**FULL STACK JAVA NOTES BY VIRAT**

TRAINING SESSION BY NITHIN ( 27-05-2024)

C

C is a general-purpose, structured, procedural, and high-level programming language developed by Dennis MacAlistair Ritchie in 1972 at Bell Laboratories. The successor of the C language was CPL (Combined Programming Language). It is mainly used for system programming such as to develop the operating system, drivers, compilers, etc

Features of C Language

* Machine independent and portable
* Modern Control Flow and Structure
* Rich set of operators
* Simple, Fast, and efficient
* Case-sensitive
* Low memory use
* Easily extendable
* Statically-typed

C++

C++ is a general-purpose programming language created as an extension of the C programming language by Bjarne Stroustrup in 1979. It is known for combining both high-level and low-level language features, providing developers with the tools to manage complex software development efficiently and effectively.

**JAVA**

Overall, Java was introduced to address the needs of developers for a portable, robust, secure, and easy-to-use programming language, and its success over the years is a testament to its effectiveness in fulfilling these requirements.

Feature C C++ Java

Paradigm | Procedural Procedural and Object-Oriented Object-Oriented

Compilation | Compiled Compiled Compiled to bytecode, interpreted by JVM

Platform Dependence | Platform-dependent Platform-dependent Platform-independent (via JVM)

Memory Management | Manual (malloc/free) Manual (malloc/free) and Automatic (constructors/destructors) Automatic (Garbage Collection)

Pointers Yes Yes No (uses references instead)

Multiple Inheritance| No Yes (using classes and virtual inheritance) No (achieved through interfaces)

Operator Overloading| No Yes No

Exception Handling | No Yes Yes

Standard Library | Standard C Library Standard C++ Library (STL) Rich Standard Library

Support for Threads | No Library-dependent (e.g., pthreads) Built-in support (java.util.concurrent)

Syntax for |

Object-Oriented Programming Not supported Supported (class, object, inheritance) Supported (class, object, inheritance)

Memory Safety | Low (prone to pointer issues) Medium (smart pointers, but still manual control) High (automatic garbage collection)

Templates/Generics | No Yes (Templates) Yes (Generics)

File Extensions | .c .cpp, .h .java, .class

Main Use Cases | System/embedded programming System/application programming, game development Enterprise applications, Android development

Developed By Dennis Ritchie (1972) Bjarne Stroustrup (1983) James Gosling (1995)

Namespace Support No Yes Yes (package)

Method Overloading No Yes Yes

Method Overriding No Yes Yes

Support for GUI No Library-dependent (e.g., Qt) Built-in libraries (AWT, Swing, JavaFX)

Concurrency No built-in support Library support Built-in support (java.util.concurrent)

Inheritance Single inheritance Multiple inheritance (with restrictions) Single inheritance (multiple via interfaces)

Memory Access Direct Direct Restricted (managed by JVM)

Performance High (closer to hardware) High (with added abstraction) Moderate (due to JVM overhead)

Binary Size Smaller executables Generally larger than C Platform-independent bytecode

**Garbage collection**

Garbage collection (GC) is a memory recovery feature built into programming languages such as C# and Java

JDK (java development kit)

jdk (java development kit)---> translator or interpretor ---> system dependent asks details

s1 and s2 are two computers jdk act as interpretor sending or receiving data.

That’s why java is platform independence because of JDK.

Standalone application---> without internet i.e, inbuilt...ex:games

deprecated---> in the upcoming version removing methods,

JAVA 8

* **all the other software features are in J8.**
* **consumes lines of code.**
* ***using internet connection----> required database..servers(sql) its a empty space..handle the data***

RDMS ( **RELATIONAL DATA BASE MANAGEMENT SYSTEM)**

A Relational Database Management System (RDBMS) is a type of database management system that stores data in a structured format, using rows and columns. This allows for efficient organization, retrieval, and management of data. RDBMSs use Structured Query Language (SQL) for querying and maintaining the database. Here are the key components and characteristics of an RDBMS

data in table format..another also relation with another table

NOSQL

NoSQL, or "Not Only SQL," refers to a category of database management systems (DBMS) that diverge from the traditional relational database management systems (RDBMS) model. Unlike RDBMS, which store data in tables with rows and columns and use SQL for querying, NoSQL databases are designed to handle large volumes of unstructured, semi-structured, or structured data in a more flexible and scalable manner.

> Backup---> (when storage increase performance decreases...need backup)

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

When implementing a backup strategy, it's essential to consider factors such as data retention policies, recovery time objectives (RTOs), recovery point objectives (RPOs), security, compliance requirements, and the overall complexity of managing and maintaining backups. Additionally, regular testing and monitoring of backups are crucial to ensure their reliability and effectiveness in case of data loss or system failure

Webapps:(runs with internet)

SERVLET:

A servlet is a Java programming language class used to extend the capabilities of servers that host applications accessed by means of a request-response programming model. Servlets are commonly used to process or store a Java class in web applications, generating dynamic content. Here are the key points to understand about servlets.

servlets--> (main roots)--->spring--->spingboot

JDBC (Java Database Connectivity):

>>>Java-based API that enables Java applications to interact with a variety of databases. It provides methods for querying and updating data in a database. Here are the key concepts and components of JDBC.

advanced version is HYBERNET

**url(universe resource locator) sending requests and response**.

Spr6ing--->plenty of modules---->specific functionality

spring batch deals more data

authorization--->specific person

Authorization is a critical aspect of security in applications, responsible for determining what actions an authenticated user is permitted to perform. It ensures that users only access resources and perform actions that they are allowed to, based on their roles, permissions, or other attributes.

|

[spring security]

|

**authentication**---> existing candidate

springcore--->

springmvc and jpa

spring actuators

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

**FRONTEND**

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

> HTML5,CSS3,BOOTSTRAP5,JAVASCRIPT,REACT,ANGULAR,TYPESCRIPT AND NODE.

BASICS OF DEVEOPS

**CORE JAVA**

* Introduce the java because of
* first program is OAK Renamed JAVA
* ILAND NAME BARLE COFFEE ISLAND

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

***1.DATATYPES:***

***2.VARAIBLES:***

***3.OPERATORS:***

***4.CONTROLL STATEMENTS***

***5.STRINGS:***

***6.ARRAYS:***

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

DATATYPE:

* data may be file or image or text...
* which type of data to be accessed
* super class of java is object

**there are two types of datatypes:**

**1.primitive(8):**

those have non methods---->CHAR(2),INT(4),FLOAT(4),LONG(8),DOUBLE(),BOOLEAN(),SHORT(2),BYTE()

CHAR--> single character i.e , single alphabate ...write in single quotes...1byte storage

INT---> deal with numbers

* **round numbers are without decimal** --->1,3
* **>float and double are decimal numbers--->1.2**

**float** - 8bytes

>**BOOlEAN** ---1byte, true or false(0 or 1)

class GFG {

// Main driver method

public static void main(String args[])

{

// Creating and initializing custom character

char a = 'G';

// Integer data type is generally

// used for numeric values

int i = 89;

// use byte and short

// if memory is a constraint

byte b = 4;

// this will give error as number is

// larger than byte range

// byte b1 = 7888888955;

short s = 56;

// this will give error as number is

// larger than short range

// short s1 = 87878787878;

// by default fraction value

// is double in java

double d = 4.355453532;

// for float use 'f' as suffix as standard

float f = 4.7333434f;

// need to hold big range of numbers then we need

// this data type

long l = 12121;

System.out.println("char: " + a);

System.out.println("integer: " + i);

System.out.println("byte: " + b);

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

System.out.println("float: " + f);

System.out.println("double: " + d);

System.out.println("long: " + l);

}

}

Output:

------

char: G

integer: 89

byte: 4

short: 56

float: 4.7333436

double: 4.355453532

long: 12121

2.non primitive---> all the non primitive objects are non primitive

predefined clasess--->string,array,date,map

usedefined classes--->user defined

BYTE<char<boolean<int<float<long<double<

**TYPE CASTING**:

>> changing one data type to another data datatype

1.implicit(widening)casting:no data loss,no warning--->short term data to long term data(lowerdatatype to higher datatype)

2.explicit(narrowing)casting:data loss,warnings----->higher data to lower data(higher to lower)

**> Boolean is not typecasting.. (for interview qs)**

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

**Variables**

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

* memory is there and allowcation is there..what the values are inserted into them called variables.
* based on the scope:
* 1.Global variables: static is present
* 2.Local variables: without static is not present
* declaration and initialization:
* declaration declares only once..duplicate declation is not possible

**syntax:**

**accessmodifier datatype refence;**

ex:

public int a;

a=10;

a is constant

>> = --> is assignment operator.

rht datatype must be equal to lft side datatype.

ex:a=20+30;----> expression

accessbility:

-----------

static and instance

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

> static is a keyword --- without creating object..is about class level

> static keyword is not present i.e, instance...object level

**KEYWORDS**: predefined words so that particular functinality...it will do automatically

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

**project structure**

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

**src-->java--->test---->create n number packages--->inside classes are there**

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

**accessmodifiers**--->it will restricted classes and objects

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

1.default--->we are not write any modifires as default

2.public---->anywhere we use...

3.private---> accessed data with in that class only

4.protect--->it is also same upto class only inheritance class to another package level

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

SYNTAXES

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

accessmodifier class classname{

statements;

}

class-->keyword,accessmod-->public

EX:

public class Man{

statements;

}

class is not a reality its assumption,blue print

ex:prepare TEA----> processs is individual like stove and variables are ingridients like milk,powder

object: its a reality one

syntax:

-------

Classname a = new Classname();

ex:

Test t = new Test();

()---> constructor

t---> reference

new---> keyword

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

28-05-2024

**Main Method**:

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

**// TODO Auto-generated method stub**

**}**

OR

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

**// TODO Auto-generated method stub**

**}**

**main method is entrance** 🡪 WITHOUT MAIN METHOD WE CANT RUN JAVA PROGRAM

* public-access modifier
* static--without creating object
* void-return type
* main-name
* ()-method declaration

**camel case** :

* **smaller-variables ex:(niThin)**
* **capitals-class or interface ex: Class ,Virat**
* **JVM is to run the main method**

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

**method signature**:

**Accessmodifier returntype methodname(){**

**---------------> method body**

**methodname(); --------------> method call**

**}**

System.out.println("WELCOME TO JAVA");

**System--> class,. ----> dot operator(inside the class going),out--> variable ,println---->method**

int a =10;

int b=20;

int c= a+ b;

public void add(){-------------------------> global variables ouside method declare

return c; -----------------------------------> inside method

}

output:30

note:

====

**> Global variables are default values**

**> local variables are not default variables**

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

* default values...
* numeric--->0
* decimal---->0.0
* character--->/000

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

static always execute once then instance will come

**KEYWORDS:**

predefined words functionality for key

private int private variable;

public int public variable;

protected int protected variable;

**OPERATORS**

* Arithmetic operators: +,-,\*,%,\
* comparison operators: <,>,=,==,!=,
* logical operators: more than one condition use logical operators----> &&, || , !
* increment and decrement operators: ++i, i++,--i,i-- (pre and post)

**Controll statements**:

if i want to skip some of the lines or looping ..

**conditional statements**: based on some condition

* if,
* if..else
* else..if
* nested if
* switch

**ex**:

**int a=10;**

**int b=20;**

**if(a>b){**

**------------------>true block**

**}else{**

**------------------->false block**

**}**

iterative statements:

* **for**
* **while**
* **do..while**

if(){

}else{

if(){

}

}

**Switch Statements:**

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

The switch statement in Java is a control flow statement that allows you to execute one block of code out of many possible blocks based on the value of an expression. It's similar to a series of if-else-if statements but can be more concise and easier to read, especially when dealing with multiple conditions that depend on the same variable or expression.

**Syntax:**

Switch(){

case1;

break;

case2;

break;

....

.....

defaultcase;

}

Switch(){

case1;

System.out.println(“I am”);

break;

case"monday";

break;

case2:

break;

}

**LOOPING statements**:

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

always condition return true or false....

while(){

------------------> until codition true

}

dowhile:

do{

-------------------🡪 first print the lines

}while(){

----------------🡪 write the condition

}

Syntax:

======

**for(initialization;condition;increment or decrement){**

**}**

ex:

----

int i=1;

for(i=1;i<11;i++){

system.out.println(i);

}

Break and Continue:

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

**Ternary Operator**:

**(condition)?T:F;**

ex:

**int c=(a>b)?a:b;**

write a program 100 to 1:

public class Countdown {

public static void main(String[] args) {

for (int i = 100; i >= 1; i--) {

System.out.println(i);

}

}

}

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

int a=40;

double b= 20;

40,20 are literals and static values(there is no change)

Dynamic means take input from user assign values to n

Dynamic approach:

**Scanner class:**

* used to take the values from the user.

**Creation of object:**

**Scanner sc = new Scaner();**

* Scanner--->Class
* sc----> reference variable
* new--->keyword
* Scaner()----->constructor

it required input stream

Scanner sc = new Scanner (System.in);

intial value and final value:

o to 100 ------>if codition and for loop

100 to 0

using scanner class

OOPS: theritocal concepts

====

Encapsulation

inheritance

Abstraction

polymorphism

**Encapsulation**:

* **wrapping or binding the different kind of single entities**
* **object---->class contain -->variables and methods after binding with object**

**Abstraction:**

* **Hiding the unkonown functionalities showing required functionalities**.

ex: update balance for the bank account(we cant..i.e,some functionalities)

**Method Signature**:

**public abstract void makeSound();**

**Shape square= new Square();**

**Shape-🡪Abstart classname**

**Square🡪ref**

**New🡪keyword**

**Square-🡪instance**

* **public-**🡪**accessmodifier**
* **abstract-**🡪**keyword**
* **void**🡪**return type**
* **makeSound**🡪**methodname**
* **()-**🡪**method braces**

Example : // Abstract class

abstract class Animal {

// Abstract method (no implementation)

public abstract void makeSound();

// Regular method

public void eat() {

System.out.println("This animal is eating.");

}

}

// Subclass (inheriting from Animal)

class Dog extends Animal {

// Providing implementation for the abstract method

@Override

public void makeSound() {

System.out.println("Woof");

}

}

// Another subclass (inheriting from Animal)

class Cat extends Animal {

// Providing implementation for the abstract method

@Override

public void makeSound() {

System.out.println("Meow");

}

}

public class AbstractionDemo {

public static void main(String[] args) {

Animal dog = new Dog();

Animal cat = new Cat();

dog.makeSound(); // Outputs: Woof

dog.eat(); // Outputs: This animal is eating.

cat.makeSound(); // Outputs: Meow

cat.eat(); // Outputs: This animal is eating.

}

}

**Polymorphism:**

* the same name have different roles or activities...

**Two methods:**

* Method Overloading
* Method OverRiding

**method signature:**

**Method Declaration**:

**Accessmodifier returntype methodname(){**

**-------------------->method body**

**}**

ex:

public int setValue(int i){

}

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

public int add(){

}

public int add(int a){

}

public int add(int b, int c){

}

public int add(int a){ ------------>its not possible (duplicate)

}

1.order of parameters must be change

2.number of parameters must be change

3.by changing the return type

4.it can happen only with in the class.

**Method OverRiding:**

* **by changing the order (or) parameters**

ex:

* final keyword uses to stop the overriding.

class:

public Class Test(){

var

meth

}

**Concrete Method:**

* **it have signature and body**.

public int getName(){

------------------🡪body is there.....

}

* **only signature is there its abstract.**

ex: (public) abstract int getName();

**we are not creating objects for abstract classes.**

**INTERFACE:**

* **all are abstract classes.**
* **if you create class as interface cannot use abstract.**
* **no chance for creating objects**
* **loosely coupling is applicable for Interface**.

**Inheritance:**

* **acquiring all the properties from the parent class to child class**
* extends and implements are two keywords implement inheritance.
* always create object only for child. .not parent

class A{

}

Class B extends A{

}

**Strings**:

* **Combination of characters or sequece of characters enclosed in double("xyz") or single quotes('xyz').**
* as a literal we can create
* **String name= "madhusudhan";**
* object way

**String name = new String();**

* **Strings are immutable(once we have given the value we can't change)**
* **it is a user defined class.**

**String have two memories:**

**1. String pool**

**2. Heap memory (JVM)**

* **String name = new String("nithin");**
* **string literal are saving in String pool**
* **we cant store duplicate values in String pool.**
* **we can store duplicate values in heap.**

**Index:**

* **possesion your elementor position of individual**

**M a d h u**

**0 1 2 3 4**

**Length:**

**total elements for characters**

**Final Index:**

* length-1

**Hash code:**

The hash Code method in Java is used to generate an integer representation of an object, which is known as the object's hash code. This hash code is primarily used in hash-based collections such as HashMap, HashSet, and Hash table to efficiently locate and manage objects.

**Methods:**

TRIM:

It is used for removing spaces before word or after word.

Ex:” Madhusudhan “

Trim()=>Madhusudhan

* **String s= "java";**

**Indexof() and LastIndexOf()**

* **i want to find the duplicate values in a String.**

ex:

**String s ="Java";**

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

**char c =s.CharAt(i);**

implement logic;

}

**SUBSTRING:**

* **A part of String is called substring. In other words, substring is a subset of another String.**

**SPLIT():**

* **The split() method of String class can be used to extract a substring from a sentence. It accepts arguments in the form of a regular expression**.

**String Buffer and String Builder are using to update the string.**

**Synchronized:**

**String Buffer:**

**StringBuffer is Synchronized:**

* **At a time process only one person or (request) can access (or) we can say Thread safe.**
* **is not accurate**

**StringBuilder:**

* **String Builder is Non Synchronized**
* **late the process is StringBuilder.**
* **is accurate**

**Arrays:**

* **if you want to save more than one value with similar type.**
* **Code Optimization:** It makes the code optimized, we can retrieve or sort the data efficiently.
* **Random access:** We can get any data located at an index position.

int a=1;

save 1 to 10:

symbol:[]

**two ways creating:**

**int [] ar = {1,2.3};**

* length is 3
* empty elements store in an arry

**int[] ar = new int[size];**

Disadvantages

* **Size Limit:** We can store only the fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in Java which grows automatically
* **no predefined methods**
* **size of array is fixed length once u create**

Using For loop:

to insert values:

syntax:

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

arr[i]=value;

}

**Syntax:**

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

**arr[i]=value;**

**s.o.p(arr[]);**

**}**

**ArrayIndexOutOfBoundException:**

* **coming for index length is final one.**
* **The Java Virtual Machine (JVM) throws an ArrayIndexOutOfBoundsException if length of the array in negative, equal to the array size or greater than the array size while traversing the array.**

**Threads:**

* Before introducing the **thread concept**, we were unable to run more than one task in parallel. It was a drawback, and to remove that drawback, **Thread Concept** was introduced.
* A **Thread** is a very light-weighted process, or we can say the smallest part of the process that allows a program to operate more efficiently by running multiple tasks simultaneously.
* **When multiple threads are executed in parallel at the same time, this process is known as** [**Multithreading**](https://www.javatpoint.com/multithreading-in-java).

Just like a process, a thread exists in several states. These states are as follows:

New

|

Runnable

|

Running **------🡪 [run()] method exists**

|

Terminated

# **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.**
* **In this tutorial, we will learn about Java exceptions, it's types, and the difference between checked and unchecked exceptions**.

**Exception:**

* **Its Abnormal condition.**
* **Exception Handling is a mechanism to handle runtime errors such as :**
* **ClassNotFoundException,**
* **IOException,**
* **SQLException**
* **RemoteException, etc.**

**Throwable**

**| |**

**Exception Error**

**| StackOverFlowError**

**IO Exception VirtualMachineError**

**SQL Exception OutOfMemeoryError**

**ClassNotFoundException**

**Runtime Exception-🡪|**

**|**

**Arithmatic Exception**

**NullPointerException**

**NumberFormatException**

**IndexOutOfBoundException-🡪|**

**|**

**ArrayIndexOutOfBoundsException**

**StringIndexOutOfBoundsException**

**Java Exception Key Words:**

* **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.
* **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.

* **Finally:**

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

* **Throw:**

The "throw" keyword is used to throw an exception.

* **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.

**Ex:**

**public** **class** MyException {

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

**try** {

**int** d=100/0;

} **catch** (Exception e) {

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

}

System.***out***.println("its not possible......");

}

}

Output:

java.lang.ArithmeticException: / by zero

its not possible......

# **Collections in Java**

1. [Java Collection Framework](https://www.javatpoint.com/collections-in-java)
2. [Hierarchy of Collection Framework](https://www.javatpoint.com/collections-in-java#collectionhierarchy)
3. [Collection interface](https://www.javatpoint.com/collections-in-java#collectionmethods)
4. [Iterator interface](https://www.javatpoint.com/collections-in-java#collectioniterator)

The **Collection in Java** is a framework that provides an architecture to store and manipulate the group of objects.

**Iterable(Parent) here “I”-🡪 Interface**

| “C”🡪 class

**Collection(I)**

|

| | |

**List** (I) **Queue(I) Set(I)**

| | |

**ArrayList ©**  **PriorityQueue**

© **Hashset**  ©

| | |

**Linkedlist**

**©**  DeQueue (I) **Linked Hashset©**

| | |

**Vector**

© **Array Dequeue ©** **SortedSet (I)**

| |

**Stack**  **TreeSet©**

©

--🡪Dequeue to extends Queue

-🡪Sorted set extends set

## **Iterable Interface**

* **The Iterable interface is the root interface for all the collection classes**.
* It contains only one abstract method. i.e.,
* Iterator<T> iterator()

## **Collection Interface**

The Collection interface is the interface which is implemented by all the classes in the collection framework.

## **List Interface**

List interface is the child interface of Collection interface. It inhibits a list type data structure in which we can store the ordered collection of objects. It can have duplicate values.

1. List <data-type> list1= **new** ArrayList();
2. List <data-type> list2 = **new** LinkedList();
3. List <data-type> list3 = **new** Vector();
4. List <data-type> list4 = **new** Stack();

## **ArrayList**

The ArrayList class implements the List interface. It uses a dynamic array to store the duplicate element of different data types.

**Ex:**

1. **import** java.util.\*;
2. **class** TestJavaCollection1{
3. **public** **static** **void** main(String args[]){
4. ArrayList<String> list=**new** ArrayList<String>();//Creating arraylist
5. list.add("Ravi");//Adding object in arraylist
6. list.add("Vijay");
7. list.add("Ravi");
8. list.add("Ajay");
9. //Traversing list through Iterator
10. Iterator itr=list.iterator();
11. **while**(itr.hasNext()){
12. System.out.println(itr.next());
13. }
14. }
15. }

Output:

Ravi

Vijay

Ravi

Ajay

## **LinkedList**

LinkedList implements the Collection interface. It uses a doubly linked list internally to store the elements. It can store the duplicate elements. It maintains the insertion order and is not synchronized. In LinkedList, the manipulation is fast because no shifting is required.

## **Vector**

Vector uses a dynamic array to store the data elements. It is similar to ArrayList. However, It is synchronized and contains many methods that are not the part of Collection framework.

## **Stack**

The stack is the subclass of Vector. It implements the last-in-first-out data structure, i.e., Stack. The stack contains all of the methods of Vector class and also provides its methods like boolean push(), boolean peek(), boolean push(object o), which defines its properties.

## **Queue Interface**

Queue interface maintains the first-in-first-out order. It can be defined as an ordered list that is used to hold the elements which are about to be processed. There are various classes like PriorityQueue, Deque, and ArrayDeque which implements the Queue interface.

Queue interface can be instantiated as:

1. Queue<String> q1 = **new** PriorityQueue();
2. Queue<String> q2 = **new** ArrayDeque();

## **PriorityQueue**

The PriorityQueue class implements the Queue interface. It holds the elements or objects which are to be processed by their priorities. PriorityQueue doesn't allow null values to be stored in the queue.

Consider the following example.

## **Set Interface**

Set Interface in Java is present in java.util package. It extends the Collection interface. It represents the unordered set of elements which doesn't allow us to store the duplicate items. We can store at most one null value in Set. Set is implemented by HashSet, LinkedHashSet, and TreeSet.

## **HashSet**

HashSet class implements Set Interface. It represents the collection that uses a hash table for storage. Hashing is used to store the elements in the HashSet. It contains unique items.

## **LinkedHashSet**

LinkedHashSet class represents the LinkedList implementation of Set Interface. It extends the HashSet class and implements Set interface. Like HashSet, It also contains unique elements. It maintains the insertion order and permits null elements.

## **SortedSet Interface**

SortedSet is the alternate of Set interface that provides a total ordering on its elements. The elements of the SortedSet are arranged in the increasing (ascending) order. The SortedSet provides the additional methods that inhibit the natural ordering of the elements.

## **TreeSet**

Java TreeSet class implements the Set interface that uses a tree for storage. Like HashSet, TreeSet also contains unique elements. However, the access and retrieval time of TreeSet is quite fast. The elements in TreeSet stored in ascending order.

## **ArrayDeque**

ArrayDeque class implements the Deque interface. It facilitates us to use the Deque. Unlike queue, we can add or delete the elements from both the ends.

ArrayDeque is faster than ArrayList and Stack and has no capacity restrictions.