**Softwares**

-Eclipse 20 12R ---->https://www.eclipse.org/downloads/packages/installer

-OpenJdk 12 -----> https://jdk.java.net/archive/

C -----> 1970

C++ -----> 1980 -----> OOP's

Java invented in 1990 -----> Sun Micro System ---------> "Green Project" ------> James gosling

Java Technology --------> Java language

java officially launched ----> 1995 as open source technology in open market (JNU public Lic) jdk 1.8 free

JDK 1.9 (2017) java is no longer Open source technology Open JDK -----> Free

1) JDK 1.9 and above ( License version official website of oracle.com)

2) OpenJDK 9,10,11,12,13,14,15 (also provided by Oracle as Open Source) https://jdk.java.net/archive/

**--JDK** bundle of software executable files along with JRE(use for development purpose)

tool like javac, javap, rmic, appletviewer, java(interpreter)

for testing we use jre

**--JRE**  stands for Java Run Time ENV

bundle of java API plugin(Jar) and operating system specific files

for windows os --> .jar + .dll

for linux os --> .jar + .sh

highly platform dependent

**--JVM** \* JVM is abstract specification

\* Concrete implementation (JRE)-----> sun microsystem (Hotspot JVM)

\* Runtime instance ( 256 MB 1 per instance)

stands for Java Virtual Machine, it's Highly platform dependent

not installed but configure env on the fly

to configure JVM we need JRE(Highly platform dependent)

it has it's own memory management (stack, heap, method section)

it has it's own language (byte code)

--**Bytecode** - also known as .class

- Makes Java technology as platform independent technology

MainClass.java --> javac --> MainClass.class--> JVM ---> interpreter ---> O.S Specific Code ----> O.S for execution

(source code) (JLS) (bytecode) .jar+O.S (java) Specific file

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

public class MainClass{

public static void main(String [] args){

System.out.println("Hello World")

}

}

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

Java is Strongly Type Lang (DataTypes)

Types

Primitive category NonPrimitive datatype category

(Basic/Value Type) (Reference type/User define Type) variable holds memory address

byte 1 byte Class (String)

short 2 byte Interface

int 4 byte 1234 Array

long 8 byte 1234l Enum

double 8 byte 765.564

float 4 byte 764.564f

boolean 1 byte (true/false)

char 2 byte

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

**Keywords** : total 50 keyword out of which 48 keywords use

"const, goto" not use in development

byte, short, int, long, double, float, char, boolean

if , else, while, for,

final, class , interface, enum , public,

private, protected, this, super, default, void

transient, assert, package, implements, extends, continue,

null , abstract, try , catch , finally , static , true , false, import, case, synchronized , volatile

**Operators** unary operators

**prefix** int n=100; ++num , -- num , int num = --n; System.out.println(num+ " "+n) -----> 99 99

**post fix**  int n=100

num++ , num-- int num = n++;

System.out.println(num+ " "+n) -----> 100 101

binary operators

Arithmetic operators ---> / , \* , + , -

Relational operators ---> < , <= , > , >= , == , !=

Logical operators ---> && , || , !

Bitwise operators ---> &,| (Not frequently use in development ) !

1 byte = 8 bit

int num=10 << >> & , |

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

**Conditional statements**

achieve using "if-else" , "else if ladder", "nested if - else"

&& (All condition must be true)

TRUE && TRUE -----------> TRUE

TRUE && FALSE -----------> FALSE

FALSE && FALSE -----------> FALSE

FALSE && TRUE -----------> FALSE

TRUE || TRUE -----------> TRUE

FALSE || TRUE -----------> TRUE

TURE || FALSE -----------> TRUE

FALSE || FALSE -----------> FALSE

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

**Selection statement**

achieve using "switch case" \* apart from int, char, now String is also use (jdk 1.7)

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

**Iteration statement**

achieve using loops like

for loop

while loop

do - while loop

enhance for loop/foreach loop (Jdk 1.5)

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

**OOP's Features**

1)Abstraction

2)Encapsulation

3)Inheritance

4)polymorphism

**1)Abstraction**

**General** - Abstraction is a process of hiding the complex implementation details from the user --> Car , Monitor, Projector, www.facebook.com,

**From development aspect** - Its key feature which helps to identify relevant key attributes of an entity as per domain requirement, hiding irrelevant attribute

CGPayrollSystm

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*payroll\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Employee**

-id, firstName, lastName, dateOfJoining, dateOfBirth, designation

pfNo, pancardno, grade

**Salary**

-basicSalary,hra,ta,da,otherAllowance,monthlyTax,netSalary,epf

**SalaryAccount**

-accountNo,bankName,ifsCode

**CGBankingSystem**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*banking\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Customer**

no, firstName,lastName,dateOfBirth,mobileNo,pancard

email,gender,age,occupation,

**Address**

addresLine1,addresLine2,city,state,pincode,country

**Account**

no,balance,type

**Transaction**

id,date,amount,status,type

**Nominee**

firstName,lastName, dob,mobileNo, pancard,adharcard,age

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*FamousPizzApp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Customer**

firstName,lastName,dateOfBirth,mobileNo, email

**Address**

addressLine1 , addressLine2 , city,state,pincode

**Order**

code, amount, timestamp, status

**Pizza**

code , size , category, price, quantity

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

**2)Encapsulation**

- It is Key process to wrap behavior & attribute of an entity into single unit called "class"

**class**

is logical classification of an entity

it's a prototype/blueprint.

encapsulate set properties and methods (attribute & behavior)

don't required memory inside JVM

"class" keyword is use to create classes in java

Also known as "UDT :- User Define Types" (Reference Type)

class Salary{

int basicSalary, hra, ta, da, otherAllowance, monthlyTax, netSalary;

}

class BankDetail{

int accpountNo;

String bankName,ifsCode;

}

class Employee{

int id,noOfLeaveTaken;

String firstName,lastName, dateOfJoining,dateOfBirth,designation,pancardNo,pfNo;

}

class MainClass{

main()

}

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

**Naming Convention**

**class** -- Need to follow camel casing MainClass , BankDetail, Employee

**variable & methods** - First word initial char need to be in small case, rest of the word initial char need to be in capital case: firstName,lastName ,withdraw(), calculateSalary(), getCustomerDetails(), saveCustomerDetails()

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

**packages**

- Are logical distribution of classes, interfaces, enum in java as per logical relevance

- "package" keyword is use to create packages in java

- "package" statement must be first statement on the top of the class

Topmost package name in java is "java"

java

java.io || java.lang || java.maths || java.sql

lang io util maths sql net

(String,Object,Therad)

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

**For Payroll case study**

domain name -------> www.capgemini.com

reverse the domain name -------> com.capgemini.www

discard the word "www" -------> com.capgemini

add the project name at last

"payroll" is project name -------> com.cg.payroll com cg payrollcom.cg.payroll.main.MainClass

com.cg.payroll.pojos.Employee

com.cg.payroll.pojos.Salary

com.cg.payroll.pojos.BankDetail

main beans/pojos/entites/dtos

MainClass.java Employee.java

Salary.java

BankDetail.java

\* Package Access Rule (Self Study)

www.capgemini.com ------> website name

com.capgemini.www ------> reverse the name

com.capgemini.banking ------> remove "www" and add projectname for example 'banking' or 'payroll'

com.cg.banking

For Banking case study ------> com.cg.banking

Com cg banking

main beans /entity/dto/pojo

Customer.java ---> real world entites

Account.java

MainClass.java Address.java

Transaction.java

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

class name ----> Employee.class

package name ----> com.cg.payroll.pojos

com.cg.payroll.pojos.Employee.class --------> correct class name (Abslute name) (package name +class name)

com.cg.payroll.main.MainClass.class --------> correct class name

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

\* classes are loaded inside JVM as per links between them

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

**Memory allocation**

basic datatype variables(int, float, double, long, char, boolean, short, byte )

-- static memory allocation process required for basic datatype

int num=100 ---> 4 byte

**Object**

Its runtime instance of class

Does required physical memory inside JVM

using class developer can able to create N number of instances of particular type/class

has created using "new" keyword

-- dynamic memory allocation (instantiation/object creation) and initialization

int num=100;

Employee employee = new Employee();

new keyword -----> instantiation/object creation

Constructor ------>initialization

**Constructor**

-- its special method who's name similar like class name

-- don't have return type

-- is use to initialize newly created Object's data member

-- during life of an object it will be call only once

-- can be overloaded

-- if zero constructor is in the class, then compiler will add default constructor(non argument constructor) in .class file

-- if constructor is added in java class, then compiler will not add default constructor

Source code (Employee.java) byte code (Employee.class)

1)Employee{ Employee{

int no; int no;

String firstName, -------------> compiler -------------> String firstName,

Employee(){}

} }

2)Employee{ Employee{

int no; int no;

String firstName, -------------> compiler -------------> String firstName,

Employee(int, String, String)

} Employee(int,String,String){

}

}

Pojos classes Rule no **1**:- compulsory add default constructor in the POJO class

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

**Access Modifiers**

**public** :- allow data members/methods access anywhere in application

**protected** :- allow data members/methods access with in package but it allow data members/ method inheritance in other package subclasses

**private** :- do not allow data members/methods access out side of class

(100% data hiding)

in the absences of above 3 by default is "default" which allow data members/methods access withing package

default keyword should not use for "default" access modifiers

Pojo class rule no **2** :- data members must be private

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

getter/setter or accessor / mutator behavior

class Employee{

private int id;

private String firstName,lastName, emailId,mobile;

public int getId(){

return id;

}

public void setId(int id){

this.id=id;

}

}

Bean classes Rule no **3**:- add the getter /setter methods to deal with private data members of POJO class

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

**this keyword**

-- it use to point current object of working class

-- hidden pointer to current object within class

-- difference between object/instance variable and local variable with the same name

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

**Types of Variables**

\* Local variables

\* Instance variables

\* static variables

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

**Types of Method**

\* **instance methods**

-are know as Behavior of an object

-need an Object to work on it otherwise JVM will propagate "NullPointerException"

**\* static methods**

- Behavior of a class

- works on static variable of class if any

- don't work on instance variable

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

**Array**

-- it's collection of similar datatype elements

-- size is fixed at runtime

-- static data structure (fixed memory size) it's limitation of array

-- whenever you create array of any type inside jvm heap memory, object of an "Array" class created

-- Array class also know "first class Object "

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

**String Class**

StringBuffer and String Builder class

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

**Has - A Relationship**

- Also know as Association relationship

**1) Composition**

Car has Engine || Employee has Salary || Employee has BankDetail

**2)Aggregate**

Car Has Insurance

CGPayrollSystm Has - A Relationship

Employee Has A Salary

Employee Has A SalaryAccount

CGBankingSystem Has - A Relationship

Customer Has A Address

Customer Has A Account

Customer Has A Nominee

Account Has A Transaction

CGFamousPizzApp Has - A Relationship

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

**3) Inheritance**

- Key feature of OOP'S, which helps to reduce duplication of code(Behavior) and variable(attributes)

- One class allow his behavior and attributes to be inherit in to other class, its call Reusability

- its creates 'Is - A' relationship between classes

- class who's variables(data members) , and code(Behavior) inherit into other classes is call parent/super class

- class who inherit variable(data members) , and code(Behavior) from other class is call child/sub class

- Whenever try create object of child class first parent class constructor get invoke

class Object{

int hashCode() ---> use to generate hash Code value for object

boolean equals(Object obj) ---> Object equality check (memory address ) this==obj

void finalize() ---> part of garbage collection(Not recommended to override)

Object clone() ---> Override for deep cloning

String toString() ---> Object data members to single string

native final wait() /notify()/notifyAll() ---> not allow to override

}

class Person extends Object{

private id,firstName,lastName;

Person(){ super()}

public Person(int id, String firstName, String lastName) {}

getId(){}/setId(int id){}

getFirstName(){}/setFirstName(String firstName){}

getLastName(){}/setLastName(String lastName){}

equals(), hashCode()

toString(){}

}

class Employee extends Person{

emailId,designation,pancardNo,pfNo,basicSalary,totalSalary;

getEmailId(){}/setSetEmailid(String emailId){}

calculateSalary()

equals(), hashCode()

String toString() {

super.toString() + emailId+

}

}

class PEmployee extends Employee{

hra,ta,da;

getHra()/setHra(hra)

getTa()/setTa(ta)

getDa()/setDa(da)

equals(), hashCode()

calculateSalary() {

}

String toString() {

super.toString() + hra+ +ta+da;

}

}

**Polymorphism**

- static polymorphism ------> Method Overloading

- dynamic polymorphism ------> Method Overriding

**Object class Methods**

**- toString()**

It returns the current object data members string representation

**- equals()**

" == " -----> value type datatype (int , float, char , double , long, byte, boolean , short)

equals() -----> use for Reference type datatype

POJO class Rule No 4 :- must override hashCode() and equals() method in POJO class

**Abstract class**

Is a kind of class which can't be use for instance creation

when class is marked as abstract there can be N(0 can be possible) number of methods as abstract but if one method is abstract then class must be declare as abstract class

1)can we have instance(non abstract) method in abstract class ---> yes

2)can we have constructor in abstract class ------> yes

3)can we have static variables and methods in abstract class ------->yes

4) final class -----------> not allowed to inherit

5) final method --------> not allowed to override

6) final variable ------> it become constant(value will not change)

Employee

caluclateSalary()

PEmloyee CEmployee

caluclateSalary() caluclateSalary()

PEmployee pemp= new PEmployee() Same class ref ----> same class object (compile time/early/ static binding)

Employee emp= new PEmployee() parent class ref ----> child class object (run time/late/ dynamic binding)

Parent type Child class type

implicit/automatic/dyanamic/upward casting

emp.getId() no error

emp.getFirstName() no error

emp.getHra() error

emp.getTa() error

Employee emp;

emp=new PEmployee()

emp.calculateSalary() ----------> PEmployee

emp = new SalesManager()

emp.calculateSalary() ----------> SalesManager

explicit/downward casting

PEmployee pemp = (PEmployee)emp;

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

**Interfaces** User define data Type -----> class, interface

- Not allowed to create instance of an interface

- Instance variables are not allow, by default variable declare as static, public, final

- All methods are by default abstract, public

- All variables are by default are static ,final(constant), public

- Before JDK 1.8 only abstract methods declaration was allowed

- JDK 1.8 onwards now we can have method definition(code) as well , only need to mark these methods as default / static

- Are use for programming by contract

- Are use for specification declaration

JDBC , Servlet, EJB , Specification ()

- are use for loose coupling

Object

Person

Employee Student Actor

class Actor extends Person implements Hero, Comedian, Villain

Actor extends ------> Person extend

Actor implements ------> Hero, Comedian, villain (Roles)

class extends class ----> inheritance

interface extends interface ----> inheritance

class implements interface ----> abstraction

interface implements class ----> not allow

interface extends class ----> not allow

**Exception**

-Is kind of abnormal situation/event which occurs during software runtime/execution

-Need address this abnormal situation with care because chance of data loss, abnormal termination of software

-Some of these events occurs due to users or developers mistake , due to ecosystem problem

-Java support object oriented way for exceptions, class are there to represent the abnormal situation and event

-Object of exceptions classes encapsulate cause, message, code line no of the problem

Object

The topmost class for Exception tree is "Throwable"

Throwable (Object can be thrown out side of code )

Error (can't able to handle it) Exception

(Checked Can be Handled)

SQLException IOException

StackOverflowError (Unchecked)

FileNotFoundException

OutOfMeomoryError RuntimeException(Handling is not Complsury) EOFException

AssertionError ClassCastException

IndexOutOfBoundException

ArrayIndexOutOfBoundException

StringIndexOutOfBoundException

NullpointerException

ArithmaticException

InputMisMatchException

**Unchecked Exception** - Compiler will not force developer to have handling code for unchecked exceptions(compiler will not check whether handling block is there or no)

- Not recommended to log these exception

- are derived from "RuntimeException"

NullpointerException,

ArithmaticException

ClassCastException

NumberFormatException

InputMissMatchException

**Checked Exception**

- Directly derived from "Exception" class

- Compiler will ensure that developer must have handling code/declaration for Checked Exception (check whether handling code/block Is there or no inside methods)

- IOException, SQLExcetion, FileNotFoundException, EOFExcetion,

Checked Exceptions are handled by Compiler -----> wrong statement

try, catch, throw, throws, finally keywords are use for exception handling

**try** - is use to encapsulate Lines of code on which try block will have watch. It is use as Observer

**catch** - In case any exception occurred in try block then it will be handle into catch block

**throw** - it use to throw an object of an exception

**throws -** it will be use for declaration of multiple exception at method level, it will inform to caller method about exceptions

**finally** - all ways execute whether exception is there or not

**Default Exception Handler :-**

JVM BootMethod <---- main() <----- nextInt()

JVM Base code where main() method is call

JVM BootStarp code method(){

main();

}

main(){ ------

------

m1();

------

------

}

m1(){

try{

------

------

------ if(condition) Some Exception has occurred

------

------

}catch(SomeException e){

after exception code

throw e;

}

finally{

----

----

}

}

**Custom / User defined Exception :-**

Developer write business logic to address the client's functional requirements

There are certain business rules need to follow, if not followed, then custom exception need throw

**Layered Architecture :-**

1)Presentation Layer

-- UI of an application

2)Business/Service Layer -- Business Logic

3)Data Assess Layer

- CRUD operations

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

Wrapper classes /Object Wrapper business data + business logic

8 primitive/value types datatypes

int, float, long, short, double, byte, char, boolean ---> fix memory size

int num1=100 ----> 4 byte

int num2 =200 ----> 4 byte

int num3 = num1+num2 ----> Arithmetic operation is allow on value type variable

num1.equals() ----> Not allowed, to call equals() method or any other method, because it requires object reference

num1.toString() ----> Not allowed because value types are not ref type and methods are allowed to invoke on only ref type

Basic types Wrapper Classes

int --> Integer parseInt(), intValue(), MIN\_VALUE, MAX\_VALUE

float --> Float parseFloat()

long --> Long

double --> Double

byte --> Byte

char --> Character

boolean --> Boolean

short --> Short

String class is not warapper class

Before Jdk 1.5

Value to Object Object to Value

int num=100 Integer iob = new Integer(100);

Integer iob = new Integer(100); int num= iob.intValue(); //iob is reference of Integer class

After jdk 1.5

Value to Object Object to Value

AutoBoxing // JVM AutoUnBoxing// JVM

Integer iob = 100;

int num=100; //

Integer iob =100; // value wrapp into Object int num= iob; ---> value will unbox by JVM first

(In AutoBoxing "JVM" create object of Integer class

and wrap value into Object

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

**Generic programming(JDk 1.5)**

Addition

Method Overloading

void add(int,int){ System.out.println(10) println(int)

int+int

}

void add(float,float){ System.out.println(10.63f) println(float)

float+float

}

void add(int, float){ System.out.println("Satish") println(String)

int+float }

void add(float,double){

float +double

}

void add(long,float){

long+float

}

void add(long , double){

long+double

}

Generic programming (Template programming)

-- Don't support basic/ value type, it only support Wrapper | Object type

void add(T n1, E n2){ T ---> ? , E----> ? template/Unknown datatype

n1+n2;

}

Satish

T --> Integer

E --> Float

add(T n1 , E n2) { ---> source code ----> Compiler ------> void add(Integer n1,Float n2){ ----> JVM

n1+n2; n1+n2;

} }

Rajesh

T --> Float

E --> Float

add(T n1 , E n2) { ---> source code ----> Compiler ------> void add(Float n1, Float n2){ -----> JVM

n1+n2; n1+n2;

} }

class, methods, constructor, variables can be Generic

-------------- Wrapper classes, Generic programming

Car{

int id, name

}

OnRentService<T>{ OnRentService<Car>{

Object [] objects = new Object[10] Object [] objects = new Object[10]

T giveOnRent(int idx){ Car giveOnRent(int idx){

(T) return objects[idx]; -------> Compiler ------> (Car) return objects[idx];

} }

void takeBack(T element, int idx){ void takeBack(Car element, int idx){

object[idx] = element; object[idx] = element;

} }

} }

OnRentService<Car> onRentRef1 = new OnRentService();

onRentRef1.giveOnRent(); ------> it will return Car type

onRentRef1.takeBack(new Car(), 2); ----> it will accept Car type

class Truck{

}

OnRentService<T>{ OnRentService<Truck>{

Object [] objects = new Object[10] Object [] objects = new Object[10];

T giveOnRent(int idx){ Truck giveOnRent(int idx){

(T) return objects[idx]; -------> Compiler ------> (Truck) return objects[idx];

} }

void takeBack(T element, int idx){ void takeBack(Truck element, int idx){

object[idx] = element; object[idx] = element;

} }

} }

OnRentService<Truck> onRentRef2 = new OnRentService();

onRentRef2.takeBack(new Truck()); ---------> it will return Truck type

onRentRef2 == onRentRef1 -----> Not allowed

OnRentService<T>{ OnRentService<Truck>{

Object [] objects = new Object[10] Object [] objects = new Object[10];

T giveOnRent(int idx){ Truck giveOnRent(int idx){

(T) return objects[idx]; -------> Compiler ------> (Truck) return objects[idx];

} }

void takeBack(T element, int idx){ void takeBack(Truck element, int idx){

object[idx] = element; object[idx] = element;

} }

} }

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

Collection API

- Dynamic in size

- Classes are an implementation of data structure algo. (LinkedList, stack, vector, hashing, tree)

- Classes strongly support Generic programming because of that classes are Type Safe at compile time

- Classes supports different operations like insertion, removal, searching, sorting, iteration

- Works with only reference type and not with value type (int ----> Not allow || Integer ----> Allow)

- Need to use Wrapper classes instead of value type datatypes (Integer, Float, Long etc.)

Collection API Types

- List

- Set

- Queue

Non Collection Type

- Map (Key, Value)

Collection<>(I)

List<> (I) Set<> (I)

ArrayList<> (C) HashSet<> (C)

LinkedList<> (C) LinkedHashSet<>(C)

Stack<>(C) TreeSet<>(C)

Vector<> (C) CopyOnWriteArraySet<> (C)

CopyOnWriteArrayList <> (C)

List Type

-- Allowed duplicate object/elements

-- Index base access -------------->indexOf(element)

-- Classes implements List<> Interface

-- ArrayList, LinkedList,Vector, Stack,CopyOnWriteArrayList

\* ArrayList<>

- Dynamic Array (size will be change at runtime, internally array used to store element)

- Order Unsorted

- Internly supported by array to store element

- Index base access is allowed - iteration is faster

- very poor in performnce Insertion, removal, searching operations are very slow

- failfirst approch (ConcurrentModificationExcetion)

\* LinkedList<>

- Order unsorted

- It is an implementation of doubly Linked list algo.

- insertion & removal is faster in linked list

- search and iteration is slow

\* Vector <>

- Order unsorted

- Deprecated api (Not recommended in development) (Synchronize )

- Original collection since jdk 1.0

\* Stack <>

- Order unsorted

- derived type of Vector

- base on stack along.

- push(), peek(), pop();

\* CopyOnWriteArrayList

- Order unsorted

- Thread safe class

- Concurrency Api (Jdk 1.5)

- fail safe implementation

Comparable<T>

Comparator<T> Collection<>

List<> Set<>

1)Duplicate objects are allowed 1)Duplicate objects are not allowed

2)Index base access is allowed 2)Index base access is not allowed

get(idx) remove(Element) equals() contains(Element) indexOf(Element)

Set<> -- Unique elements collection

-- Do not allow duplicate objects //---> equals() from element Type Object ----hashCode and equals

-- Index base access is not allowed

-- sort() don't work with Set

-- HashSet, LinkedHashSet, TreeSet, CopyOnWriteArraySet (1.5)

\* HashSet

- Unordered Unsorted

- use hashing alogo. to store elements in bucket system

- hashCode() and equals(); need to override in element type in order to get correct output

- faster in search

\*LinkedHashSet

- Order unsorted

- it is like linked list in set

\*TreeSet

- Orderd Sorted

Collection<E>

List<E> Set<E> Map<K,V>

Key -----> Value

Map<> hashCode() equals()

-- It's key value pair base algo

-- Dictionary, HashMap, LinkedHashMap, TreeMap, Hashtable, Properties

-- Key type need override hashCode() and equals()

\* Hashtable

- deprecated API

- synchronized (Thread safe) in old way

- do not allow null key and null value

\* Properties - 1 extra constructor (String , String)

\* HashMap

- allow 1 null key and multiple null value

- Nonsynchronized

- substitute for hash table

\* LinkedHashMap

- Linkedlist in map

\* TreeMap

Treeset in map

\* ConcurrentHashMap

- concurrency api (java.util.concurrent) 1.5

- synchronized (using lock interface (concurrency way))

Common Set of operation supported by Collection api are insertion, removal, searching, sorting, iteration

=====================================================================================**JDK 8 new Features**

1) Default or static method inside Interface

- Only abstract methods was allowed inside interface

- After JDK 1.8 onwards interface can have default or static method

2) lambda expression -----> functional programming

1)Interface which have only 1 method as abstract called functional interface

2)Able to create lambda expression for only for functional interface

3)functional interface can have as much as "default" and "static" method

4)@FunctionalInterface annotation can be use to mark functional interface

5)Interface from past version of Jdk also can be considered for lambda expression if they content 1 method

- Runnable

- Comparable

- Comparator

2) Functional package

IntConsumer ----> void accept(int)

LongConsumer ----> void accept(long)

DoubleConsumer ----> void accept(double)

Consumer<T> ----> void accept(T) String , Customer, Employee

IntSupplier ---> int getAsInt()

DoubleSupplier ----> double getAsDouble()

Supplier<T> ----> T get(); String , IOException,ProductDetailsNotFoundException

IntPredicate ----> boolean test(int)

LongPredicate ----> boolean test(long)

Predicate<T> ----> boolean test(T)

IntFunction<R> -----> R apply(int)

Function<R,T> -----> R apply(T)

Also have the compatibility with old version interface like Runnable , Comparable , Comparator,

3) Optional<> Type

======================================XXXXXX======================================

Maven

-Project build tool (DevOps)

-manage the project dependences (Jar)

JRE ----> Default API classes (JRE) already added in the class path

Frameworks ----> Set API Lib use in application dev ----> Cucumber, Selenium , Spring

String.class ----> JRE (Java Api class)

org.h2.Driver ----> H2 database h2.jar (collection of .class files)

Cucumber.class ----> Cucumber plugin(3rd party API) ----> Jar ---> Search ---> Web Portal -----> download ---> add in classpath

In application ----> A.class ----> ABC.jar 1.3 (3rd party API)

A.class ----> B.class ----> PQR.jar 2.0(3rd part API)

B.class ----> C.class ----> LMN.jar 2.5(3rd part API)

Central Repo :- http://mavenrepository.com (All API Jar files are available)

Maven project -------> pom.xml -------> Page Object Model (POM)

ABC.jar 1.3 -----> pom.xml -------> Maven tool In build tool -------> Central Repository(Cloud) ------> download todm2 folder)

In application ----> A.class ----> ABC.jar 1.3 (3rd party API) (configure metadata of ABC.jar file in side POM.xml of an application)

A.class ----> B.class -----> PQR.jar 2.0(3rd part API) (Inside "ABC.jar" file, MetaData of "PQR.jar" file is stored)

B.class ----> C.class -----> LMN.jar 2.5(3rd part API) (inside " PQR.jar" file, MetaData of "LMN.jar" file is stored )

POM. xml it is base on the Object Repository Pattern (also use in Selenium as well)

Group Id :- com.springframework (Let's assume domain name :- www.springframework.com)

Artifact Id :- core.jar context.jar mvc.jar

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

GroupID :- com.cg

Artifact ID :- GoogleSerach Git ConferenceRoomBooking

package name :- com.cg.google com.cg.git com.cg.roombooking

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

Unit Testing using Junit framework (TDD ---> Test Driven Development):

- It is done by developer

- Unit testing fixes defects very early in the development phase that's why there is a possibility to occur a smaller number of defects in upcoming testing levels

- It helps to improve correctness of an application

- helps to test unit code (piece of code) (small functionality of a code)

- Junit Framework (Assertion)

- Negative and Positive Aspect need to test

Assertion and Unit Test

IF-Else Condition check

int num1=100,num2=100;

if(num1==num2) { --> checking condition , if fail then else part will execute

}else{

}

Assertion --> assumption

"assert" keyword --> jdk 1.4 (before jdk 1.4 it can be use as class name , or variable name , method name)

Assertion

int add(int a,int b){

return a+b;

}

int n1=100 , n2=100 (need to test "addition" functionality)

int expectedAns =200

int actualAns= add(n1,n2)

assert(expectedAns==actualAns) (expectedAns and actualAns must equals -----> Assumption)

if assertion fail then JVM will throw "Runtime Error" "AssertionError"

bydefault assertion is disabled, need unable using VM arg "-ea"

Junit 4.12

//@BeforeClass annotation base method will be executed by JUnit framework only once before all test cases

//method must be declare static

@BeforeClass

public static void setUpTestEnv(){}

//@Before annotation base method will be executed by JUnit framework before each test cases

@Before

public void setUpTestMockDataEnv(){}

@Test

public void test1(){}

@Test

public void test2(){}

@After

public void tearDownMockDataEnv(){}

//@AfterClass annotation base method will be executed by JUnit framework only once after all test cases

//method must be declare static

@AfterClass

public static void tearDownTestEnv(){}

import org.junit.Assert;

1)Assert.assertEquals (int, int) -------> == operator

Employee expctedEmp= new Employee(100,"Satish")

Employee actualEmp= new Employee(100,"Satish")

2)Assert.assertEquals(Object,Object) it assert that object's must be equals ---------> equals()

Assert.assertEquals(expctedEmp,actualEmp) -----> equals() ------> Employee, if not overridden then it will call from Object

String expectedString="Hello World"

String actualString="Hello world"

Assert.assertEquals(expectedString,actualString) ---------> equals() ---->String

Customer expecetdCustomer = new Customer(101,"Satish");

Customer actualCustomer = new Customer(101,"Satish");

Assert.assertEquals(expecetdCustomer,actualCustomer); --------> equals() -----> Customer

assertEquals(int , int ) ---> value type expectedAns and actualAns must be equal so it will use "==" operator

assertEquals(double, double ) ---> "=="

String expectedString ="Hello wolrd"; ----> Reference type expectedString and actualString must be equal so it will use .equals() --> ?

String actualString ="Hello World";

class Object {

boolean equals(Object obj) ----> memory address

}

class Product{

id,

name;

starRating

getter / setter

boolean equals(Object)

}

Product expectedProduct = new Product(111, "Pen", 4.5)

Product actualProduct = new Product(111, "Pen", 4.5)

assertEquals(expectedProduct,actualProduct) ----> Reference type it need equals() ----> Product

assertEquals(Object,Object) ---> assumption check that both objects must be equals other wise "AssertionError" will thrown

assertNotEquals(Object,Object) ---> assumption check that both objects must not be equals, other wise "AssertionError" will thrown

assertTrue(boolean) -----> assumption check that boolean value must be true, other wise "AssertionError" will thrown

assertFalse (boolean) -----> assumption check that boolean value must be false, other wise "AssertionError" will thrown

Product expectedProduct = new Product(111, "Pen", 4.5)

Product actualProduct = new Product(111, "Pen", 4.5)

assertSame (Object , Object) ----> assumption check that both objects memory address must be same other wise "AssertionError" will thrown

assertNotSame(Object , Object) ----> assumption check that both objects memory address must not be same other wise "AssertionError" will thrown

assertNull(Object) ----> assumption check that object reference must be null, other wise "AssertionError" will thrown

assertNotNull(Object) ----> assumption check that object reference must not be null, other wise "AssertionError" will thrown

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

@BeforeClass ----> Only Once Before all test cases (method must be declare "static")

@Before ----> before every test case

@After -----> after every test case

@AfterClass ----> Only Once After all test cases (method must be declare "static")

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

IO Programming :-

Stream - Its an abstraction of data transfer between source and destination

- Operations like Read and Write can be perform with Streams

- two types of streams are supported in IO API

- Byte Stream

- Character Stream

- NIO is new concepts been added in jdk 1.7

- its support channels

Stream

Byte Character

1 byte per char 2 byte memory per char

Low level IO High level IO

1 character at a time get transfer String get transfer at time

InputStream and OutputStream Reader and Writer

Byte Stream

InputStream OutputStream

FileInputStream FileOutputStream

ObjectInputStream ObjectOutputStream

BuuferedInputStream BufferedOutputStream

**Serializable**

- Marker/Tagged Interface

- no single method declaration

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

1)Assert.assertEquals(int expectedInt , int actualInt) == operator

Assert.assertEquals(double expectedDouble, double actualDouble)

Assert.assertEquals(Object expectedObject , Object actualObject) ---> equals()

Assert.assertNotEquals(Object expectedObject , Object actualObject)

Assert.assertTrue() assert that outcome must be true

Assert.assertFalse() assert that outcome must be false

Assert.assertNull() assert that return value must be null

Assert.assertNotNull() assert that return value must not be null

Assert.assertSame(Object,Object) assert that object reference must be equals (memory address)

Assert.assertNotSame(Object, Object)

Employee expctedEmp= new Employee(100,"Satish")

Employee actualEmp= new Employee(100,"Satish")

Assert.assertEquals(expectedEmp , actualEmp) // expectedEmp.equals(actualEmp)

Assert.assertSame(expectedEmp , actualEmp) // (expectedEmp == actualEmp)

----------------------------------------------------------------------------------------------------------------------------------- Collections.sort(List<? extends Comparable>)

2000 sort(){

List

Comparable comparableRef.compareTo() }

2) swapping Comparable<T>

int compareTo(T t)

String implements Comapable<String> Integer implements Compalable<Integer>

int compareTo(String str ){ int compareTo(Integer iob){

// //

} }

} }

Employee implements Comaparable<Employee> int compareTo(Employee emp){ // positive number return swap} }

DevTeam TestingTeam automation testing (BDD,TDD)