Java

Program – set of instructions

Machine level language – binary language – 0s and 1s

Compiler – Compiler converts program to machine understandable

format all at once and then execute.

int sum(int a ,int b){

return a+b;

}

Eg: c, c++, scala,smalltalk …

Interpreter- Interpreter converts each line of program to machine level

while it is executing.

Eg: Ruby,python etc..

Java is a platform and language.

Java follows WORA – write once run any where.

Features:

Simple

Platform Independent

Distributed

MultiThreaded

Robust

Secure

GarbageCollection- deallocation of memory when object its no longer being used- its taken care by jvm

Java is compiled and interpreted.

FileName.java – source code

compiled by javacompiler to bytecode- FileName.class

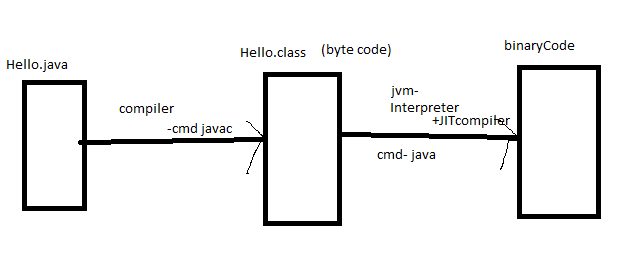
Interpreter converts to binary code and executes.

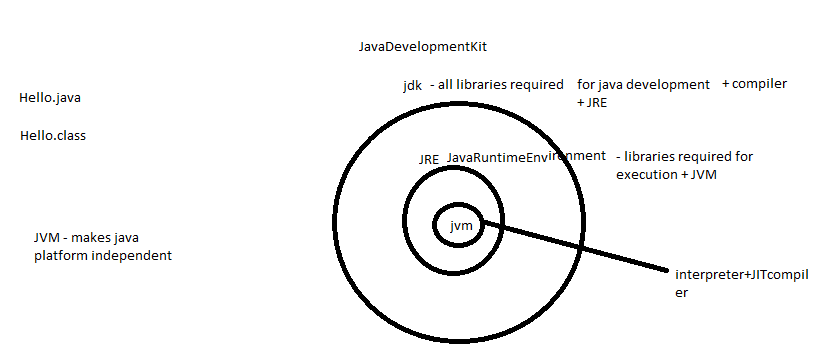
Java:

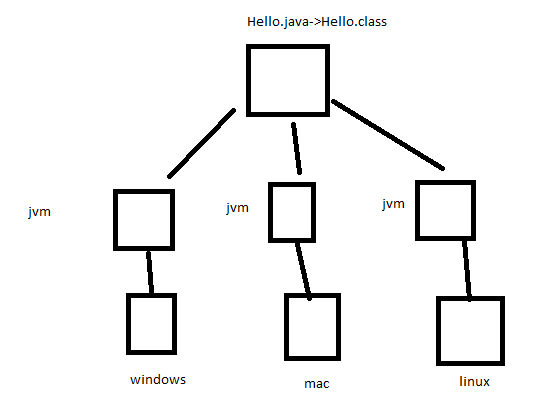
JDK – JavaDevelopmentKit

JRE- JavaRuntime environment

JVM-JavaVirtualMachine – jvm is system dependent but it makes JavaLanguage system independent.







jvm- interpreter + JIT(just in time) compiler

int sum(int a ,int b){

return a+b;

}

display all even numbers from 1-100

display(){

for(int i=1;i<=100;i++){

if(i%2==0){

print(I + “is even”);

}

}}

interpreter- 100\*1=100seconds + 3 seconds = 103seconds

JIT compiler – logic inside for loop is compiled to machine code in 1second , saved in memory and reused for all iterations.

1sec + 3 seconds – 4seconds

IDE : Integrated development environment which assists developer

to create, execute, debug and manage projects easily.

Java-Eclipse, Intellij-Ide’s-Jetbrains,Netbeans

Javascript- Webstorm , php storm

Android- AndroidStudio

IOS- xcode

Python- Pydev, pycharm

Open eclipse- create workspace-

Window-preferences- java- installed jre’s- point to jdk path instead of jre

Java is statically typed language

Javascript:

Var a=10;

Var a=”hello”;

Java:

Int a=10;

String a=”hello”;

Package : Package is a namespace/folder structure for organizing classes and interfaces

In a logical manner which helps projects easier to manage.

The first line in java program is package name.

Followed by import statements if any.

Followed by class declaration.

Java comments:

// single line coimments

/\* multi line comments \*/

Class: Class is like a blue print from which objects are created. Class defines state and behavior

of object.

Student{

Properties-variables:state

Id

Name

Actions/functions- methods: behaviour

Training

Mocks

CodingTests

Interviews

}

Object: Object is the instance of class which defines state through variables and actions through methods.

Objects:

Student s1= new Student();

S1.name=”Ahad”;

S1.id=1;

Student s2= new Student();

S2.name=”Ahad”;

S2.id=1;

Student s3= new Student();

S3.name=”Ahad”;

S3.id=1;

**Keywords** : reserved by language for their implementations:

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/\_keywords.html

**AccessModifiers**:

Public

Private

Protected

Default/package

**DataTypes**:

Primitive

Objects

**Variables** :

AccessModifier datatype variablename;

Public String color;

**Java is case sensitive**

Datatype- What kind of data the variable will hold

Number

Characters

Setofcharacters- String

decimals

Boolean- true or false

Int – integer

String

**Method:**

It is a set of code which holds the actual logic and can be called at any point making

the code reusable.

greet(String name){

Print(“hello ”+ name);

}

Int add(int a,int b)

{

return a+b;

)

**Method signature(syntax) and method body(actual logic inside method):**

**Accessmodifier** **outputDataType/void** if u don’t return any **methodname**(**input arguments if any/or leave blank**){

…Logic…

**Return** data as per declared datatype in methodsignature

**No need to add return** statement if its void

}

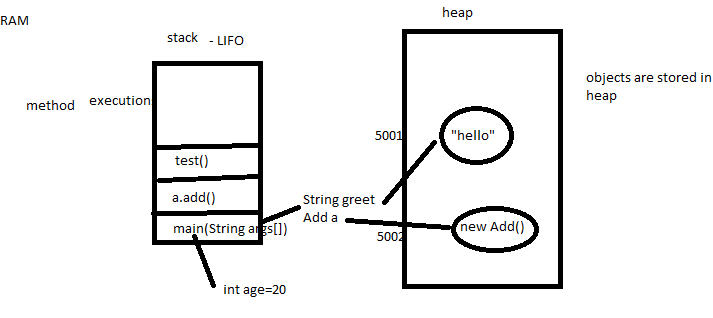
Always return statement is last line that wil be executed in a method.

**Compilation issues**- programmer deviates from language standards/ violates syntax rules.

Creating object in Java:

ClassName objname/reference variable = new ClassName();

Stack Heap:



Global or class level variables when not intialised with data will be given default values as per respective datatypes.

Object when they are not initialized – default value is null

For primitives it depends on datatype:

Eg- int-0, float-0.0

**Assignment 1:**

Create any class with properties and actions /call those methods/variables by creating object

Create a Calculator class with 4 methods- add, subtract,divide and multiply.

[training@whitebox-learning.com](mailto:training@whitebox-learning.com)

GarbageCollections- jvm deallocate the memory of objects in heap once it finds they are not being referenced from anywhere.

ctrl+shift+o- shortcut to import classes.

Constructor:

Constructor is similar to a method in java but it will not have return type, and it wil have same name as class name.

It is called by default whenever we create object.

It can take parameters which is a parameterized constructor.

Purpose of constructor is that it create an empty frame in memory

to reserve certain memory which is needed for object.

Uses of constructor:

Initialize any data required for the object because it wil be called initially When we create object.

A class can have more than one constructors with same class name

but different input parameters which is called constructor overloading.

2 ways to initialse class data:

Constructor Initialization

Setter Initialization

Access Modifiers: scope of class/method/variable is determined through access modifier.

Public – accessed anywhere within or outside the class

Private – accessed only within the class

Protected- accessed within the same package and also by subclasses

Outside the package.

protected variables can be accessed outside the package thru inheritance

default/package- accessed within the package

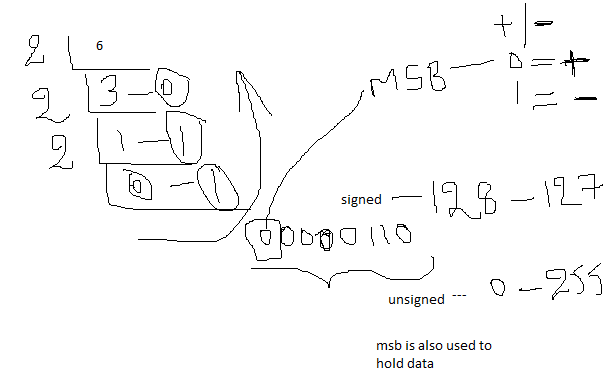
**wider🡪narrow scopes:**

public, protected ,default, private

all 4 access modifiers can be used w.r.to methods and variables

but for class we can use only public and default.

Datatypes:



Primitive: like literals, no need to use new keyword

Byte – integer with 8 bits => -128 to 127

Short- integer with 16 bits

Int- integer with 32 bits

Long- integer with 64 bits

Float - decimal with 32 bits

Double - decimal with 64 bits

Char – any keyboard character

Boolean- true or false

Other than primitive remaining all are objects in Java

byte->short->int->long->float->double

TypeCasting:

Explicit or downcasting – giving lower range values to higher range which we need to mention explicitly.

When you are doing downcasting make sure you are within the ranges to which you are casting.

Implicit or upcasting – giving higher range to lower range which

Is done by default.

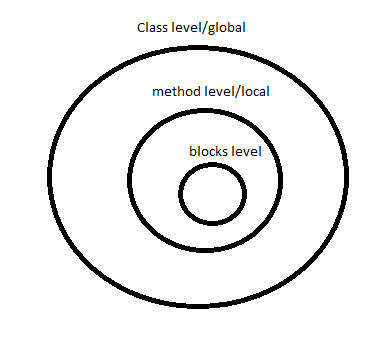
String – it is not primitive, it is class and we create objects for that class

Other than primitives remaining all are objects in java.

**Scope of variables**:

Global/Outer Scope

Local/inner Scope



**OOPS**: Object oriented programming concepts

**Encapsulation** – hiding the implementation behind an interface/ binding the variables and methods together by making variables as private and giving public getters and setters.

**Inheritance –** Child class extends parent class and inherits the features of parent class.

All the public and protected variables and methods of parent class can be accessed by child class.

Default level variables/methods can be accessed only if child is present

In the same package as the parent.

Advantage of inheritance is it makes code reusable/ avoid redundant logic.

Abstraction

Super: super is keyword used within child class which helps to access

parent class variables or methods.

Also super() is added in child constructor by default in the very first line

to call parent constructor if parent constructor don’t have any paramters.

If parent constructor is parameterized then developer should pass the parameters by super(parametervalues)

**Overriding** : It comes through inheritance, we can override the parent method logic in the child class when you want to implement a child specific behavior.

In overriding,The method signature should be same as parent method –

Static and final methods cannot be overridden.

**Accessmodifier returnType** methodName(input parameters)

But few exemptions with access modifier and return type:

Access modifier can be given a wider scope but not a narrow scope

Eg: parent method have protected- we can make it public

Overloading

Return type can be covariant return type means child class overridden method can return Subclass returntype

Eg:

public Greet hello()

Public StudentGreet hello()

**Polymorphism –** existence in many forms

A parent class reference can hold parent class object as well as child class object.

At compile time, the reference type decide which method should be called

At run time, the object type decides which method should be called.

Polymorphism applies to overridden methods(through inheritance)

TypeCasting w.r.to objects:

Explicit casting/downcasting: parent ref to child ref you need to cast explicitly.

When we are doing downcasting we should be careful that object is still of child type though ref is parent type else we get class cast exception.

Implicit casting/upcasting: child ref to parent –iit is done implicitly

**Rules for identifiers** – variables names /method/class names

First char can be \_, $ or letter but it should not be number

After first char u can include numbers.

No limit for length of variable.

**Standards/Naming Convention:**

Class/Interface names should start with capital letters- init Cap convention , meaning if more than word from second word again start with upper case

Mobile

AndroidMobile

Method and variable names will follow camelCase, initialWord start with lower case and if more than word from second word it start with upper case

Eg:

id

studentId

brandName

constants : all caps

PI=3.14

NAME=Wbl

MethodOverLoading : Having more than one method in the same class with same name but **different input parameters**. We can differ the parameters either by count of parameters or by datatype of parameters.

You may or may not vary return type, access modifiers.

**Overriding and Overloading differences:**

Overriding is through inheritance, when we try to override parent class method in child class.

In overriding the child class should maintain the same method signature as parent.

|  |  |
| --- | --- |
| Overloading | Overriding |
| Method arguments **must** change-either by number of args or by datatype | Method arguments **must not** change |
| Return type can be changed | Cannot change the return type except the covariant return types |
| Exceptions declared in method signature can change | Cannot change exceptions of super class in subclass method overriding-u can still add any unchecked (runtime)exceptions,narrower checked exxcpetions, but u cannot throw new or broader checked exceptions |
| Access modifiers can change | Cannot make **more restrictive** access modifier or we cannot reduce the visibility… |
| It decides which method to call at compile time- it exhibits virtual method invocation at run time | At run time based on which object is getting(subclass or superclass) created it calls that particular method, but at compile time it just decides on object reference – so we need to make sure that the reference objet has the method u r calling at compile time. But remember that at runtime, Java uses virtual method invocation to dynamically select the actual version of the method that will run, based on the actual instance |

Abstraction : through abstract class and inheritance .

**Assignment 2:**

Write a program to demonstrate inheritance

Write a program to demonstrate polymorphism

Write a program to demonstrate encapsulation

Write a program to demonstrate primitive typecasting

Write a program to demonstrate object typecasting

**abstract class:** It is create using keyword abstract class

We can have both abstract(method declaration without implementation) and concrete(method declaration with implementation) methods.

we can have constructor in abstract class.

we can have any scope to variables or concrete methods in abstract class. But for abstract methods in abstract class scope should be public

or protected.

We cannot create object for abstract class directly, when we create

child class object abstract class constructor is called.

whenever you have atleast one abstract method in a class- then class should be declared abstract.

abstract class may or may not have concrete methods.

**When to use:** When you want to provide common logic to child classes using concrete methods and you want to declare some abstract methods which child must implement.

**interface**: It is created using keyword interface

It is 100% abstract class and it is like a contract which child classes should follow.

It is like setting rules which child should follow.

All variables in interface are public static and final.

All methods are public and abstract by default.

We cannot create constructor in interface and also we cannot create object for interface.

In java , we can achieve multiple inheritance by interfaces.

Java supports multi level inheritance but not multiple inheritance with classes.

one interface can extends other interface or even it can extend more than one interface.

VALID:

**public** **interface** D **extends** A,B{

}

**When you choose interface:** When we don’t need to provide any common logic but just set rules to child classes, also in some cases if requirement needs **multiple inheritance** we choose interface

Eg:

Assume you have 100 child classes for interface and u want add new methods after few days and then all 100 classes will be forced to implement new feature if u add in existing interface.

If you don’t want to disturb existing things u can add it in a new interface and child classes which require that method will implement by multiple inheritance.

Also when multiple inheritance is needed we can use interface because in java we cannot achieve multiple inheritance through classes.

**static**: static means class specific but not instance/object specific.

static can be used w.r.to variables, methods and blocks.

static variables data is shared between the objects- it is not specific to any object.

we can directly call static variables and methods using classname.var/methName- we don’t need objects, even if we call them through objects it wil not throw error but internally it will still use className and call.

But u will see a warning msg in eclipse when u r calling static members with object.

static methods/blocks cannot access non static members but non static method can access static members.

static blocks are loaded first at the time of class loading.

**when to use:**

When we have common method to be used by everyone- generally we call helper/util methods in real time, we make such methods static bcz

we don’t want developer to create object and call whenever they want to use those methods.

Static methods cannot be overridden.

**final**: It means cannot be changed/modified.

final can be used wr.to variable,methods, classes.

final variable means we cannot reinitialize data – once assigned the value is constant.

final method means it cannot be overridden by sub class

final class means we cannot create subclass for the final class/ means final class cannot be inherited

Java classes support multi level inheritance

We can achieve multiple inheritance in Java only thru interface

class A{

public test(){

}

}

class B {

}

multiple inheritance- which is not supported in java

class C extends A,B{

}

assuming A and B are interfaces:

This is multiple inheritance thru interfaces which is poss in java

class C implements A,B{

}

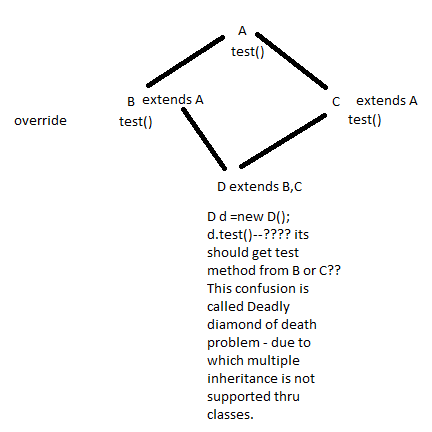
multilevel inheritance- which is possible in java

class D extends A{

}

class E extends D{

}



**this** is the keyword which refers to current object

It is useful when both local and global variables have same names in a class and we want to refer global variables in method or constructor.

ctrl+s- save single file

ctrl+shift+s- save all unsaved changes from diff files.

**Object** class – is the **super** class for all classes in Java by default.

method of Object class:

toString

hashCode

equals

wait

notify

notifyAll

finalize

clone

**Operators and loops:**

**https://docs.oracle.com/javase/tutorial/java/nutsandbolts/opsummary.html**

Operands – on which we apply operator

eg: a+b-> a and b are operands, + is operator

if operator is used between 2 operands – binary operators

if operator is used for single operand – unary operator

**Summary of Operators**

The following quick reference summarizes the operators supported by the Java programming language.

**Simple Assignment Operator**

= Simple assignment operator

**Arithmetic Operators**

+ Additive operator (also used

for String concatenation)

- Subtraction operator

\* Multiplication operator

/ Division operator

% Remainder operator

**Unary Operators**

+ Unary plus operator; indicates

positive value (numbers are

positive without this, however)

- Unary minus operator; negates

an expression

++ Increment operator; increments

a value by 1

-- Decrement operator; decrements

a value by 1

! Logical complement operator;

inverts the value of a boolean

**Equality and Relational Operators**

== Equal to

!= Not equal to

> Greater than

>= Greater than or equal to

< Less than

<= Less than or equal to

**Conditional Operators**

&& Conditional-AND

|| Conditional-OR

?: Ternary (shorthand for

if-then-else statement)

**Type Comparison Operator**

instanceof Compares an object to

a specified type

**Bitwise and Bit Shift Operators**

~ Unary bitwise complement

<< Signed left shift

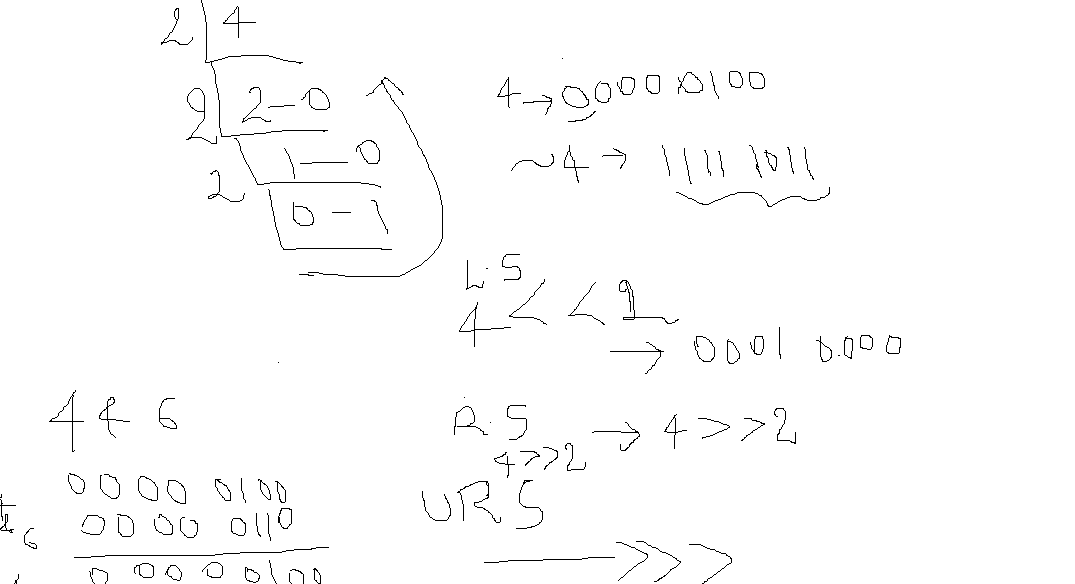
>> Signed right shift

>>> Unsigned right shift

& Bitwise AND

^ Bitwise exclusive OR

| Bitwise inclusive OR

****

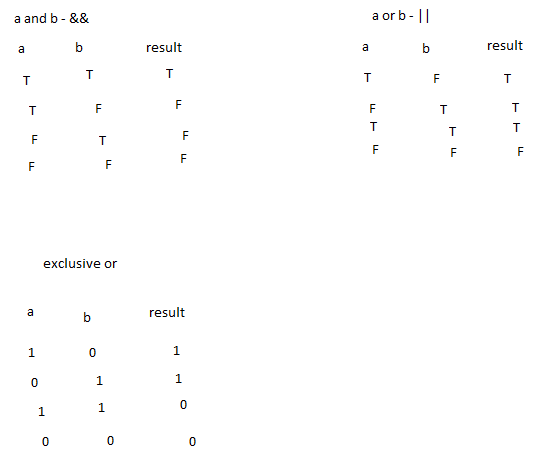
**use of bitwise:**

http://www.leepoint.net/data/expressions/bitops.html

conditionalOperators/short circuit:

&& - if first expression is false then it wil not execute second expression.

||- if first expression is true then it wil not execute second expression.



**Assignment 3:**

**Training- schedules, recordings, assignments, students List**

**StudentDetails- name, course,adress**

**QaTraining, UITraining, HadoopTraining, DevOpsTraining**

ConditionalStatements/Flow Control:

if else:

can be single or more than one by using else if

switch case: data passed in side switch can be either int, byte, short,char, String, Enum

Byte, Character, Short,Character

switch(""){

case "":

case "":

...

default:

}

**loop statements:**

while

do ..while

for

for( initialise; condition ; increment/decrement){

}

//1 to 10

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

print(i);

}

logic after for loop if present will be executed

branching statements:

break- exit from the current block of execution

continue - just continue with next iteration by skipping remaining logic in current iteration

while - wil execute logic inside loop only when condtion is not satisfied

dowhile - wil execute logic inside do atleast once even when condtion is not satisfied

getting student results from db

student name from each results

when no of results is not known but loop should continue till u reach end of results

while(rs.hasNext()){

student=rs.next();

}

**Assignment 4:**

write a program to find whether given number is even or not- return true/false

write a program that prints all multiples of 5 in between 1 to 100

WrapperClasses:

byte - Byte

short - Short

int - Integer

long - Long

float - Float

double - Double

boolean - Boolean

char – Character

Why Wrapper Classes:

To use the primitives inside Java Collections which allows only Objects.

They have easy to use methods eg: convert string to int , or int to string

Or any decimal to hexadecimal can be done by methods inside the wrapper

Classes.

Autoboxing: When primitive should be converted to WrapperClass or when Wrapperclass object should be converted to primitive it is done internally using Autoboxing concept in Java:

Primitive to WrapperClass- int to Integer – Boxing

WrapperClass to Primtive- Integer to int – unboxing

Arrays – Array itself is an object which can store group of primitives or objects.

The size of array should be given whenever we are creating array object

Int[] ids= new int[20];

Collections:

Collection:

Group of similar data

Store, retrieve, manipulate, delete, sort and search

Collections are present in Java.util- package

Collection- Interface

List – **ArrayList**,LinkedList,Vector, Stack

Set-**HashSet**,LinkedHashSet,**TreeSet**

Map- **HashMap**,LHM,**TreeMap**,HashTable

Queue-PriorityQueue

Collections- Class – which has some utility/helper methods

If collection is sorted => it is ordered

If collection is ordered it need not be sorted

ArrayList:

It allows duplicates. It allows null values(more than 1).

It follows the order of insertions.

It is faster for iterations and search purpose.

It is unsorted but it is ordered.

It allows null.

Vector:

Vector can handle the capacity and the method in vector are synchronized.Vector is slow in execution than array list.

It allows null.

LinkedList: Double linked list structure is followed in java

LinkedList has prev and next element refereneces which makes it faster for

Insertions and deletions.

It allows null.

hashcode

equals

Whenever 2 objects are being compared – hashcode and equals methods are used.

if hashcode is different for 2 objects which are having same values- still objects are not considered equal.

Only if hashcode is same, then the actual content is verified whether it is same or not

Set:- Allows only unique elements

HashSet- the data is unsorted and unordered, it will allow one null.

Internally depends on hashcode which makes it faster.

LinkedHashSet: It is faster for insertions and deletions,it is ordered by insertion order, It allows one null.

TreeSet- sorted in natural ascending order, sorted => it is ordered

It will not allow null.

Map: Key value collection, keys are unique but values can be duplicated.

HashMap – unordered, unsorted, it allows one null key and multiple null values

If you are using **userdefined object as key** in HashMap then you need to override equals and hashcode. Or else if u try to get value from map – u see its null

HashTable- It will not allow null- both keys and values, methods in HashTable is synchronized, It is slower than HashMap.

LinkedHashMap- maintains insertion order, faster for insertions and deletions, one null key- multiple null values

TreeMap-sorted in natural asc order, it will not allow null key but it allows null values.

Java Collections iterators are internally failfast- when we try to iterate and at the same time try to modify the collection it throws ConcuurentModificationException.

But Few Collections like ConcurrentHashMap are internally implementing failsafe iteration

which allows concurrent modifications

Comparable and Comparator: Sorting purpose.

2 interfaces used to implement user defined way of sorting.

Comparable- compareTo

This forces the class to implement Comparable and for this we need to modify the existing class.

comparator- compare

We can do this by modifying existing class or by taking a new class that imeplements comparator.

Inside TreeMap and TreeSet if we want to use user defined objects then we need to implement

either comparator or comparable.

We can use this comparator or comparable to sort list of userdefined objects(students) also by using Collections.sort(listObj).

If its normal list with String or any wrapper class directly we can pass to Collections.sort without

need to implement comparable or comparator.