continue

Array:-

Group of similar data stored in a single variable is called as array

Array index starts with zero end with size-1

Eg:-

int[] mark=new int[] {46,34,67,23,78,89};// primitive array

String[] city=new String[] {"","","","",""};// reference type array - String predefined class

Employee[] employees=new Employee[] {e1,e2,e3,e4,e5,e6};//reference type array - Employee userdefined class

Object Programming:-

Classes:-

Encapsulation – data (variable) and the code (method) together kept inside the class it is called encapsulation

Putting all variable and the method inside the class

To make perfect encapsulation

All variable you need to keep it as private

Create a public getter and setter method through which other programs can

Access your data

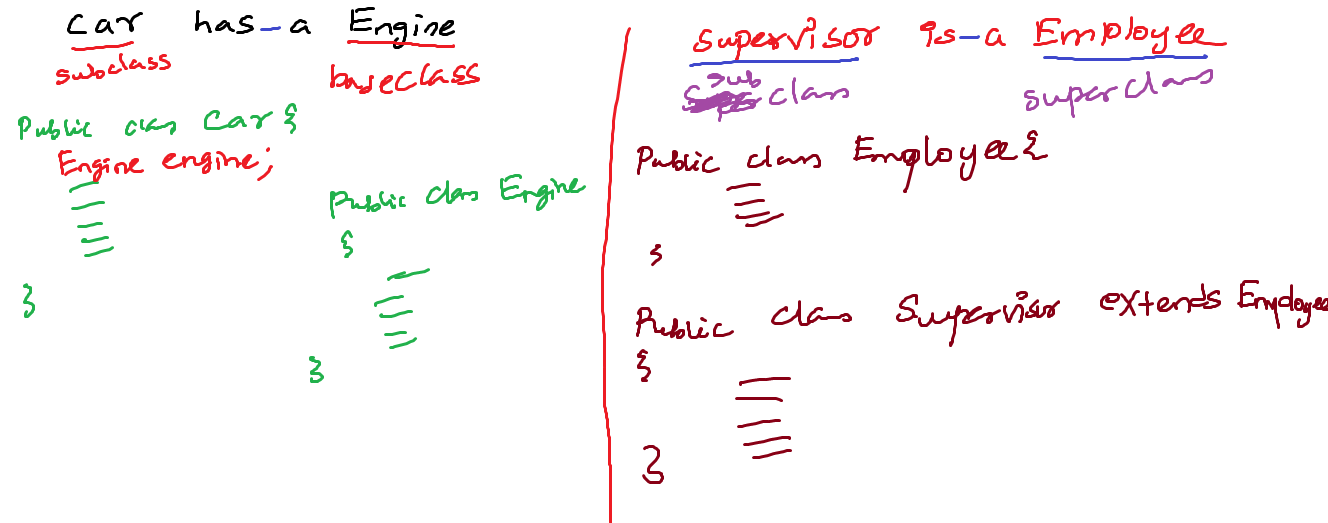
Inheritance can be achieved in two ways:-

Is-a inheritance

If you use **extends or implements** keyword to do a inheritance

Has-a inheritance

If you create an **reference** of one class into another class



Polymorphism:-

More than one form

A task can done more than one form

Send

Person A -----------------------------------------------------------🡪 Person B

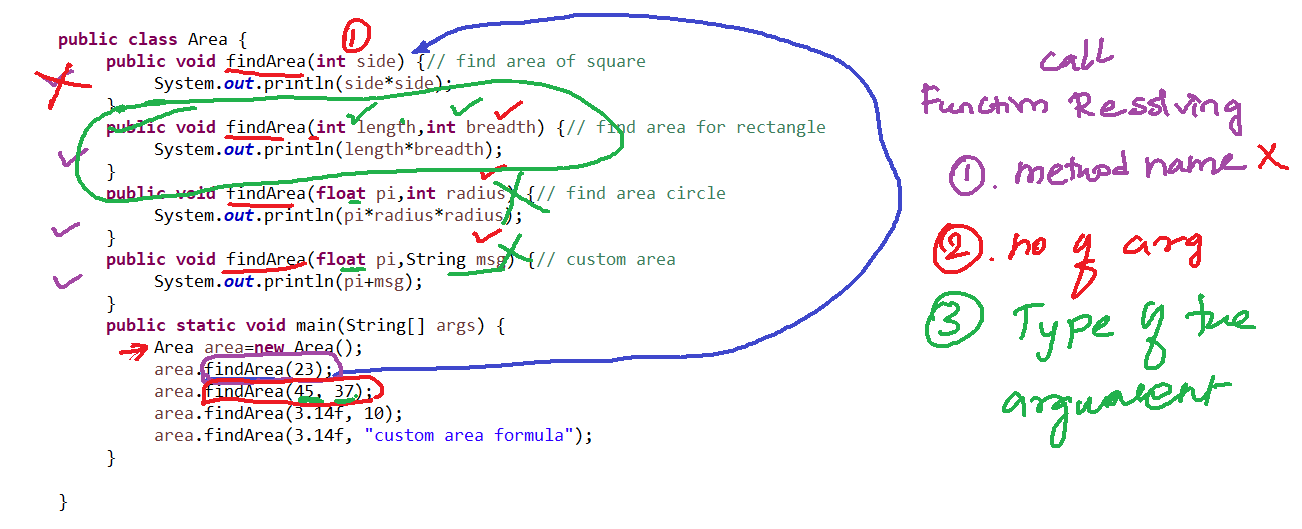
SendParcel (courier)

SendParcel (speedpost)

SendParcel (friends)

SendParcel (myself)

findArea:-



Function call resolving or method binding:-

For the function, call which method implementation or definition get executed decision-making is called function call resolving

If any conflict comes in the function call resolving

The function call resolving decision will be made based

1. Method name
2. No of arguments
3. Type of the arg

Function with same different argument in same or different class it is called as method overloading

Function with same same argument in different class it is called as method overriding

Abstraction:-

Hiding the implementation and showing the essential information is called abstraction

Car has very complex many mechanical parts, as you need to know how to drive the car

Ex:1

**public** **void** findArea(**float** pi,**int** radius) { System.***out***.println(pi\*radius\*radius);

}

This Implementation show all the details it is not abstraction

If method does have any implementation, it is called concrete method or implemented method or non-abstract method

Ex:2

**public** **void** findArea(**float** pi,**int** radius);

It is not the complete implementation and it is showing only the essential information (abstraction)

If method does not have any implementation, it has only the declaration. Abstract method or unimplemented method

Two ways you can implement abstraction:-

1. Abstract class - 100% abstraction
   1. Abstract class does not allow you to create object but you can inherit the abstract class

To other classes

1. Interface – 100% abstraction

If all the methods abstract method

For the abstract class and interface, you cannot create an object. But you can create reference variable

|  |  |
| --- | --- |
| Abstract Class | Interface |
| 1. Abstract class can have concrete or implemented methods | Interface can have abstract method or unimplemented methods |
| 1. An abstract class should have minimum one abstract methods may have zero or more concrete method | All the methods in the interface are abstract methods |
| 1. Inherit the abstract class to other child class **extends** keyword | Inherit the interface to child class **implements** the interface |
| 1. Abstract class methods need to explicitly add abstract keyword for abstract methods | All the method in the interface by default public abstract methods |
| 1. All the variable access modifier non access modifier default set by the developer | All the variable declared in the interface by default public static final variable |

A Class member:-

1. Member variable
2. Member function or method
3. Constructor
   1. Constructor is similar to your member function. It is special function that should not contain the return type and the name should be similar to your class name.
   2. Constructor will be called automatically. When we create an object for the class. Right after the object is got created then the constructor will be automatically

Constructor overloading

* A constructor with the same name and diff argument or parameter is called constructor overloading

No arg constructor

Arg constructor

Default no arg constructor

//jvm is going add one no arg constructor automatically by default during excution no arg default constructor

// jvm default no arg constructor it will all the instance and static variable and it initiliaze with the default values

Casting:-

Primitive Type Casting

* Primitive type
  + Converting one primitive to other primitive type it us called as primitive type casting
    1. Implicit type Casting or widening

byte – 1 byte – 8bits

short – 2 bytes – 16 bits

int – 4 bytes – 32 bits

long – 8 bytes – 64 bits

float – 4 bytes – 32 bits

double – 8 bytes – 64 bits

char – 2 bytes – 16bits

boolean – 1 bit

Explicit type Casting or Narrowing

Object Type Casting

* Object Type
  + Converting One object type to another Object type it is call Object type casting
    - 1. Implicit Object Casting or widening

Taking child class object reference and assigning into a parent class object reference

* + - 1. Explicit Object Casting or Narrowing

Taking parent class object reference and assigning into a child class object reference

Wrapper classes:-

A Object or reference type of primitive value

Primitive types (Wrapper classes)

byte – 1 byte – 8bits ----🡪 Byte

short – 2 bytes – 16 bits ----🡪 Short

int – 4 bytes – 32 bits ---🡪 Integer

long – 8 bytes – 64 bits ----🡪 Long

float – 4 bytes – 32 bits ----🡪 Float

double – 8 bytes – 64 bits ---🡪 Double

char – 2 bytes – 16bits ----🡪 Character

boolean – 1 bit ---🡪 Boolean

**Converting primitive type to the reference type or object type, we can do it by wrapper classes**

**Collection in java -> upcoming topic**

**Collections in java always deals with the object or reference types not with the primitive type**

**As a programmer -🡪 as a primitive type**

**User 🡪 Input 🡪 primitive type**

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

**int** salary = 10;

// Way 1:// convert primitive int type to the ref type

Integer iSalary = **new** ~~Integer~~(salary);

System.***out***.println(iSalary);

//Way 2:// convert primitive int type to the ref type

Integer i1=Integer.*valueOf*(10); // static method then we used to call the method using classname

System.***out***.println(i1);

**int** cSalary=i1.intValue();// converting reference Integer object into primitive int type

**int** icSalary = iSalary.intValue();

Integer ii=3456;// Autoboxing - automatically converts primitive value to reference type

**int** cc = ii; // autounboxing - automatically converts reference type into primitive type

**float** f=34.34f;

Float f1=Float.*valueOf*(f);

**float** f3 = f1.floatValue();

**char** ch='4';

Character c = Character.*valueOf*(ch);

**char** c1 = c.charValue();

}

Primitive type to reference type

Reference type to primitive type

String to a wrapper class reference type

String to a primitive type

Syntax error:-

Java exception:-

During the java code execution (Runtime), an abnormal condition will happen it is called **Java Exception**

**If exception is not handled it terminate the program execution abruptly** or immediately, it will allow to not execute the program further statements.

If the programmer handles an exception can terminate the program gracefully

Exception in thread "main" java.lang.ArithmeticException: / by zero

at exceptionhandlingdemo.Demo.main(Demo.java:9)

**----------------------------**

**Exception in thread "main" java.lang.NumberFormatException: For input string: "sfd20"**

**at java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)**

**at java.base/java.lang.Integer.parseInt(Integer.java:652)**

**at java.base/java.lang.Integer.parseInt(Integer.java:770)**

**at exceptionhandlingdemo.Demo.main(Demo.java:10)**

**--------------------------------------------------**

**Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 7 out of bounds for length 3 at exceptionhandlingdemo.Demo.main(Demo.java:12)**

**Exception Type - java.lang.ArithmeticException**

**Exception message - / by zero**

**Packagename.classname.methodname the line number where exception created**

**When an exception happens during the code, execution it will create java exception class object. That exception object has**

1. **Exception Type**
2. **Exception message**
3. **Packagename.classname.methodname the line number where exception created**

Java exception handle:-

1. Try catch

**Try block where you suspect the code can create an exception**

**try**{ //try block

}**catch**() {// catch block

}

**During the code execution finally block always gets executed whether the exception created or not created**

**finally**{ // finally block

}

1. Propagate the exception throws keyword

Upto jdk 1.6

After jdk 1.7

Java Exception:-

1. Checked exception

If the exception class extends Exception class its comes under the category of Checked Exception

1. Unchecked Exception

If the exception class extends RuntimeException class its comes under the category of Unchecked Exception

Own Exception or Custom Exception or User Defined Exception:-

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

Bank – account no – “”, null – InvalidAccountNumberException

Money->negative–InvalidDenominationException  
 Aadhar No -> InvalidAadharNumberException

If my **exception class** extends **Exception** – **custom checked exception**

If my **exception class** extends **RuntimeException** - **custom unchecked exception**

String:-

Collection of characters are called as string

String name=”capgemini”;

char[] name=new char[9];

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| c | a | p | g | e | m | i | n | i |

Name

String is a predefined class

Strings are immutable (value will not changes)

Two ways to create a String

1. String literal (Immutable)
   1. String collegeName=”IIT”;
2. String class object(Immutable)
   1. String collegeName=new String(“IIT”);
3. StringBuilder(Muttable)
4. StringBuffer(Muttable)

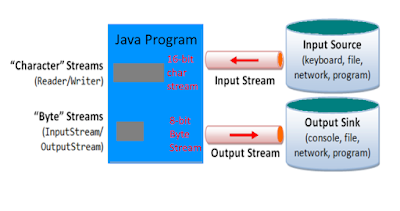
Java IO Stream:-

Input

(System.in) Keyboard, browser, File 🡪 Java Program

Output

Java Program 🡪 console, browser(System.out), File



Two types of streams in java

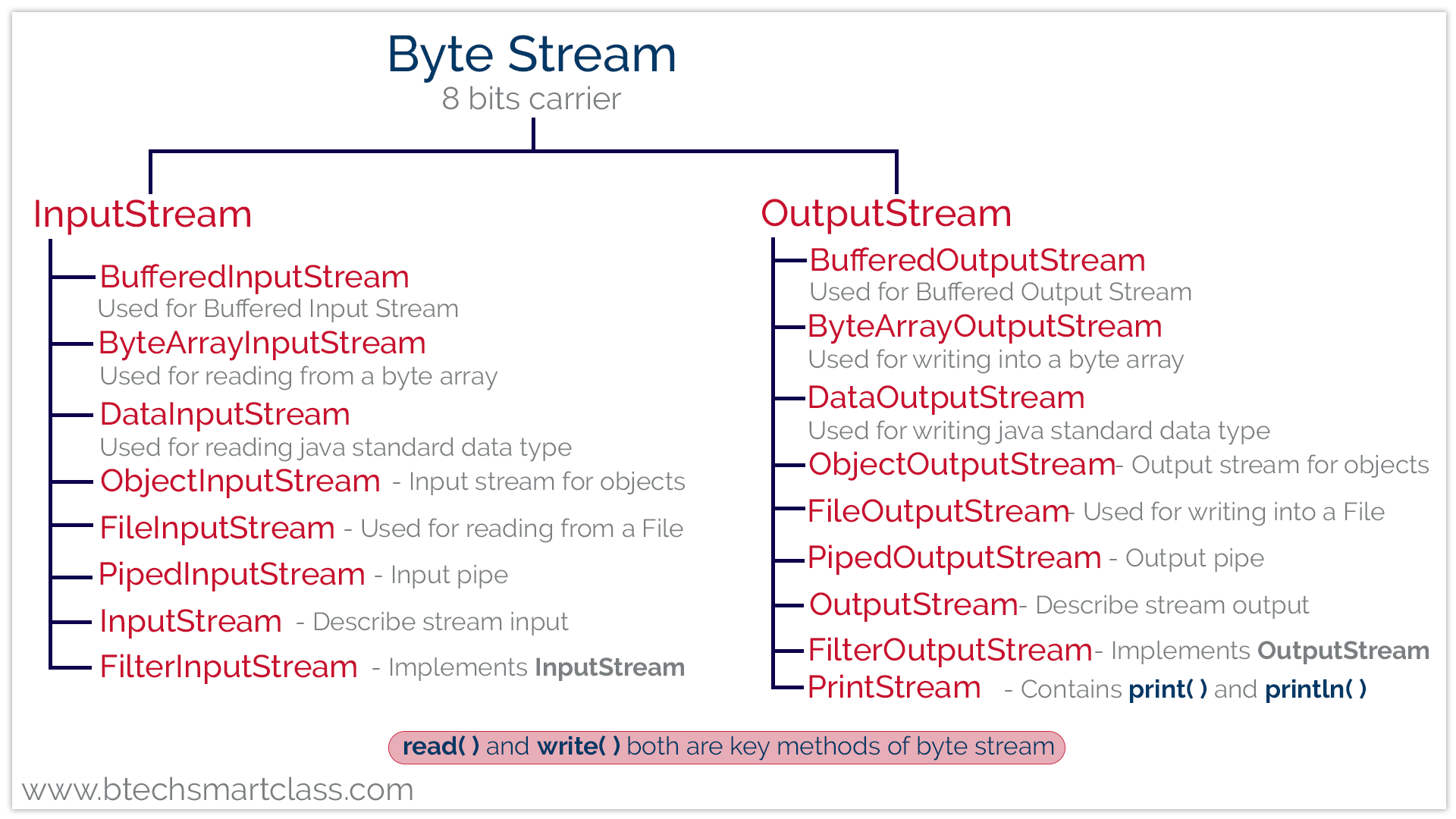
1. Byte Oriented stream (0-128)

Read or write the binary form of data

Image, video, audio

Read -> InputStream

Write -> OutputStream



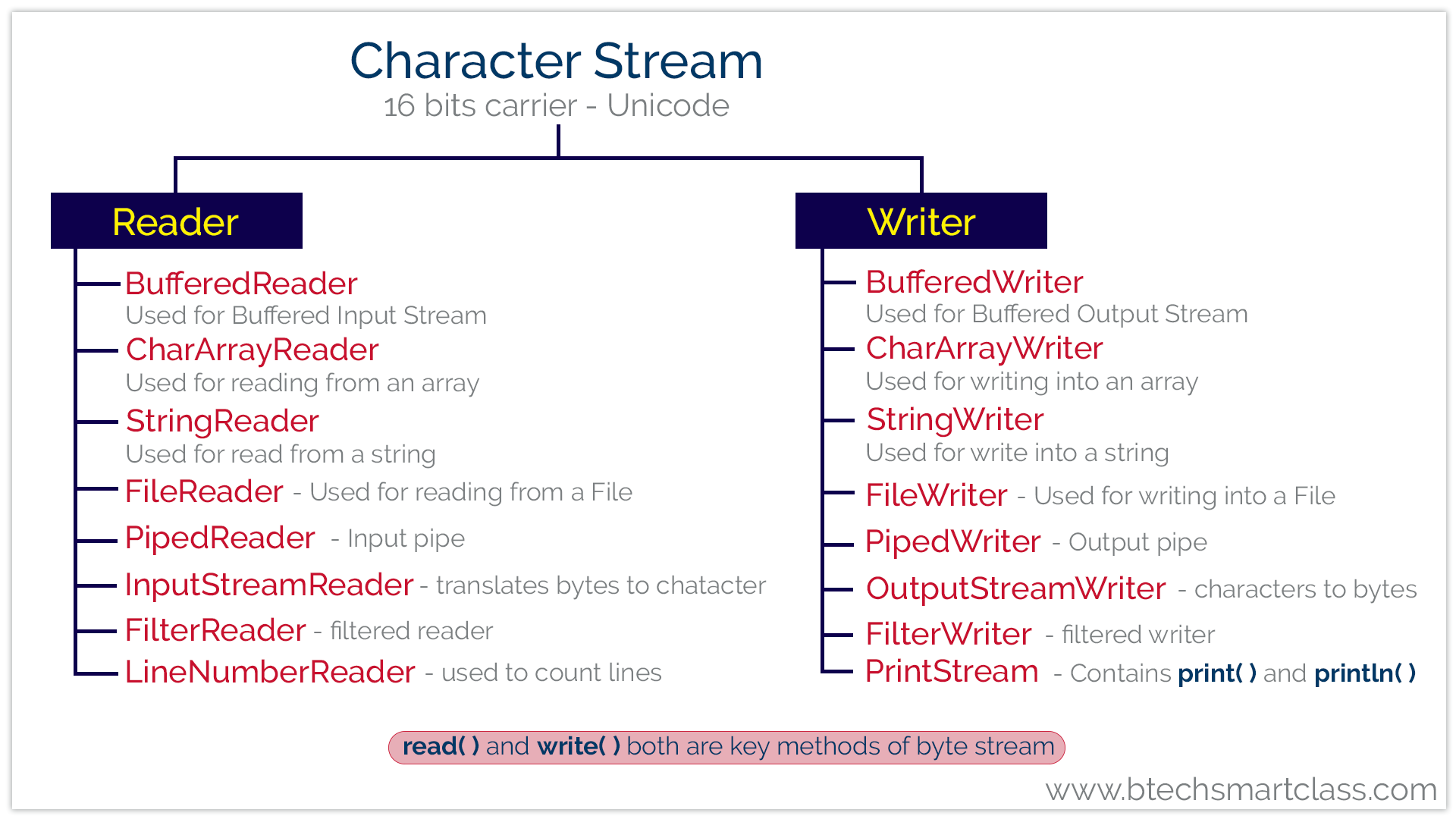
2.Character Oriented stream (0-256) – read / write

If you want to read text or character

Doc, xls, txt

Read -> Reader

Write -> Writer



Resources in java:-

If your java code or predefined class java code uses system hardware those classes are called as resources

Serialization:-

Array:-

Group or collection of similar types of primitive values or reference values (objects) are called as array

The group of values can be accessed by a single array variable name along with the index number

Array Index starts from 0 to size-1

Array -> primitive values, Reference values

Pro:-

* Array elements or item can be access by sequentially and randomly

0 1 2 3 4 5 6

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 56 | 34 | 89 | 45 | 9 | 80 | 44 |

Search 44

for(int i=0;i<array.length;i++){

if(array[i]==44){ //7 sec

}

}

Array[6]==44 – 1sec

Cons:-

* Once array size is fixed then you cannot increase or decrease further
  + int[] marks=new int[5];// static size
* Insertion and deletion of element involves movement of physical data back and forth.

Collections:-

Collection is a collection of or group of similar type of reference values (Object) that can be accessed by single variable name are called as collection.

Collections are dynamic in size. Size grow and shrink

Collection is a framework in java. Set of interfaces abstract classes and concrete classes

Each collection classes are backed by one data structure algorithm

All the collection are available java.util package

Collection stores a group object.

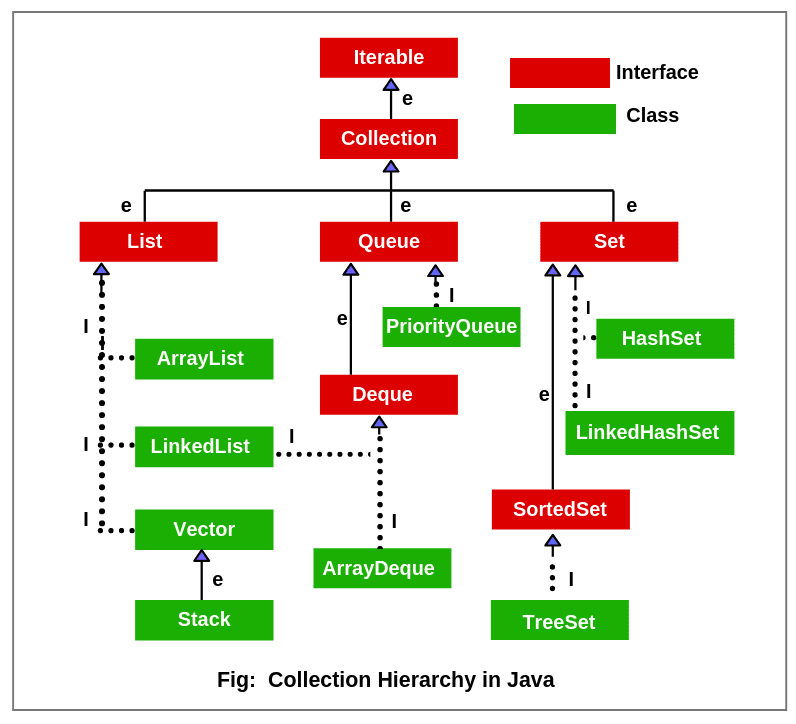
1. predefined object

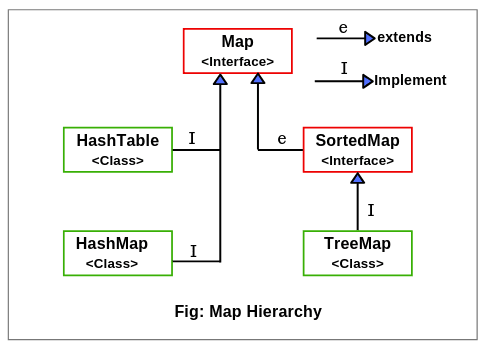
2. userdefined object

Datastructures -> array, linkedlist, stack, queue, hashing technique, tree and graph

Collection Class Hierarchy:-

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





How to you decide which collection class you need to use

1. Ordered or unordered

2. Duplicate

3. null

Collection – store group of objects

String s=new String();

s -> predefined string class object

Employee e=new Employee();

e -> user defined Employee class object

ArrayList list=new ArrayList();

-> array[16\*2] – index – random access

List.add(34);

LinkedList -> Doubly LinkedList datastructure

|  |  |
| --- | --- |
| ArrayList | Linkedlist |
| Array ds | Doubly linkedlist ds |
| Random access | Sequential access |
| Insertion deletion is very difficult | Insertion and Deletion is easy by rearranging the pointers |

Vector:-

HashSet :-

Unorederd

Not Accepting the duplicate (unique)

It will accept one null value

LinkedHashSet:-

Ordered

Not accepting the duplicate (unique)

It will accept one null value

TreeSet:-

Sorted Order (unorder)

Not accepting the duplicate (unique)

It will not accept even one null value

|  |  |
| --- | --- |
| Set Interface | Map Interface |
| HashTable,HashSet, LinkedHashSet, TreeSet | HashTable,HashMap,LinkedHashMap,TreeMap |
| HashSet – Unordered, no duplicate, one null value | HashMap – Unordered key, no duplicate key, one null key |
| LinkedHashSet – Ordered, no duplicate, one null value | LinkedHashMap – Ordered key, no duplicate key, one null key |
| TreeSet – Sorted in Ascending order, no duplicate, not even a single null accepted | TreeMap – Sorted in Ascending order key, no duplicate key, not even a single null key accepted |

List -> ArrayList, LinkedList, Vector -> Collections.sort()

Comparable –

Comparator –

HashSet,LinkedHashSet,TreeSet

Multitasking:-

More than one task executed in a computer is called as multitasking

Multitasking can achieved by 2 ways:-

1. Process based multitasking

Heavyweight

Context switch between the processes very costly

2. Thread based multitasking

Lightweight

Context switch is easy

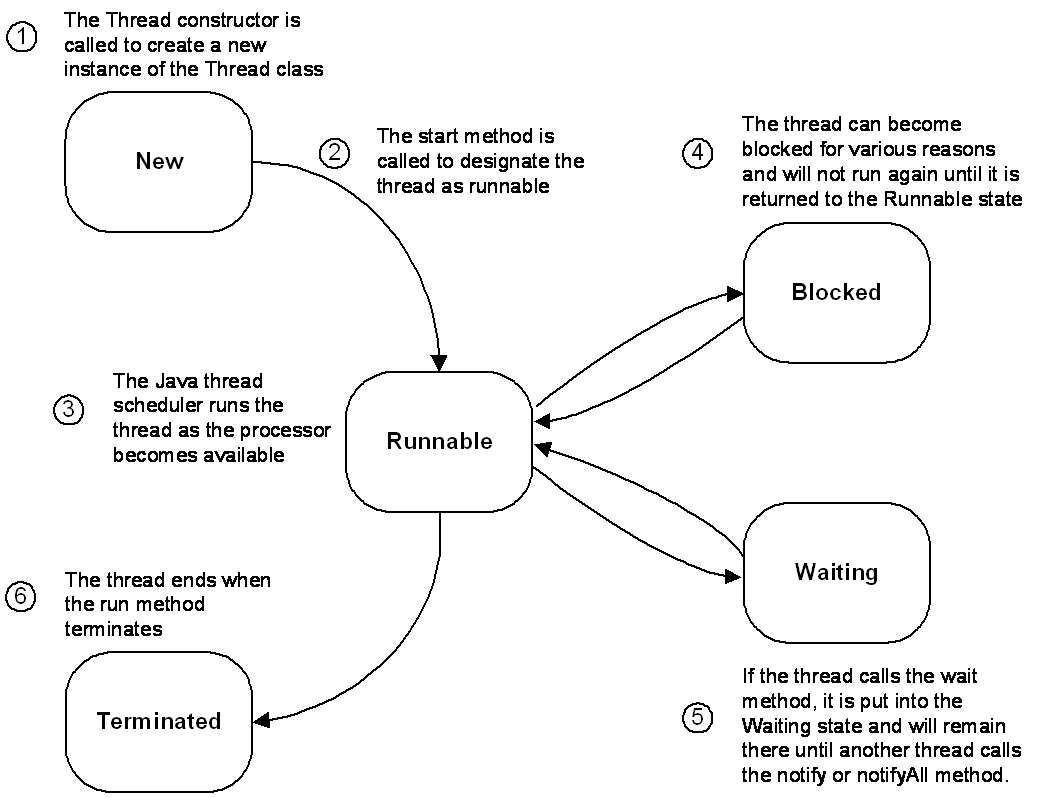
Thread -> executable in a cpu or processor

In java you can create a thread

Two ways you can create a thread

1. By extending a thread class

2. By Implementing a Runnable interface



Thread deadlock:-

All the threads are waiting for the same resource

Synchronized – can control the deadlock

Jdk 1.8:-

=======

Functional interface

Default method

Static method

Lamda expression

forEach

Stream API

LocalDate Loclatime

Functional Interface:-

* If an interface has exactly one abstract method that interface can call it as functional interface
* You can have one or more implemented method inside then functional interface
* From jdk 1.8 inside the interface you can write instance method with the default keyword and static method static keyword

1. User defined functional interface

2. Predefined functional interface

Runnable

Comparable

Comparator

Predicate

Function

Consumer

Lamda Expression rules:-

1. lamda expression does not take access modifier and nonaccess modifier

2. lamda expression does not take return type

3. lamda expression does not take method name

4. lamda expression after the arg need to place an arrow operator (->)

5. lamda expression implementation has only one statement can remove curly braces

5. lamda expression implementation has only one statement that too it is a return statement you can remove return keyword

Lamda expression:-

5. In the lamda expression has an arguments you can the data type of the argument

It provides a clear and concise way to represent one method

public void add(){

s.o.p(“welcome”);

}

Equivalent Lamda Expression:-

----------------------------------

()->s.o.p(“welcome”); // lamda expression

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

public void add(int a,int b){

s.o.p(“welcome”);

}

Equivalent Lamda Expression:-

----------------------------------

(a,b)-> s.o.p(“welcome”);

public void add(){

return 56;

}

Equivalent Lamda Expression:-

----------------------------------

()-> 56;

public void add(int a,int b){

return a+b;

}

Equivalent Lamda Expression:-

----------------------------------

(a,b)->a+b;

Lamda expression can be written with the help of only **functional interface**

Method Reference:-

It is shorthand for lamda expression

Instance method – Method reference

Static method – Method Reference

Constructor – method Reference

IO stream – input and output

Stream API:- process, managing, manipulating the group of objects

Introduced in Java 8, the Stream API is used to process collections of objects. A stream is a sequence of objects that supports various methods, which can be pipelined to produce the desired result.

The features of Java stream are –

* 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, they only provide the result as per the pipelined methods.
* Each intermediate operation is lazily executed and returns a stream as a result, hence various intermediate operations can be pipelined. Terminal operations mark the end of the stream and return the result.

Collections VS Stream APIEnums:-

Enumerations serve the purpose of representing a group of named constants in a programming language

Enums are used when we know all possible values at **compile time**

In Java (from 1.5), enums are represented using **enum** data type

In Java enums, we can also add variables, methods and constructors to it.

The main objective of enum is to define our own data types(Enumerated Data Types).

**Declaration of enum in java :**

* Enum declaration can be done outside a Class or inside a Class but not inside a Method.

**Important points of enum :**

* Every enum is internally implemented by using Class.

enum Color{

RED, BLUE, GREEN;

}

/\* internally above enum Color is converted to

class Color

{

public static final Color RED = new Color();

public static final Color BLUE = new Color();

public static final Color GREEN = new Color();

}\*/

* Every enum constant represents an **object** of type enum.
* enum type can be passed as an argument to **switch** statement.
* Every enum constant is always implicitly **public static final**. Since it is **static**, we can access it by using the enum Name. Since it is **final**, we can’t create child enums.

**Enum and Inheritance :**

* All enums implicitly extend **java.lang.Enum class**. As a class can only extend **one** parent in Java, so an enum cannot extend anything else.
* **toString() method** is overridden in **java.lang.Enum class**, which returns enum constant name.
* enum can implement many interfaces.

**values(), ordinal() and valueOf() methods :**

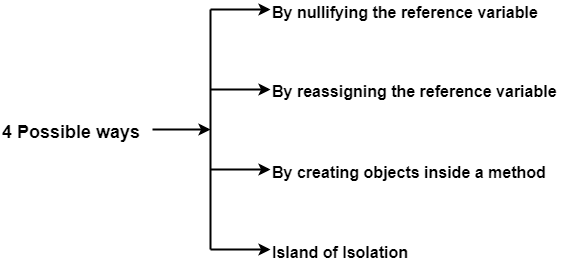
* These methods are present inside **java.lang.Enum**.
* **values() method** can be used to return all values present inside enum.
* Order is important in enums.By using **ordinal() method**, each enum constant index can be found, just like array index.
* **valueOf() method** returns the enum constant of the specified string value, if exists.

**enum and constructor :**

* enum can contain a constructor and it is executed separately for each enum constant at the time of enum class loading.
* We can’t create enum objects explicitly and hence we can’t invoke enum constructor directly.

**enum and methods :**

* enum can contain both **concrete** methods and **abstract** methods. If an enum class has an abstract method, then each instance of the enum class must implement it
  1. Writing Code That Explicitly Makes Objects Eligible for Garbage Collection



**Software testing**

**Testing -> we going to check piece of code is working as we expected**

**QA -> Quality Assurance**

**Unit Testing**

Unit testing is the process of checking small pieces of code to ensure that the individual parts of a program work properly on their own, speeding up testing strategies and reducing wasted tests.

### Integration Testing

Integration testing ensures that an entire, integrated system meets a set of requirements. It is performed in an integrated hardware and software environment to ensure that the entire system functions properly.

### End to End Testing

End to end testing is a technique that tests the application’s workflow from beginning to end to make sure everything functions as expected.

### Functional Testing

Functional testing checks an application, website, or system to ensure it’s doing exactly what it’s supposed to be doing.

### Non Functional Testing

Nonfunctional testing verifies the readiness of a system according to nonfunctional parameters (performance, accessibility, UX, etc.)  Which are never addressed by functional testing.

### Performance Testing

Performance testing examines the speed, stability, reliability, scalability, and resource usage of a software application under a specified workload.

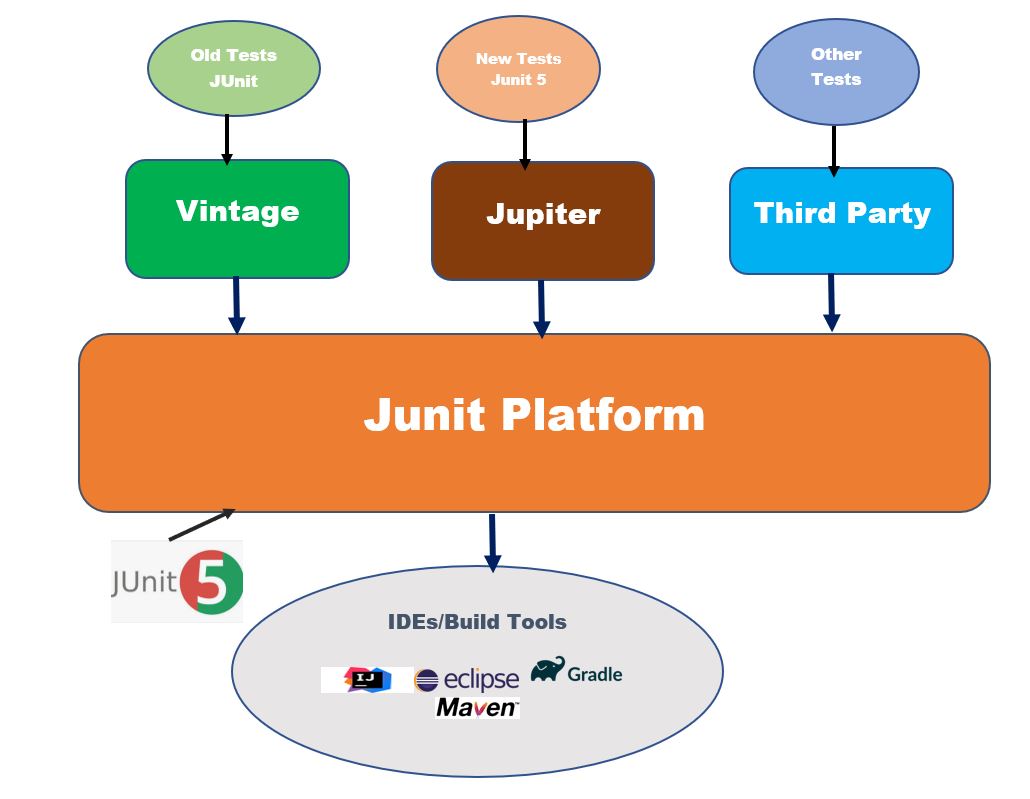
### Regression Testing

Regression testing is performed to determine if code modifications break an application or consume resources.

Unit testing in java

Junit – Testing Framework

**Unit test-> manual, automatic ->**



**Junit:**

**How to write a test case method**

**1. test case method must be public**

**2. test case method doesn’t return anything**

**3. test case method doesnot any argument**

**4. test case method name should with the keyword test**

**5. all the test case method annotated with @java**

**Test case**

**It is a java function. That has a rule. On top of the method @Test**

**@Test**

**Public void testFunctionName(){**

**}**

**Lifecycle method junit5:-**

**@BeforeAll -> In the test class before any test case gets execute before all works**

**@AfterAll -> In the test class after executing all the testcase after all works**

**@BeforeEach-> it will execute before each test case execution**

**@AfterEach -> it will execute after each and every test case executed**

**Note:-**

**Junit 5 needs jdk1.8**

I need junit5 dependencies

**Junit 5 Assumption:-**

**ASSUMPTION BASICALLY MEANS “DON’T RUN THIS TEST IF THESE CONDITIONS DON’T APPLY”.**

**Assumptions** class provides following overloaded methods.

1. Assumptions.assumeTrue() –  If the  condition is true, then run the test, else aborting the test.
2. Assumptions.false() –  If the  condition is false, then run the test, else aborting the test.
3. Assumptions.assumingThat() –   is much more flexible, If condition is true then executes, else do not abort test continue rest of code in test.

**Example**

public class JUnit5\_Assumptions\_Test {

// assumeTrue() -> Run only if environment is DEV

@Test

void testOnlyOnDevEnvironment() {

assumeTrue("DEV".equals(System.getenv("ENV\_SETUP")));

User user = new User(1, "Peter", "peterm@email.com");

UserService.saveOrUpdate(user);

assertTrue(UserService.users.get(new Long(1)) == user);

}

// assumeFalse() -> Run only if environment is not PROD

@Test

void testOnlyIfNotOnProdEnvironment() {

assumeFalse("PROD".equals(System.getenv("ENV\_SETUP")));

User user = new User(1, "Peter", "peterm@email.com");

UserService.saveOrUpdate(user);

assertTrue(UserService.users.get(new Long(1)) == user);

}

// assumeTrue() -> Run only if environment is DEV, if not display message

@Test

void testOnlyOnDeveloperWorkstation() {

assumeTrue("DEV".equals(System.getenv("ENV\_SETUP")),

() -> "Aborting test: not on developer workstation");

// not perform if assuption fail

User user = new User(1, "Peter", "peterm1@email.com");

UserService.saveOrUpdate(user);

assertTrue(UserService.users.get(new Long(1)) == user);

}

// assumeThat()

@Test

void testInAllEnvironments() {

assumingThat("DEV".equals(System.getenv("ENV\_SETUP")),

() -> {

// perform these assertions only on the DEV server

UserService.saveOrUpdate(new User(1, "Peter", "peterm@email.com"));

});

// perform these assertions in all environments

User user = new User(1, "Peter", "peterm1@email.com");

UserService.saveOrUpdate(user);

assertTrue(UserService.users.get(new Long(1)) == user);

}

}

## **What Is a**DynamicTest**?**

The standard tests annotated with @Test annotation are static tests which are fully specified at the compile time. **A DynamicTest is a test generated during runtime**. These tests are generated by a factory method annotated with the @TestFactory annotation.

A @TestFactory method must return a Stream, Collection, Iterable, or Iterator of DynamicTest instances. Returning anything else will result in a JUnitException since the invalid return types cannot be detected at compile time. Apart from this, a @TestFactory method cannot be static or private.

The DynamicTests are executed differently than the standard @Tests and do not support lifecycle callbacks. Meaning, the **@BeforeEach and the @AfterEach methods will not be called for the DynamicTests**.

@TestFactory Collection<DynamicTest> **dynamicTestsWithCollection**() { **return** Arrays.asList( DynamicTest.dynamicTest("Add test", () -> assertEquals(2, Math.addExact(1, 1))), DynamicTest.dynamicTest("Multiply Test", () -> assertEquals(4, Math.multiplyExact(2, 2)))); }

The @TestFactory method tells JUnit that this is a factory for creating dynamic tests.

**Each of the DynamicTest consists of two parts, the name of the test or the display name, and an Executable**.

@TestFactory Iterable<DynamicTest> **dynamicTestsWithIterable**() { **return** Arrays.asList( DynamicTest.dynamicTest("Add test", () -> assertEquals(2, Math.addExact(1, 1))), DynamicTest.dynamicTest("Multiply Test", () -> assertEquals(4, Math.multiplyExact(2, 2)))); }

@TestFactory Iterator<DynamicTest> **dynamicTestsWithIterator**() { **return** Arrays.asList( DynamicTest.dynamicTest("Add test", () -> assertEquals(2, Math.addExact(1, 1))), DynamicTest.dynamicTest("Multiply Test", () -> assertEquals(4, Math.multiplyExact(2, 2)))) .iterator(); }

@TestFactory Stream<DynamicTest> **dynamicTestsFromIntStream**() { **return** IntStream.iterate(0, n -> n + 2).limit(10) .mapToObj(n -> DynamicTest.dynamicTest("test" + n, () -> assertTrue(n % 2 == 0))); }

Parameterized Test

parameterized tests. This feature enables us to **execute a single test method multiple times with different parameters.**

In order to use JUnit 5 parameterized tests, we need to import the [junit-jupiter-params](https://search.maven.org/search?q=a:junit-jupiter-params%20AND%20g:org.junit.jupiter) artifact from JUnit Platform. That means, when using Maven, we'll add the following to our pom.xml:

<**dependency**>

<**groupId**>org.junit.jupiter</**groupId**>

<**artifactId**>junit-jupiter-params</**artifactId**>

<**version**>5.7.0</**version**>

<**scope**>test</**scope**>

</**dependency**>

**public** **class** **Numbers**

{

**public** **static** **boolean** **isOdd**(**int** number)

{

**return** number % 2 != 0;

}

}

@ParameterizedTest

@ValueSource(int s = {1, 3, 5, -3, 15, Integer.MAX\_VALUE}) // six numbers **void** **isOdd\_ShouldReturnTrueForOddNumbers**(**int** number)

{

assertTrue(Numbers.isOdd(number));

}

BDD:-

Behavioral Driven Development

Requirement -> code (logic) –> test (test class-> test case)

TDD:-

Test Driven Development

Requirement -> test (test class- test cases) -> code (implementation)

Mockito Framework->

## **Connect to PostgreSQL database**

psql -d database -U user -W

## **2) Switch connection to a new database**

\c dbname username

## **3) List available databases**

\l list the database

## **Listing databases in PostgreSQL using SELECT statement**

SELECT datname FROM pg\_database;

## **4) List available tables**

\dt

## **5) Describe a table**

\d table\_name

## **6) List available schema**

\dn

## **7) List available functions**

\df

## **8) List available views**

\dv

## **9) List users and their roles**

\du

SELECT current\_schema();

## What is Test Driven Development (TDD)?

Test driven development is an iterative development process. In TDD, developers write a test before they write just enough production code to fulfill that test and the subsequent refactoring. Developers use the specifications and first write test describing how the code should behave. It is a rapid cycle of testing, coding, and refactoring.

TDD means letting your tests drive your development (and your design). You can do that with unit tests, functional tests, and acceptance tests.

## What are the advantages of TDD?

### Code Quality

### Application Quality

### Increases Developers’ Productivity

### Higher Test Coverage

### Living Documentation

## Steps of Test Driven Development

### Step 1: Create a test and make it fail (Red)

### Step 2: Make the test pass by any means necessary (Green)

Postgressql

1. Create a database

Syntax:

CREATE DATABASE database;

2. Alter a database

ALTER DATABASE database;

3. Rename the database

ALTER DATABASE name RENAME TO new\_name

4. List available databases

\l

5. Switch between the database

\c databasename username

6. delete a database

DROP DATABASE databasename;

Managing the table structure:-

1. create a table

2. Alter the table – add a column, remove a column, rename the column

3. drop the table

4. rename the table

5. view the table structure

1. Create a table

CREATE TABLE tablename(column1name datatype(size),

Column2name datatype (size), column3name datatype (size),

……);

2. To view table structure or definition

\d tablename

3. Alter the table – Add a column

ALTER TABLE tablename ADD COLUMN columnname datatype(size)

4. Alter the table – Drop a column

ALTER TABLE tablename DROP COLUMN columnname;

5. Alter the table – Rename the columnname

ALTER TABLE tablename RENAME COLUMN oldcolumnname TO newcolumnname;

6. Rename the table

ALTER TABLE tablename RENAME TO newtablename;

7. delete a table

DROP TABLE tablename;

Psql(shell)

Rajesh ----------- postgresql

(commands -> SQL)

SQL (Structured Query Language) -🡪 RDBMS(mysql,oracle)

1. DDL(Data Definition Language)- structure

1.1 CREATE TABLE

1.2 ALTER TABLE

1.3 RENAME TABLE

1.4 DESCRIPTION TABLE

1.5 DROP TABLE

2. DML (Data Manipulation Language) (ROW DATA)

2.1 INSERT

INSERT INTO tablename values(value1,value2,value3,..);

INSERT INTO tablename (columnname1,columnname2,columnname3,…)values(value1,value2,value3,..);

2.2 SELECT -> read and show the table and the data

SELECT \* FROM tablename;

SELECT \* FROM tablename WHERE condition;

Where -> filter the row

Select -> filters the column from the selected row

Select \*

2.3 UPDATE

UPDATE tablename SET columnname=newvalue,

columnname=newvalue, columnname=newvalue,….

Where condition;

2.4 DELETE

DELETE FROM tablename WHERE condition;

3. DCL (Data Control Language)

3.1 GRANT

3.2 REVOKE

4. TCL (Transaction Control Language)

4.1 COMMIT

4.2 ROLLBACK

4.3 SAVEPOINT

Postgres Datatypes:-

Integer input -> smallint int bigint

Decimal input -> decimal, numeric, real, float

Character input ->

|  |  |
| --- | --- |
| character varying(***n***), varchar(***n***) | variable-length with limit |
| character(***n***), char(***n***) | fixed-length, blank padded |
| text | variable unlimited length |

Date input -> date

Time input -> time

Date time input -> timestamp

SQL Clauses:-

Clause – It is a filter

**Select clause** – from the selected table and the rows. filter the columns

**From clause** – database has many table. Filter and takes one table

**Where clause** – a selected table has many rows. It is filter the rows based on condition.

Where condition -> filter

**Limit Clause** -> it is going to limit no of records return by the where clause

**Distinct clause** used filter the duplicate value from the column

**Order By Clause ->** sort the resultset data

**Group By Clause ->** group the record or row based on one column

**Having Clause ->** always comes with group by clause

It is similar where clause

Where condition ->it filters the row from the table based on condition

Having condition -> it filters the row from the grouped by resultset based on condition

**SQL Functions->** select clause can call the function

**Min(columnaname)**

**Max(columname)**

**Sum(columname)**

**Avg(columnname)**

Count(columnname)

SQL Wild cards:-

* + If it varchar, char, text datatype column values partially can use wild cards search
  + Wild characters %\_
    - % - any character
    - \_ - single character
    - \* - all
  + Wildcards can be used in the where clause

Ex:-

S% -> name starts charcter S followed by any no character

%V% ->

ResultSet -> After the select query was executed. The select query results memory pointed by resultset. Limit clause from resultset

Constraints:- it is conditions applied on a column

Datatype:-

Restrict the value to be given in the column

Size:-

Restrict the length of the data

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Studentid  Integer(3) | Name  Varchar(10) | Cgpa  float | City  Varchar(8) | Phone  Bigint |
| 213 | Rajesh | 7.8 | Chennai | 568688 |
| 678 | Vishal | 8.8 | hyb | 678788 |
|  | Mani | 5.6 | chennai | 787668 |
| 213 | Rajesh | 7.8 | Chennai | 568688 |

Constraint can be set in the column

1. NULL -> that column null value

2. NOT NULL -> That column will not accept null value

It will accept duplicate value

3. UNIQUE -> that column all value must be different (doesnot accepts a duplicate)

It will accept a null value

4. PRIMARY KEY -> unique + not null

-> That column all values must be unique,

Doesnot accepts a null value

5. CHECK -> That column value should satisfy check condition

6. FOREIGN KEY ->

Cross join -> row are multiplied, columns are added

It will generate more data

Inner join -> it is going to return which are matching the join condition

Outer join -> it is going to return a results join condition both matching and non-matching record

Left outer join

Right outer join

Full outer join

Natural Join -> also a type of inner join but natural join does not takes explicit join condition. Inner join takes explicit join condition

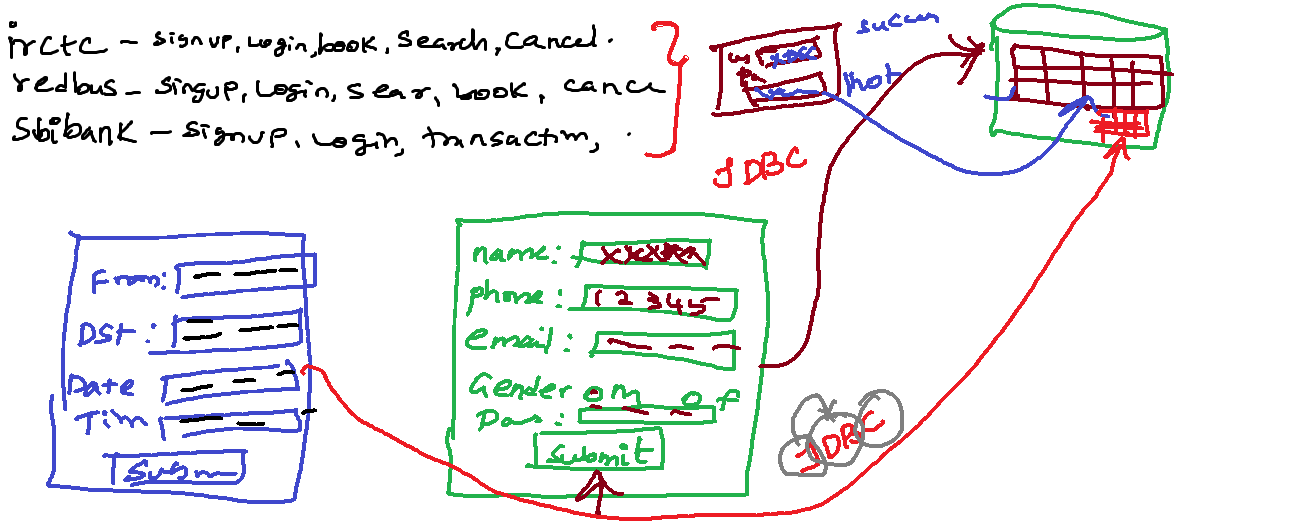
Natural join will not take an explicit join condition. Internally natural join will take both table common column name and it will create join condition implicitly

If both the table doesnot have a common column name natural join will yields the cross join results

Self join -> If we write join query using single table it is self join

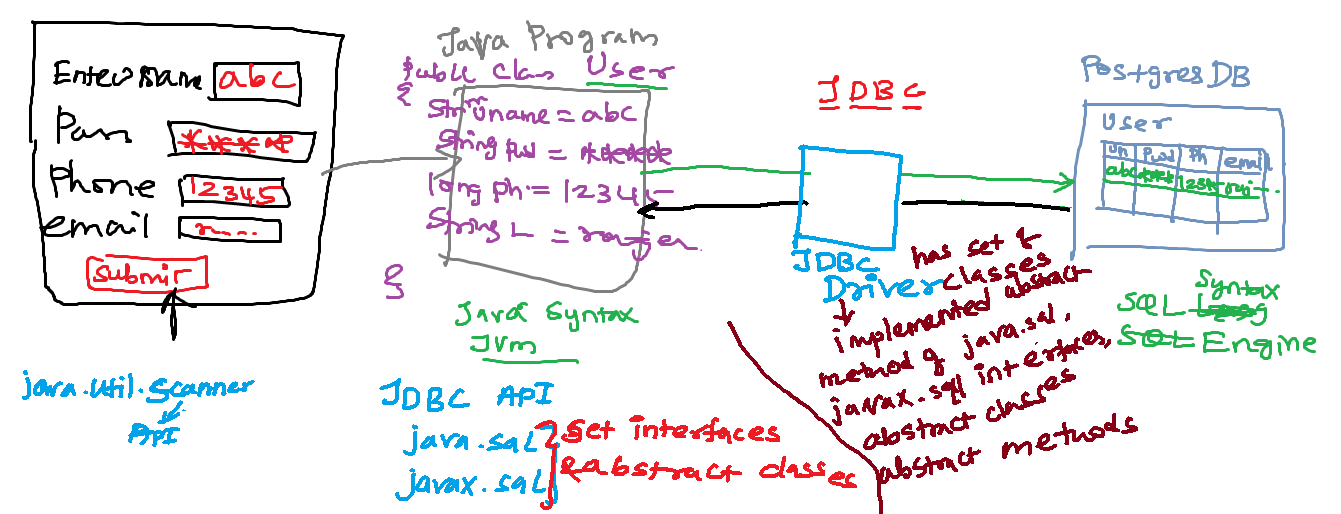
JDBC(Java Database Connectivity):-

Why JDBC:-



Jdbc api – java.sql, javax.sql – set of interfaces and abstract classes which has abstract unimplemented methods

Jdbc Driver -> has a set of classes that implements java.sql, javax.sql interface and abstract class abstract methods are implemented inside the classes of Jdbc Driver



Java.sql

DriverManager(C)-> getConnection(url,uname,password)

Connection(I) ->

Statement createStatement() **throws** SQLException;

PreparedStatement prepareStatement(String sql)

**throws** SQLException;

CallableStatement prepareCall(String sql) **throws** SQLException;

Java JDBC API – JDBC Driver

Introduction JPA:-

* ORM – Object relational Mapping
* JPA ORM specification – it is a standard. what to do.
* JPA -> guidelines to perform ORM
* JPA – It’s a specification without implementation
* JPA has set classes, interfaces, annotations help you do ORM

JPA(specificaton)

|

|

|--------------------|

Hibernate (JPA providers) EclipseLink(JPA providers)

(Implementation of JPA (Implementation of JPA

Specification) Specification)

* [DataNucleus](https://en.wikipedia.org/wiki/DataNucleus)
* [EclipseLink](https://en.wikipedia.org/wiki/EclipseLink)
* [Hibernate](https://en.wikipedia.org/wiki/Hibernate_(Java))
* [OpenJPA](https://en.wikipedia.org/wiki/OpenJPA) (from version 2.2.0)

Java Project -> JPA specification – JPA API available javax.persistence package

JPA Implementation or provider -> Hibernate API – org.hibernate package

Jdbc Driver -> postgressql jdbc driver -> org.postgresql package

Maven, Gradle -> if I need third party dependency into my project

Project management tools like maven, gradle is going bring third party jars from the repositories

JPA Architecture:-

