**how to set path**

cd /d F:\java **-/d** for change disk location - **cd** for change path in that disk

how to run code

F:\java>javac test.java

F:\java>java Palindrome -use java and main class name

**Ambiguity means in state of confusion**

**Wrapper classes** provide a way to use primitive data types (int, boolean, etc..) as objects.

It convert primitive data type into object.

**java**--is not 100% object oriented it has primitive data type

java was developed by sun microsystems in 1995.james gosling is known as the father of java

befor1.6 java owned by sun microsystems (run without main method)

after 1.7 java owned by oracle (main method compulsory)

**constructor**

constructor is a special method with the same name as the class. It is automatically called when an object is created. Constructors are used to initialize the objects attributes/properties. There are two types of constructors: \*default constructor, which initializes attributes to default values,

\* Parameterized constructor, which allows you to pass values during object creation for custom initialization.

A constructor is a special method within a Java class.

It’s gets invoked when you create an instance (object) of that class.

The constructor’s job is to initialize the object’s state (such as setting initial values for variables).

constructor ==initialize and call class

**String:**

Implements Serializable, CharSequence, and Comparable interfaces.

Comparison methods:

==: Tests for reference equality.

.equals(): Compares the content of two strings.

compareTo(): Compares two strings lexicographically and returns an integer.

When using the equals method for objects, it checks the hash code for equality.

String's equals method compares content.

**main 5 blocks**

**static** 1.run one time only syntax -static { }

2.it will always execute at the time of class loading

3.static blocks execute one by one/line by line

**non-static** = 1. execute just before object syntax -{}

2. you call object non-static block execute

3.run line by line

**constructor**

**main method**

**method**

**difference jdk---jre----jvm**

**java-dev-kit** = 1.it is used to develop your java application

2.it is used by developer

3.it is pack of jdk and it contain jre and jvm

**java-runtime-environment**= 1.it is used to run java application

2.it contain class libraries

3.it is used at client-side machine

**java-virtual-machine**= 1.it is used to compile and run program

2.doesnot physically exist

3.it compile java code to bytecode

**java.Lang** package is fundamental package 2.not need to import this package

**program**==**stack memory** =compile time - stores variable/method stored

==**heap memory**=runtime- stores constructor

**class** is imaginary entity--memory is not present---collection of object

**object** is real entity---it has state and behaviour

**OPP's object oriented programming**

**4 pillars of opps concept**

**inheritance**--- inheritance is the most powerful feature of opps

it allows us to inherit the properties of one class into another class

eg--earth and mars inherits the parent class solar system and solar system inherits the milky way galaxy, so milky way galaxy is the top parent class for class solar system, earth and mars acquiring properties of one class into another class--main function is code reusability= we use extends to use inheritance--method overriding

eg--solar system-planets||company-employee||country-citizen||furniture-chair||instrument-musicallinstrument||electronics-smartphone||food-fruit||literature-book||building-apartment||plant-flower||operatingSystem-MobileApp||Graphics-3Dmodel,1D,2D||

**java support 3 inheritance**

**1.single inheritance**=one to one --child class is derived from only one parent class

it inherits the properties and behaviour of single parent class

**2.multi-level inheritance**=parent-child-grandchild--class that has more than one parent

living Being—Animal--Human

**3.hierarchical inheritance**=one parent many child---if a number of classes are derived from a single

parent class --vehicle-car-bike-truck

**4.multiple inheritance**=one child many parent--ambiguity in calling method and constructor

animal-live on surface – live on water

**5.hibrid inheritance**= combination of hierarchical and multiple inheritance

zoo-(land Animal – aquatic Animal)-penguin

A a=new A();==parent

B b=new B();==parent +child

A ab=new B();==parent +override method

**Polymorphism**---is the ability to perform many things in many ways--function behave differently based on the actual implementation

eg--delivery person delivers item to user -if it's a postman he will deliver the letters.-if its a food delivery boy he will deliver the food to user

**Encapsulation**---is the process of wrapping code and data together into single unit

eg--a capsule which is mixed of several medicines, the medicines are hidden data to the end user

**Abstraction**---is the process of hiding implementation details and exposes only the functionality to the user--we deal with ideas and not events--user only know what it does rather than how

eg--a driver will focus on the car functionality (start, accelerate, break, stop) he does not bother about how the function mechanism works internally

**type casting** ==changing one data type to another data type

**implicitly/winding**==byte b=12; short s=b; byte->short->int->double--small size datatype to larger

datatype--java done by it self

**explicitly/narrowing**==int i =22; short s=(short) i ; ---larger datatype to smaller datatype

we have to use syntax to done this casting

**access modifier ==** Access modifier used to define scope of method, constructor, class, main method

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Same Class** | **Same Package** | **Outside Package** |
| **Private** | TRUE | False | False |
| **default** | TRUE | TRUE | False |
| **protected** | TRUE | TRUE | Only applicable when inheritance is used |
| **Public** | TRUE | TRUE | TRUE |

private== only accessible in same class

default==accessible in same class+ same package

protected==accessible in same class+ same package + outside package if inheritance is use

public==accessible in same class+ same package + outside package

**We can give Access modifier to** ->Class, method, constructor, global variable, Main method

**Access modifier not use** -> Local variable, Static, Non-Static

If I crate private constructor, I cannot create object outside class

Default constructor’s access modifier depends on the class’s access modifier

Method access modifier is public, private

If I create private method then that method only accessible within class

**Class-** Public, default

**Global variable/constructor/method-** all

**Main method-** Public

If I create public class I can create class with different name than file name

If default class is created, I cannot create class with different name than file name

**keyword**==reserved word used in java

**Final**==restrict from changing--is non access modifier--you need to initialized a value to variable

we can initialize value in constructor but value become changeable

eg.-public final int i=50

**we can able to use final in**-=class, global variable, local variable, method

**not use in** static, non-static

if we use final in class we cannot extend class to child class/we can restrict class from inherit

variable – locally, globally

if we use in method level, we cannot change implementation

in case of object, we cannot initialize again

constructor-- available only access modifier

**Static**==memory create when class loading--used for memory management

if we want to use same variable more time then we use static

sratic keyword used for memory allocation

**super**==used to call immediate parent class constructor

any constructors first line is super/this--we cannot use both in single line/same constructor

using super we can call constructor, method,

not applicable in any static class

**this**==used to call same class constructor ----using this we can call constructor, method,

not applicable in any static class

**overriding**==covariant--method name, class name,

**overloading**==it happens in same class-method name same--parameter different or sequence change) --return type/access modifier doesn’t matter--we can use any method in this (static, final, public, protected, private) --main method can be overload--constructor can overload

**overriding**==it happens in parent-child class--method name same--parameter same and sequence same--(static, final, protected, private) cannot be override--return type must be same or co-orient--constructor cannot be override

--we cannot or main method

**Polymorphism**==one task perform in many ways—

Polymorphism is considered one of the important features of Object-Oriented Programming. Polymorphism allows us to perform a single action in different ways. In other words, polymorphism allows you to define one interface and have multiple implementations.

**Compile-Time Polymorphism in Java**==It is also known as static polymorphism. This type of polymorphism is achieved by function overloading or operator overloading.

**Method Overloading**=When there are multiple functions with the same name but different parameters then these functions are said to be overloaded.

Subtypes of Compile-time Polymorphism

1. Function Overloading==It is a feature in C++ where multiple functions can have the same name but with different parameter lists. The compiler will decide which function to call based on the number and types of arguments passed to the function.

2. Operator Overloading==It is a feature in C++ where the operators such as +, -, \*, etc. can be given additional meanings when applied to user-defined data types.

3. Template==it is a powerful feature in C++ that allows us to write generic functions and classes. A template is a blueprint for creating a family of functions or classes.

Runtime Polymorphism in Java==It is also known as Dynamic Method Dispatch. It is a process in which a function call to the overridden method is resolved at Runtime. This type of polymorphism is achieved by Method Overriding.

Subtype of Run-time Polymorphism

i. Virtual functions==It allows an object of a derived class to behave as if it were an object of the base class. The derived class can override the virtual function of the base class to provide its own implementation. The function call is resolved at runtime, depending on the actual type of the object.

**narrowing**=overloading cz =

**abstraction**==hiding initial details and showing functionality

if we know method is going to use but we know the implementation then we use abstract class

sharing essential data at essential time--it is used to achieve 0% to 100% abstraction

--abstract can be achieved by abstraction keyword and interface

in abstraction constructor are present--we cannot create object

abstract method cannot be privet, static, final

we are going to use abstract keyword in child class if we don’t want to implement all methods from parent class in child class

**interface**-interface is a blueprint of class--it is used to achieve 100% abstraction--all method in interface is by default abstract--

in case of inheritance, we use "implements" not extends---in interface constructor are not present--we cannot create object--in static we need to compulsory assign a value to variable

we need to assign value because variable is by default public/static/final/

----class implements interface— (interface extends interface) java not allow interface implements class

**marker interface**--- it is a empty interface which does not conduct any method, variable etc..

--it is used to give extra information to JVM

eg=public interface test{ }

some marker interfaces=serialization, remote, randomAaccess, cloneable, etc.

**Encapsulation**---is the process of wrapping code and data together into single unit

eg--a capsule which is mixed of several medicines, the medicines are hidden data to the end user

we can achieve incantation by creating private data members(variable)

we not use inheritance in adv java because it create tightly - cupelling...(is a relation)

if one service is down all service are down it reflect on all application

we are going to use "has-a-relation"

hashcode, tostring, equals,

global variable also know as instants variable

pojo class plane old java object classes

**Aggregation**[=-- is a has a relationship---in this we can create reference entity—

If a class have an entity reference, it is known as Aggregation. Aggregation represents HAS-A relationship. Aggregation is a special form of Association, which is defined as an unidirectional one way connection between classes or entities

int id; String name; Address address;//Address is a class

**Array**==collection of similar type of data--it is a non primitive data type--entity store in a single unit

--array used for memory management--size declare is necessary --default value of full array is Null and single data’s value depend on declaration(int-0,string-null..etc) --array is linier data structure

--size of array is fixed it depend on declaration—

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.To declare an array, define the variable type with square brackets:

String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

**Array Length--**To find out how many elements an array has, use the length property:

System.out.println(cars.length);

**Multidimensional Array in Java**=In such case, data is stored in row and column based index (also known as matrix form).

Syntax=dataType[][] arrayRefVar; (or) dataType [][]arrayRefVar; (or) dataType arrayRefVar[][];

==int[][] arr=new int[3][3];//3 row and 3 column

**serialization** == serialization in java help to transport the code from one jvm to another ---it is used to converting the state of an object into a byte stream.--Serializable is marker interface it is used to mark java classes so that the object of these classes may get a certain capabilities implemented by the class which needs to persisted

for serializing the object we call the writeObject() method of ObjectOutputStream class,

**Deserialization**== Deserialization is the process of converting object stream to actual java object to be used in program.--it is used to converting byte stream into state of an object.

Deserialization we call the readObject() method of ObjectInputStream class--

**Shallow Copy--** Creating a new reference that points to the same memory location. This is also called a Shallow copy.

When we do a copy of some entity to create two or more than two entities such that changes in one entity are reflected in the other entities as well, then we can say we have done a shallow copy.

--In shallow copy, new memory allocation never happens for the other entities, and the only reference is copied to the other entities. The following example demonstrates the same.

**Deep Copy--** Creating a copy of object in a different memory location. This is called a Deep copy.

When we do a copy of some entity to create two or more than two entities such that changes in one entity are not reflected in the other entities, then we can say we have done a deep copy.

--In the deep copy, a new memory allocation happens for the other entities, and reference is not copied to the other entities. Each entity has its own independent reference. The following example demonstrates the same.

**diffrence==**

|  |  |
| --- | --- |
| Shallow Copy | Deep Copy |
| It is fast as no new memory is allocated. | It is slow as new memory is allocated. |
| Changes in one entity is reflected in other entity | Changes in one entity are not reflected entity. |
| Shallow Copy stores the references of objects to the original memory address. | Deep copy stores copies of the object’s value. |
| Shallow Copy reflects changes made to the new/copied object in the original object. | Deep copy doesn’t reflect changes made to the new/copied object in the original object. |
| Shallow Copy stores the copy of the original object and points the references to the objects. | Deep copy stores the copy of the original object and recursively copies the objects as well. |

**Exception**==an event, which occurs during the execution of a program, that disrupts the normal flow of the program’s instructions.

There are many typical causes for exceptions in Java, including:

1.-Invalid input data 2.-Requests for missing or non-existent files 3.-Exceeding memory limits for the Java Virtual Machine (JVM) 4.- Code errors

**Exception Handling**==The Exception Handling in Java is one of the powerful mechanism to handle the runtime errors so that the normal flow of the application can be maintained.

Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

**1) Checked Exception**

The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions. For example, IOException, SQLException, etc.

**2) Unchecked Exception**

The classes that inherit the RuntimeException are known as unchecked exceptions. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc.

**3) Error**

Error is irrecoverable. Some example of errors are OutOfMemoryError, VirtualMachineError, AssertionError etc.

Java provides five keywords that are used to handle the exception. The following table describes each.

Keyword==Description

1) **try**==The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally.

2) **catch**==The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.

3**) finally**==The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not.

4) **throw**==The "throw" keyword is used to throw an exception.

5**) throws**==The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature.

---Built-in exceptions are the exceptions that are available in Java libraries. These exceptions are suitable to explain certain error situations. Below is the list of important built-in exceptions in Java.

1.ArithmeticException: It is thrown when an exceptional condition has occurred in an arithmetic operation.

2.ArrayIndexOutOfBoundsException: It is thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.

3.ClassNotFoundException: This Exception is raised when we try to access a class whose definition is not found

4.FileNotFoundException: This Exception is raised when a file is not accessible or does not open.

5.IOException: It is thrown when an input-output operation failed or interrupted

6.InterruptedException: It is thrown when a thread is waiting, sleeping, or doing some processing, and it is interrupted.

7.NoSuchFieldException: It is thrown when a class does not contain the field (or variable) specified

8.NoSuchMethodException: It is thrown when accessing a method that is not found.

9.NullPointerException: This exception is raised when referring to the members of a null object. Null represents nothing

10.NumberFormatException: This exception is raised when a method could not convert a string into a numeric format.

11.RuntimeException: This represents an exception that occurs during runtime.

12.StringIndexOutOfBoundsException: It is thrown by String class methods to indicate that an index is either negative or greater than the size of the string

13.IllegalArgumentException : This exception will throw the error or error statement when the method receives an argument which is not accurately fit to the given relation or condition. It comes under the unchecked exception.

14.IllegalStateException : This exception will throw an error or error message when the method is not accessed for the particular operation in the application. It comes under the unchecked exception.

**TRY**==The try block is used to specify a block of code that may throw an exception.

Alongside code that could trigger an exception, try blocks can also contain code that doesn’t cause exceptions.

Programs can contain more than one try block. Additionally, it is possible to have one try block inside of another try block.

**CATCH**==The catch block is used to handle the exception if it is thrown.

Every try block needs at least one catch block.

Finally blocks are an optional addition to try/catch blocks.

Finally blocks are used for code that you want to be run no matter what happens exceptionwise with your code.

They are the last thing that executes and they will execute every time you run your program.

One use of a finally block would be to make sure a file is closed at the end of a program.

Try blocks can have multiple catch blocks.

Catch blocks must come directly after the try block they are paired with.

Catch Block Contents

The contents of a catch block are what you want to happen when an exception occurs. you can either print a message or have code that does something to fix the error

**Catch Block Messages** ( Displaying a Description of an Exception )

1.Using toString()

Throwable overrides the toString( ) method so that it returns a string containing a description of the exception. You can display this description in a println( ) statement by simply passing the exception as an argument.

2.getMessage(): returns an error message that’s specific to each type of exception. Pair this with a println statement.

3.printStackTrace() : prints the type of error and where it occurred in the program These methods are especially helpful for debugging and finding the reason for an exception

**FYNALLY**==The finally block is used to execute the code after the try and catch blocks have been executed.The finally keyword is used to execute code (used with exceptions - try..catch statements) no matter if there is an exception or not.

1. Exception occurs in try block and handled in catch block

2. Exception occurred in try-block is not handled in catch block

3. Exception doesn’t occur in try-block: In this case catch block never runs as they are only meant to be run when an exception occurs. finally block(if present) will be executed followed by rest of the program.

----Control flow in try-finally--

1. Exception raised: If an exception has occurred in the try block then the control flow will be finally block followed by the default exception handling mechanism.

2. Exception not raised: If an exception does not occur in the try block then the control flow will be finally block followed by the rest of the program

**The try block contains the code that may throw an exception.**

**The catch block catches and handles specific types of exceptions.**

**Multiple catch blocks can be used to handle different types of exceptions.**

**The finally block is optional and is always executed, whether an exception occurred or not.**

**Pipe**==the pipe | operator can be used in the catch block to catch multiple exception types in a single block. This feature is known as "multi-catch" and was introduced in Java 7.

the pipe | operator can be used in the catch block to catch multiple exception types in a single block. This feature is known as "multi-catch" and was introduced in Java 7.

**THROW**==The throw statement allows you to create a custom error.

The throw statement is used together with an exception type. There are many exception types available in Java: ArithmeticException, FileNotFoundException, ArrayIndexOutOfBoundsException, SecurityException, etc:

An **exception** is an undesired event that disrupts the normal flow of a program and may lead to its termination.

In Java, **throwable** entities can be categorized into two types: errors and exceptions.

**1.Errors:**

System errors, such as StackOverflowError, cannot be handled in Java.

**2.Exceptions:**

Exceptions arise from user mistakes and are further divided into checked and unchecked exceptions.

**A.Checked exceptions (compile-time):**

Examples include ClassNotFoundException (when a class is not found in the given class path) and IOException (when a class fails to load or a file is not found).

**B.Unchecked exceptions (runtime):**

Examples include ArithmeticException (invoked by arithmetic errors like division by zero), NullPointerException (when operations are performed on null variables),

NumberFormatException (when converting strings to numbers or vice versa), InputMismatchException (when invalid values are obtained by a scanner, e.g., assigning a string to a roll number),

ArrayIndexOutOfBoundsException (when operations exceed the assigned array length), StringIndexOutOfBoundsException (similar to ArrayIndexOutOfBoundsException but for strings), and NegativeArraySizeException (when trying to assign a negative value to an array).

Exception handling can be implemented using the try-catch block.

We can used **try catch** in any inner blocks->main Mthod ,method, constructor,

In try we write doubtable code

In catch we handel exception

Java introduced **multiple catch** blocks in the 1.7 update, allowing the handling of different exceptions.==try{}catch{}catch{}

**Ruls**-we need to give child exception in 1st catch then its parent exception, The hierarchy should be followed, giving child exceptions before parent exceptions. --try{}catch(arithmaticException e){} catch(Exception e){}

Catch msg==sop(e); e.getMessage(); e.printStackTrace(); e.toString();

To handle multiple exceptions in a single line when the same execution is required, the **pipe operator** is used.

**Innertrycatch**—try{}try{}catch{}catch{} –

**Finally—** The finally block is used to close resources and always gets executed. The System.exit() method can prevent the execution of the finally block.

Final---it is a keyword

Finally—is is a block

Finalize-it is a method

**try(resources**)— try with resources (introduced in Java 1.7) simplifies resource closure for autocloseable methods like connections or scanners.

Closeble-scanner

**Throws**==used to declare exception, used in constructor,method..

The throws keyword is used to declare exceptions in constructors and methods, converting compile-time exceptions to runtime exceptions.

If we don’t want to handle exception then we used throws

**Throw**=used to throw exceptions explicitly and to throw custom exceptions.

**custumException**==used to create custome exeption

**rules in constructor**

If a parent class constructor throws an exception, it is compulsory for the child class constructor to declare the same or a subclass exception

If the parent class constructor throws a runtime exception, there is no requirement for the child class constructor to declare an exception.

**Rules in methods**

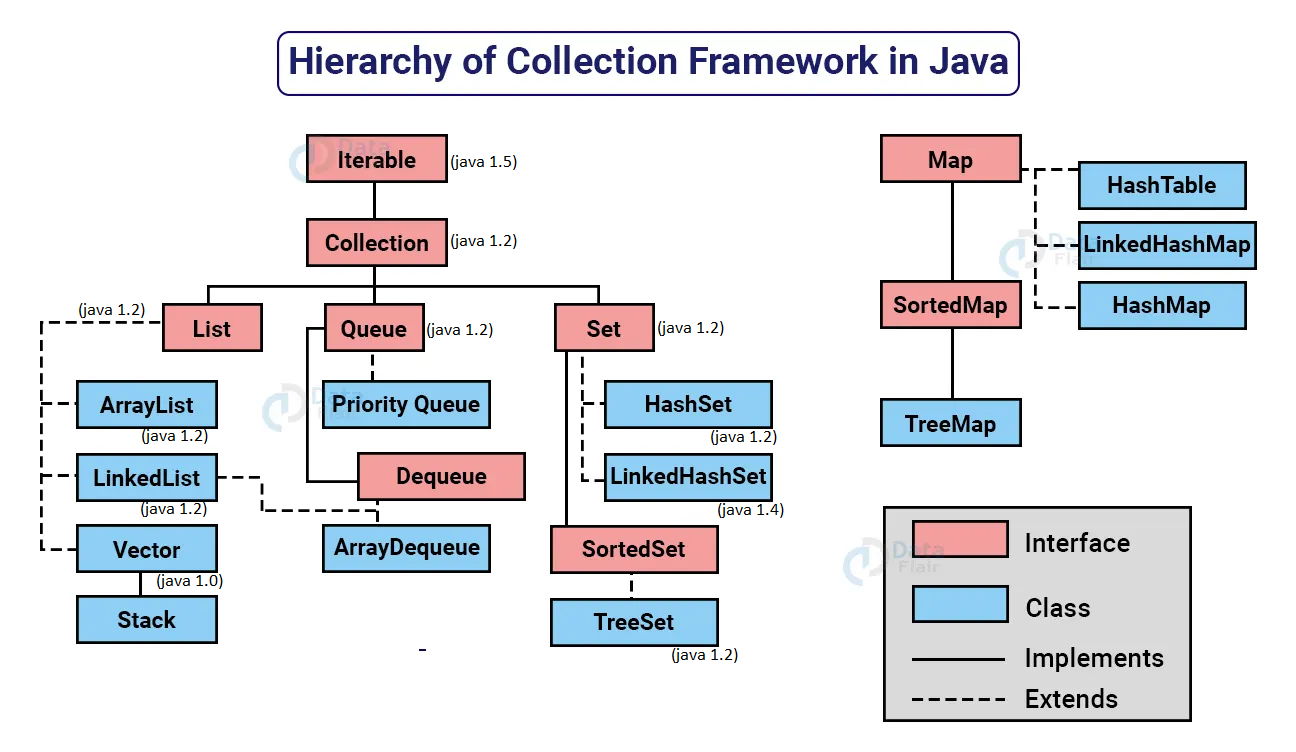
When it comes to method overriding, there is no need to declare the same exception in the overriding method. If an exception is required, it should be a subclass of the parent exception.

If there is a need to throw an exception in a method, it is advisable to throw the parent exception.

**Array**== An array is a collection of similar types of data, allowing the storage of multiple objects in a single unit. It possesses a fixed size,--- homogenise – no predefine method – is efficient for memory management, and offers faster data access. Arrays can store both primitive and non-primitive data.

**Collection=** a collection is a group of similar or different data types, capable of storing multiple objects in a single unit.--size not fixed (Growable) support homogenous and heterogeneous data, provide predefined methods, are relatively slower compared to arrays, and are not optimized for memory management. Collections exclusively contain objects.

**Framework**— The Java Framework is a collection of predefined classes and interfaces introduced in version 1.2 and located in the java.util package.



**List =** List, a type of collection,

-- maintains the insertion order, allows multiple null values, and Duplicate allowed

**Generics =** introduced in Java 1.5, are used to specify the data type stored in lists.

Array Vs Collection

|  |  |
| --- | --- |
| Array | Collection |
| Arrays are fixed in size that is once we create an array we can not increased or decreased based on our requirement. | Collection are growable in nature that is based on our requirement. We can increase or decrease of size. |
| Write in memory Arrays are not recommended to use. | Write in memory collection are recommended to use. |
| With respect to performance Arrays are recommended to use. | With respect to performance collection are not recommended to use. |
| Arrays can hold only homogeneous data types elements. | Collection can hold both homogeneous and heterogeneous elements. |
| Arrays can hold both object and primitive data type . | Collection can hold only object types but not primitive datatypes such as int, long, short, etc. |
| Array don’t have pre define methods | Collection have predefine methods |

| **Feature** | **Array** | **Collection** |
| --- | --- | --- |
| Size | Fixed size | Variable size (growable) |
| Memory Usage | Not recommended | Recommended |
| Performance | Recommended | Not recommended |
| Data Types | Homogeneous and heterogeneous | Only object types, no primitive data types |
| Predefined Methods | Limited | Abundant predefined methods |

**Under Line Data Structure** – ULDS means how its stored the data internaly

**Arraylist()**-> Arraylist’s ULDS is Growable array –it is liner data Structure

--Default size of array list is fist is 0 after adding one element its become 10

-- it will change when load factor fill 0.75 % then its become 20

Load factor =total element store / size

(Big O) O(n) complexity appear when we try to add or delete element

Array list is suitable for insertion and retrieval of data. Use an ArrayList for storing and accessing data

Syntax:- ArrayList<String> cars = new ArrayList<String>();

**cars.add("Volvo");** // to add elements to the ArrayList , use the add() method:

**cars.get(0);**// use the get() method and refer to the index number:

**cars.set(0, "Opel");**// use the set() method and refer to the index number:

**cars.remove(0);//** use the remove() method and refer to the index number:

**cars.clear();**//To remove all the elements in the ArrayList, use the clear() method:

**cars.size();**//To find out how many elements an ArrayList have, use the size method:

**Collections.sort(cars);**//Another useful class in the java.util package is the Collections class, which include the sort() method for sorting lists alphabetically or numerically:

**For loop:-** (int i=0; i<cars.size; i++) {sop(“cars.get(i)”):}

**For Each**:- (String str:cars) { sop(str);}

**Iterator**:- Iterator it = cars.iterator(); while (it.hasNext()) sop(it.next();

**Enumeration(1.0)**:- Enumeration<String> e= Collections.enumeration(cars);

while (e.hasMoreElements()) Sop(e.nextElement());

listIterator+= in this we can add remove get next value get previos value

**Linkedlist ->** Default size is not define – before java 1.8 uddate default size is 12

--ULDS is Doubly linkedlist it is a non linear data structure

In Linkedlist we store data node vies – address of previous data | data |address of next data

Use an LinkedList to manipulate data.

Syntax- LinkedList<String> cars = new LinkedList<String>();

**Stack –** ULDS is stack – introduce in java 1.0 update –Store in LIFO manner

RandomAccess not possible—eg.-photo gallery store data in LIFO manner

**Arraylist**, a specific implementation of the List interface, utilizes a growable array with a default size of 10, dynamically adjusting when the load factor reaches 75%. It operates with a time complexity of O(n) for add or delete operations, making it suitable for insertion and retrieval of data.

**Linkedlist**, another List implementation, lacks a predefined default size, using a doubly linked list as its underlying data structure. It stores data in nodes, each containing the address of the previous data, the data itself, and the address of the next data.

**Stack**, introduced in Java 1.0, follows the Last In First Out (LIFO) principle, making random access impossible. It is useful for scenarios such as storing a photo gallery in a LIFO manner.

**The enhanced for-each loop** was introduced in Java 1.5.

The **Set** interface was introduced in Java 1.2.

**HashSet:**

Introduced in Java 1.2, a HashSet is a part of the Set family.

Does not preserve insertion order.

Allows only one null value; multiple null values are not permitted.

Enforces uniqueness (no duplicates), and sorting is not used.

When we try to add dublicate values it will print ‘faulse’

Insertion depends on hashcode – UDDS==hash table---initial capacity is 16

Hashset has 4 constructor 1.default 2.(initial capacity) 3.(IC + loadfactor) 4.(collection c)

We can add homogeneous and heterogenies

**LinkedHashSet:**

Introduced in Java 1.4, a LinkedHashSet extends both Set and HashSet.

Preserves insertion order.

Allows only one null value; multiple null values are not permitted.

Enforces uniqueness (no duplicates), and sorting is not used.

UDDS==hash table and doubly linkedlist /linkedlist

We can add homogeneous and heterogenies

**TreeSet:**

Introduced in Java 1.2, TreeSet is part of the hierarchy: Set -> SortedSet -> NavigableSet -> TreeSet.

Does not preserve insertion order. Depends on sorting order –num=ascending / char- alphabetical

Does not allow null values; there can be no multiple null values.

Enforces uniqueness (no duplicates) and uses sorting in ascending order.

We can add homogeneous and comparable data

It is advised not to use null values in TreeSet because it internally uses the compareTo method for comparison, and introducing a null value may result in a NullPointerException.

We not use null value in treeset because internally treeset used compareto method to compare veriable and if there is null value it throwsruntime exception nullpointerException

**FailFast –** is a concept that occurs during iteration in any class and try to add and remove any object from list In such cases, it throws a ConcurrentModificationException.

**FailSafe –** in this using CopyOnWriteArrayList and CopyOnWriteArraySet, as well as concurrent hash map methods, to avoid exceptions during modification.

**Collections** == is a utility class in Java that contains static methods for various collection operations:

Collectons.addall(list, 33,44,55,44);

Collectons.frequency(list, 44); //return type int and count how many time occurs

Collectons.sublis, Collectons.reverorder, Collectons.rotat, Collectons.shuffel, Collectons.min, Collectons.max, Collectons.mean, Collectons.replaceall , Collectons.unmodifiablelist

**Final** – When an object is declared as final, it cannot be reinitialized or pointed to a different location

**Unmodifiablelist** –we can reinitialize, we can not modify list --the list itself cannot be modified, but it can be reassigned.

Collection.sort(list);// for sorting we can used two interface

|  |  |
| --- | --- |
| comparable | comparator |
| It is a interface | It is a interface |
| In this we used Methods colled compareTo() | In this we used Methods colled compare() |
| In this we need to modify existing class | In this we need to add another classes |
| It used in case of default, natural sorting order | It used customice sorting order |
| In this we used data like integer,String | In this we used classes as data |

**Map** == map does not extends any interface –map comes under collection framework –but not follow collection hierarchy

Introduce in 1.2 --

Insertion order not preserve

In map we store data in key and values in pair format

One key +value =entry

Entry-entry is a inner interface—it is store in map interface

**Rules** of Map.key—

Duplicate keys are not allowed, and adding a duplicate key will override the existing value.

One null key is allowed, and adding a second null key will also override the value.

Map.value—Duplicate value allowed

Multiple null also allowed

**HashMap--** Introduce in 1.2 –it is a class -- Insertion order not preserve – extends abstractmap

ULDS= HashTable –default size is 16—internal of hashset is hashmap

Rules of Map.key—

Duplicate key not allowed if added value will be override

1 null key is allowed when we add two nulls value will be override

Map.value—Duplicate value allowed

Multiple null also allowed

**Linkedhashmap –** introduce in java 1.4 – extends hashmap ---insertion order preserved

Dublicate key not allowed its overrides value

**Treemap**-introduce in 1.2 – it is used for sorting, and sorting depends on the keys.

ULDS -BlackOckTree **BlackRedTree**

Null key not allowed –duplicate values – Keys must be comparable

Null value allowed

We can not use iterator directly –we need to convert it into collection

There are three ways

1.map.keyset()—in this we convert tree into key’s set –then perform iteration

2.map.entryset()—in this we convert tree into entry’s set –then perform iteration

3.map.valueset()—in this we convert tree into value’s set –then perform iteration

Ehen we try to get keys value which is not store then values of that key printed as NULL

**Internal structure of hashset**

Hashset h=new Hashset();//hashset internal is hashmap

h.add(“abc”);//->

internal -> put(“abc”,PRESENT) // PRESENT is predefined veriable//final static variable

**Internal structure of Linkedhashset**

h.add(“abc”);//-> Linkedhashset internal is Linkedhashmap---2 veriable before ,after(next)

internal -> put(“abc”,PRESENT)

Hashset, LinkedHashset, HashMap,…etc all hash’s insertion depends on Hashcode

**HashMap**🡪defauult value is 16,internaly work on hash table--- in this buckets created

(array)table[] = 0-15

**Internal structure**

1.h.put(1,”abc”);

\\Hashcode generation

2.all entryes hashcode gget generated(50)

[\\ then](file:///\\then) used indexfor() method for use formula

3.Hashcode % Highestindex //50%15

\\ reminder is 5

Entery get store in bucket index 5

When we try to add null key (null,”xyz”) then its first call put for null key method ->table[0]

When 1 bucket fills upto 8 then its datatype changes to binary tree

**? what if equals method always return “false” ?**//its not happends logically

Ans—entry get store without overriding ,value get added (1,abc)(1,hhh)(1,pqr)(1,mnq)

**Hashing** means Hashcode generation

Equal method check content only

When we add duplicate then used equalto method to check key is same ot not

If key is not same colution happends other wise override first value

Rehashing -when all bucket get fill then highest index value become double

**WeekHashMap**—when entry is null then its go to garbage collector then GC colled finalized method to remove object , weekhashmap called garbage collector directly when null value appear

**HashMap** – Noy used GC to remove null value –hashmap dominets gc that’s why gc not called in Hashmap  
HashMap dominates garbage collection (GC) because it doesn't directly invoke the GC to remove null values. However, WeakHashMap behaves differently; when entries become null, it directly invokes the garbage collector to remove those entries

**Class Loaders:**

Class loaders in Java are responsible for loading Java classes into memory at runtime. There are three built-in class loaders in Java: Bootstrap class loader, Extension class loader, and System class loader.

Class loaders follow a hierarchical delegation model, where each class loader first delegates the class loading request to its parent before attempting to load the class itself.

Class loaders enable dynamic class loading, which allows Java applications to load classes dynamically based on runtime conditions.

**Object Class Methods:**

The Object class is the root class for all Java classes and provides several methods that can be overridden by subclasses.

Some of the commonly used methods of the Object class include equals(), hashCode(), toString(), getClass(), wait(), notify(), and notifyAll().

These methods are overridden in subclasses to provide custom behavior based on the requirements of the application.

**Reflection in API:**

Reflection in Java is a feature that allows inspection of classes, interfaces, fields, and methods at runtime.It enables dynamic instantiation of classes, invocation of methods, and access to fields, even if they are private.

Reflection is useful for building frameworks, libraries, and tools that need to interact with unknown or dynamically loaded classes.

Some commonly used classes in Java Reflection API are Class, Method, Field, Constructor, etc.

Java Reflection is a process of examining or modifying the run time behavior of a class at run time. Java Reflection API is used to manipulate class and its members which include fields, methods, constructor, etc. at runtime.

**JVM Architecture:**

JVM (Java Virtual Machine) is a crucial component of the Java Runtime Environment (JRE) that executes Java bytecode.The main components of JVM architecture include:

Class Loader: Loads classes into memory dynamically.

Runtime Data Areas: Includes method area, heap, stack, PC register, and native method stack.

Execution Engine: Executes Java bytecode by interpreting it or by using Just-In-Time (JIT) compilation to native machine code.

Native Interface: Enables interaction between Java code and native applications or libraries written in other languages.

JVM provides platform independence by abstracting hardware dependencies and providing a consistent runtime environment for Java programs across different platforms.

**How to Create Object in Java**

The object is a basic building block of an OOPs language. In Java, we cannot execute any program without creating an object.

**Using new Keyword:-** When we create an instance of the class by using the new keyword, it allocates memory (heap) for the newly created object and also returns the reference of that object to that memory. The new keyword is also used to create an array.

**Using clone(**) method:- The clone() method is the method of Object class. It creates a copy of an object and returns the same copy. The JVM creates a new object when the clone() method is invoked. It copies all the content of the previously created object into new one object. Note that it does not call any constructor. We must implement the Cloneable interface while using the clone() method. The method throws CloneNotSupportedException exception if the object's class does not support the Cloneable interface. The subclasses that override the clone() method can throw an exception if an instance cannot be cloned.

**Using newInstance()** method of the Class class:- The newInstance() method of the Class class is also used to create an object. It calls the default constructor to create the object. It returns a newly created instance of the class represented by the object. It internally uses the newInstance() method of the Constructor class.

**Using newInstance()** method of the Constructor class:- It is similar to the newInstance() method of the Class class. It is known as a reflective way to create objects. The method is defined in the Constructor class which is the class of java.lang.reflect package. We can also call the parameterized constructor and private constructor by using the newInstance() method. It is widely preferred in comparison to newInstance() method of the Class class.

**Using Deserialization:-** serialization is the process of converting an object into a sequence of byte-stream. The reverse process (byte-stream to object) of serialization is called deserialization. The JVM creates a new object when we serialize or deserialize an object. It does not use constructor to create an object. While using deserialization, the Serializable interface (marker interface) must be implemented in the class.