* String Class in Java- (String has no limits)
* It presents Jre in-build libraries (package-> java.lang)
* It is sequence of Characters/ Collection of characters, Objects of String is immutable which means it is constant & cannot be changed once created.

Two Ways to create String-

1. By String Literal-

Java String literal is created by using double quotes.

It directly refers to “ Constant Pool Memory also called the string literal pool “

Ex-

String S = “Java”;

2. By Using new Keyword-

It directly refers to “Heap memory”

When we create a String object using “new” keyword then Memory allocation is done in “Heap memory” and “constant pool memory”. (It always creates 2 objects)

->But when we create String object using new it directly refers to Heap memory but if we want to forcefully want it to refer to contant pool memory then we need to add intern() method .

Ex-

String S = new String(“Java”); 🡪new keyword ex.

String S1 = new String(“Java”).intern();🡪intern() method ex.

String In-Build Method-

| *String Method* | *Description* |
| --- | --- |
| *1.[int length()](https://www.geeksforgeeks.org/java-string-length-method-with-examples/)* | *Returns the number of characters in the String.* |
| *2. [Char charAt(int i)](https://www.geeksforgeeks.org/java-string-charat-method-example/)* | *Returns the character at ith index.* |
| *3. [String substring (int i)](https://www.geeksforgeeks.org/substring-in-java/)* | *Return the substring from the ith  index character to end.* |
| *4. [String substring (int i, int j)](https://www.geeksforgeeks.org/substring-in-java/)* | *Returns the substring from i to j-1 index.* |
| *5. [String concat( String str)](https://www.geeksforgeeks.org/java-string-concat-examples/)* | *Concatenates specified string to the end of this string.* |
| *6. [int indexOf (String s)](https://www.geeksforgeeks.org/java-string-indexof/)* | *Returns the index within the string of the first occurrence of the specified string. If String s is not present in input string then -1 is returned as the default value.* |
| *7. [int indexOf (String s, int i)](https://www.geeksforgeeks.org/java-string-indexof/)* | *Returns the index within the string of the first occurrence of the specified string, starting at the specified index.* |
| *8. [Int lastIndexOf( String s)](https://www.geeksforgeeks.org/java-lang-string-lastindexof-method/)* | *Returns the index within the string of the last occurrence of the specified string. If String s is not present in input string then -1 is returned as the default value.* |
| *9. [boolean equals( Object otherObj)](https://www.geeksforgeeks.org/boolean-equals-method-in-java-with-examples/)* | *Compares this string to the specified object.* |
| *10. [boolean  equalsIgnoreCase (String anotherString)](https://www.geeksforgeeks.org/equalsignorecase-in-java/)* | *Compares string to another string, ignoring case considerations.* |
| *[11. int compareTo( String anotherString)](https://www.geeksforgeeks.org/java-lang-string-compareto/)* | *Compares two string lexicographically.* |
| *12. [int compareToIgnoreCase( String anotherString)](https://www.geeksforgeeks.org/java-string-compareto-method-with-examples/)* | *Compares two string lexicographically, ignoring case considerations.*  *Note: In this case, it will not consider case of a letter (it will ignore whether it is uppercase or lowercase).* |
| *13. [String toLowerCase()](https://www.geeksforgeeks.org/java-string-tolowercase-examples/)* | *Converts all the characters in the String to lower case.* |
| *14. [String toUpperCase()](https://www.geeksforgeeks.org/java-touppercase-examples/)* | *Converts all the characters in the String to upper case.* |
| *15. [String trim()](https://www.geeksforgeeks.org/java-string-trim-method-example/)* | *Returns the copy of the String, by removing whitespaces at both ends. It does not affect whitespaces in the middle.* |
| *16.[String replace (char oldChar, char newChar)](https://www.geeksforgeeks.org/java-lang-string-replace-method-java/)* | *Returns new string by replacing all occurrences of oldChar with newChar.*  *Note: s1 is still feeksforfeeks and s2 is geeksgorgeeks* |
| *17. [boolean contains(CharSequence sequence)](https://www.geeksforgeeks.org/java-string-contains-method-example/" \t "_blank)* | *Returns true if string contains contains the given string.* |
| *18. [Char[] toCharArray():](https://www.geeksforgeeks.org/java-string-tochararray-example/)* | *Converts this String to a new character array.* |
| *19.[boolean startsWith(String prefix)](https://www.geeksforgeeks.org/string-startswith-method-in-java-with-examples/" \t "_blank)* | *Return true if string starts with this prefix.* |

|  |  |  |
| --- | --- | --- |
| *String Objects* | *StringBuffer Objects* | *StringBuilder Objects* |
| *Immutable* | *Mutable* | *Mutable* |
| *Asynchronized because it immutable.* | *It is Synchronized. This Means that multiple threads Cannot call the methods of String Buffer Simultaneously.* | *It is asynchronized. This means that multiple threads can call the methods of String Builder Simultaneously.* |
| *Thread-safe* | *Thread-safe* | *Not thread-safe* |
| *Objects can be created either through string literal or through new operator* | *Objects can be created only through new operator* | *Objects can be created only through new operator* |
| *Objects are stored in string constant pool as well as heap memory* | *Objects are stored in heap memory only.* | *Objects are stored in heap memory only.* |
| *Slower* | *Due to Synchronization it is lot Slower than String Builder.* | *Due to asynchronous there is no preliminary check for multiple threads so it is lot Faster than String Buffer.* |

StringBuilder & StringBuffer Functions:-

|  |  |  |
| --- | --- | --- |
| *[append(String str)](https://www.geeksforgeeks.org/stringbuilder-append-method-in-java-with-examples/" \t "_blank)* | *Appends the specified string to the end of the StringBuilder.* | *sb.append("Geeks");* |
| *insert(int offset, String)* | *Inserts the specified string at the given position in the StringBuilder.* | *sb.insert(5, " Geeks");* |
| *[replace(int start, int end, String)](https://www.geeksforgeeks.org/stringbuilder-replace-in-java-with-examples/" \t "_blank)* | *Replaces characters in a substring with the specified string.* | *sb.replace(6, 11, "Geeks");* |
| *[delete(int start, int end)](https://www.geeksforgeeks.org/stringbuilder-delete-in-java-with-examples/" \t "_blank)* | *Removes characters in the specified range.* | *sb.delete(5, 11);* |
| *[reverse()](https://www.geeksforgeeks.org/stringbuilder-reverse-in-java-with-examples/" \t "_blank)* | *Reverses the sequence of characters in the StringBuilder.* | *sb.reverse();* |
| *[capacity()](https://www.geeksforgeeks.org/stringbuilder-capacity-in-java-with-examples/" \t "_blank)* | *Returns the current capacity of the StringBuilder.* | *int cap = sb.capacity();* |
| *[length()](https://www.geeksforgeeks.org/stringbuilder-length-in-java-with-examples/" \t "_blank)* | *Returns the number of characters in the StringBuilder.* | *int len = sb.length();* |
| *[charAt(int index)](https://www.geeksforgeeks.org/stringbuilder-charat-in-java-with-examples/" \t "_blank)* | *Returns the character at the specified index.* | *char ch = sb.charAt(4);* |
| *[setCharAt(int index, char)](https://www.geeksforgeeks.org/stringbuilder-setcharat-in-java-with-examples/" \t "_blank)* | *Replaces the character at the specified position with a new character.* | *sb.setCharAt(0, 'G');* |
| *[substring(int start, int end)](https://www.geeksforgeeks.org/stringbuilder-substring-method-in-java-with-examples/" \t "_blank)* | *Returns a new String that contains characters from the specified range.* | *String sub = sb.substring(0, 5);* |
| *[ensureCapacity(int minimum)](https://www.geeksforgeeks.org/stringbuilder-ensurecapacity-in-java-with-examples/" \t "_blank)* | *Ensures the capacity of the StringBuilder is at least equal to the specified minimum.* | *sb.ensureCapacity(50);* |
| *[deleteCharAt(int index)](https://www.geeksforgeeks.org/stringbuilder-deletecharat-in-java-with-examples/" \t "_blank)* | *Removes the character at the specified position.* | *sb.deleteCharAt(3);* |
| *[indexOf(String str)](https://www.geeksforgeeks.org/java-string-indexof/" \t "_blank)* | *Returns the index of the first occurrence of the specified string.* | *int idx = sb.indexOf("Geeks");* |
| *[lastIndexOf(String str)](https://www.geeksforgeeks.org/stringbuilder-lastindexof-method-in-java-with-examples/" \t "_blank)* | *Returns the index of the last occurrence of the specified string.* | *int idx = sb.lastIndexOf("Geeks");* |
| *[toString()](https://www.geeksforgeeks.org/stringbuilder-tostring-method-in-java-with-examples/" \t "_blank)* | *Converts the StringBuilder object to a String.* | *String result = sb.toString();* |

*String buffer creates an empty constructor with an initial capacity. 🡪16 Characters*

String Compare-

Object 🡪 Object equals() method is override for String Compare

Object.equals() -Compares 2 Objects of Object Class.

Ex-

String S = “Java”;

String S1= new String(“Java”);

String S2 = “Java”;

String S3 = “java”;

String S4 = new String(“Java”).intern();

Three ways to compare- (Returns Boolean Values)

1.equals(); – (It Checks Contents for comparison)

String.equals() -Compares 2 Objects of String Class.

Ex-

S.equals(S2); 🡪true

S.equals(S3); 🡪false

S1.equals(S2); 🡪true

.equalsIgnoreTo(); - (It Checks Contents ignores Case is upper or lower)

S.equalsIgnoreTo(S3); 🡪true

2.== Operator – (It Checks references/ addresses)

Ex.

(S == S1) 🡪false

(S == S2) 🡪true

(S1 == S3) 🡪false

(S == S4) 🡪true

* Exception handling- ( predefined package 🡪 java.lang)

What is Exception in Java-

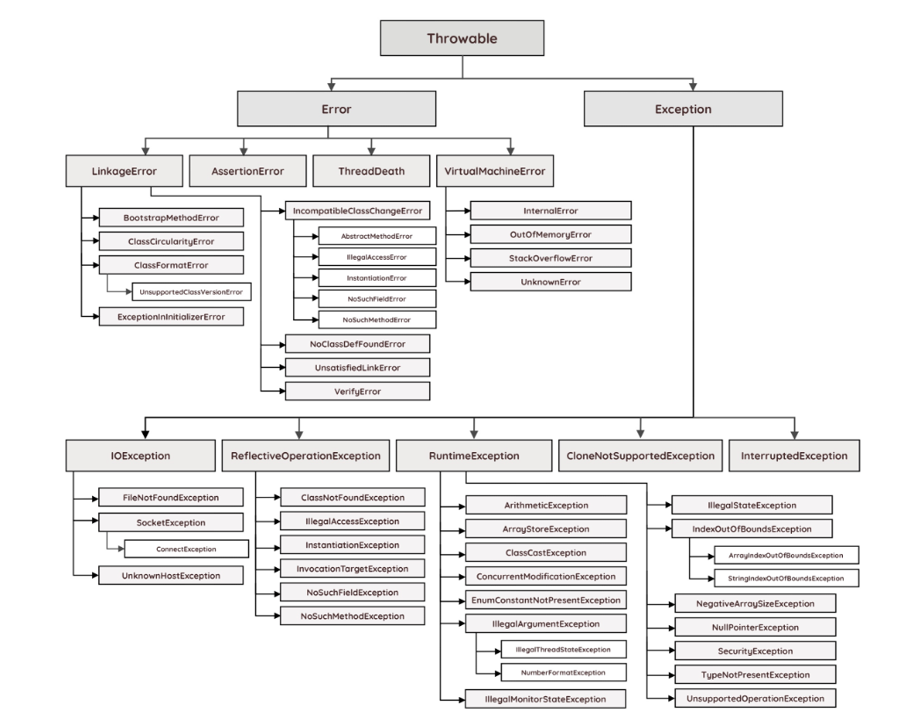
It is an event that disrupts the normal flow of program. If Exception occurs in program then below line of code doesn’t execute.

What is Exception handling in Java-

It is a powerful mechanism that handles runtime exceptions. So the normal flow of the program can be maintained.

Hierarchy of java Exception-

* which are having able in last of the word the 99% of them are interfaces.
* But throwable is a Class.



Ways to handle –

1.try catch block-

try{

This is the try block

In this block , keep those statement which may throw run time exception

}

Catch{

This is the catch block ,

It takes one argument of type java.lang.Exception

This block catches the exception thrown by try block

}

Nested Try catch block in Java:(Inner Try catch block)-

In java try catch blocks can be nested i.e one try block can contain another try-catch block.

Multiple catch block in java (till jdk 1.6)-

- In some cases, a single statement may throw more than one type of exception. In such cases, Java allows you to put more than one catch block. One catch block handles one type of exception.

- When an exception is thrown by the try block, all the catch blocks are examined in the order they appear and once catch block which matches with exception thrown will be executed.

- we can add multiple catch blocks only in sequence of parent-child relationship

|- catch(child/same level exception first ){ }

|- catch(child/same level/parent exception) { }

|- catch(exception) { } 🡪 (add parent exception last if we miss any exception in above code then it will detect on it)

Printing Exception message ways:

- *printStackTrace()* method : It prints the name of the exception, description and complete stack trace including the line where exception occurred .

- *getMessage()* method: Mostly used, It prints the description of the exception.

- *e.toString():* It prints name and description of the exception

Try catch using pipe (|) operator:

- From java 7 onwards, there is one way for handling multiple exceptions. Multiple exception thrown by the try block can be handled by a single catch block using pipe (|) operator.

- ex. ()

Catch() { } 🡪(parent child sequence)

2.finally block- (till jdk 1.6 after jdk 1.7 it was upgrades to resources (try-catch-resources))

try{

This is the try block

In this block , keep those statement which may throw run time exception

}

Catch{

This is the catch block ,

It takes one argument of type java.lang.Exception

This block catches the exception thrown by try block

}

finally{

this block is executes everytime.

to close inbuild classes like *Scanner* it was used.

it can be written directly after try block or after catch block.

}

Q. write the finally block but don’t execute that block?

-> using inbuild class - System.exit(0); (the program will be exited from that line)

3.throw

4.throws

Types of Java Exception-

1.Checked (Compile time)-

These exceptions are checked at compile time. If your code has a possibility of throwing a checked exception, you must either handle it using a try-catch block.

2.Unchecked (Runtime)-

These are exceptions that are not checked at compile time and can occur during runtime.

3.Error

* Serialization- (inbuild package🡪 java.io)
* Conversion of object into bytestream(byte code).
* Data security while transferring data between 2 applications (or in different machines).
* To persist(data needed for long time) that data in database.
* Deserialization- (java.io)
* Conversion of bytestream (txt file) in object.
* It is use to reconvert the file which is converted in serialization.

Throw:(User)- Java throw keyword is used to throw an exception explicitly.

We can throw either checked or unchecked exception

It’s mainly used to throw a custom exception

for example

throw new exception class (“error message”);

Throws: (developer)-

The java throws keyword is used to declare an exception. It gives an information to the programmer that there may occur an exception. So it is better for the programmer to provide the exception handling code so that the normal flow of the program can be maintained.

A table with text and words

Description automatically generated

* Collection-

Collection-

- It is a framework that provides an architect to store & manipulate the group of objects.

- In collection we can perform operation on data such as searching, sorting, insertion, manipulation & deletion.

- it represents single unit of object.

Framework-

- It provides readymade architecture.

- It represents a set of classes and interfaces.

- It is optional.

Collection Framework- (inbuild package 🡪 java.util) (jdk 1.2)

- it is a centralized and unified theme to store and manipulate the group of objects.

- it provides some pre-defined classes and interfaces to handle the group of objects.

- you can store the objects as a **list** or as a **set** or as a **queue** or as a **map** and perform operations like adding an object or removing an object or sorting the objects without much hard work.

- supports homogeneous data

Difference between Array & Collection framework-

Array

1. Array are not re-sizable. size of array is Fixed.

2. Can store both primitive and reference types.

3.supports homogeneous data(primitive single type) & heterogeneous data(by using object)

Collection framework

1. collection are dynamic & re-sizable

2. Can only store objects (Wrapper classes for primitives).

3. Uses Generics for type safety (ArrayList<String> ensures only String objects are stored).

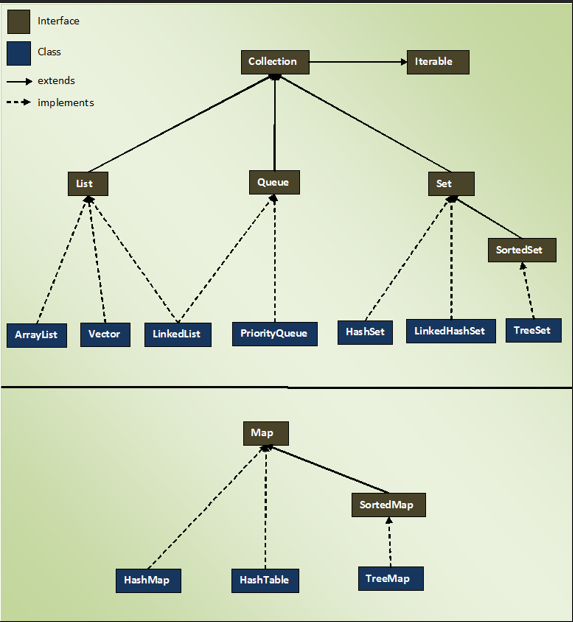
The entire collection is divided into the four interfaces-

1: List

2: Queue

3: Set

4: Map



1. List Interface- (package- java.util)

- List interface represent an ordered or sequential collection of objects.

- This interface has some methods which can be used to store and manipulate the ordered collection of objects.

- The classes which implemented the list interfaces are called as Lists.

- Arraylist, linkedlist , vector are some examples of list interface .

List Properties-

- follows insertion order.

- maintain indexing.

- allows duplicate element.

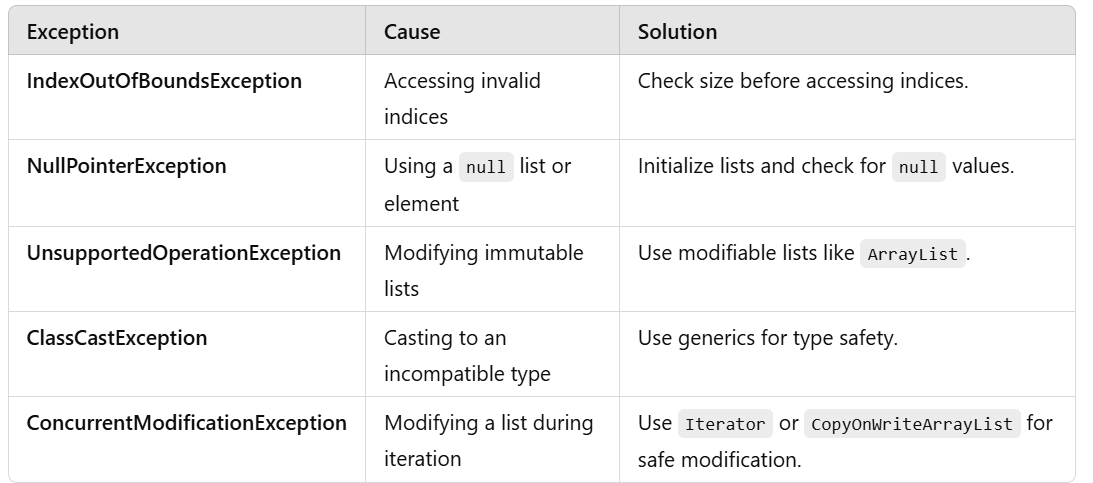
- random access to any element.

- multiple null element.

Generic-

<String> 🡪 generic can add primitive datatypes & only custom classes can be added .

* Collection framework is non generic before jdk 1.5 since 1.5 it is generic.
* Allows only 1 type of object.
* It is type-safe so type-casting is not required at runtime.



1. ArrayList Class- (package java.util)

Java ArrayList class uses a dynamic array for storing the elements. It is like an array, but there is no size limit. We can add or remove elements anytime. So, it is much more flexible than the traditional array.

When an ArrayList is created, its default capacity or size is 10 if not provided by the user. The size of the ArrayList grows based on load factor and current capacity.

The Load Factor is a measure to decide when to increase its capacity. The default value of load factor of an ArrayList is 0.75f

ArrayList in Java expands its capacity after each threshold which is calculated as the product of *current capacity* and *load factor* of the ArrayList instance.

Threshold = (Load Factor) \* (Current Capacity),

For example, if the user creates an ArrayList of size 10,

Threshold = Load Factor \* Current Capacity

= 0.75 \* 10

≅ 7

This means after adding the 7th element to the list, the size will increase as it has reached the threshold value. Internally,

a new ArrayList with a new capacity is created and the elements present in the old ArrayList are copied in the new ArrayList.

declaration,

List<E> al = new ArrayList<E>(); 🡪(loose coupling)

al.add(null);

ArrayList<E> als = new ArrayList<E>(); 🡪(tight coupling)

Als.add();

1. Java LinkedList class – (package- java.util)

- A Linked List in Java is a data structure used to store a sequence of elements, where each element (node) contains data and a reference (or pointer) to the next node in the sequence.

- Java LinkedList class uses a doubly linked list to store the elements.

- It provides a linked-list data structure. It inherits the Abstract List class and implements List and Deque interfaces.

The important points about Java LinkedList are:

o can contain duplicate elements.

o maintains insertion order.

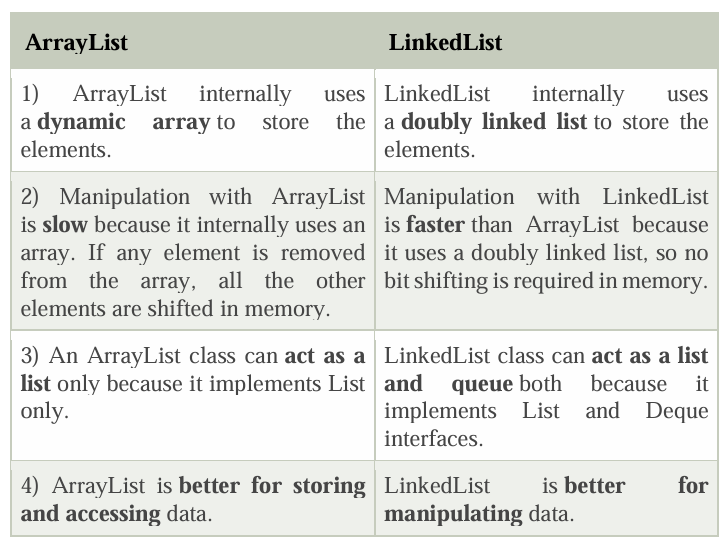
o non synchronized.

o is fast because no shifting needs to occur.

o class can be used as a list, stack or queue.

A screenshot of a computer

Description automatically generated



2. Set – (package- java.util)

- The set interface defines a set. The set is a liner collection of objects with no duplicates. Duplicate elements are not allowed in a set. The set interface extends collection interface. Set interface does not have its own methods. All its methods are inherited from collection interface. The only change that has been made to set interface is add () method will return false if you try to insert an element which is already present in the set.

The classes which implemented the set interfaces are called as hashset, Treeset ,Linkedhashset are some examples of Set interface.

Properties of Set:

* set contain only unique elements. It does not allow duplicate.
* Set can contain only one null element.
* Random access of elements is not possible.

1. HashSet- (package- java.util)

Java HashSet class is used to create a collection that uses a hash table for storage. It inherits the AbstractSet class and implements Set interface.

* Characteristics:
  + Allows null values (only one null is allowed).
  + HashSet stores the elements by using a mechanism called hashing.
  + HashSet contains unique elements only.
  + HashSet class is non synchronized.
  + HashSet doesn't maintain the insertion order. Here, elements are inserted on the basis of their hashcode.
  + HashSet is the best approach for search operations.
  + The initial default capacity of HashSet is 16, and the load factor is 0.75

A screenshot of a computer program

Description automatically generated

1. LinkedHashSet Class- (package- java.util)

Java LinkedHashSet class is a Hashtable and Linked list implementation of the Set interface. It inherits the HashSet class and implements the Set interface.

The important points about the Java LinkedHashSet class are:

o Java LinkedHashSet class contains unique elements only like HashSet.

o Java LinkedHashSet class provides all optional set operations and permits null elements.

o Java LinkedHashSet class is non-synchronized.

o Java LinkedHashSet class maintains insertion order.

A computer code with text

Description automatically generated

A screenshot of a computer

Description automatically generated

1. Tree Set- (package- java.util)

It implements the NavigableSet interface and extends the AbstractSet class. It is a part of the Java Collections Framework and is used to store elements in a sorted and unique order. Internally, it uses a TreeMap (a Red-Black tree) to manage the elements.

The important points about the Java TreeSet class are:

o Java TreeSet class contains unique elements only like HashSet.

o Java TreeSet class access and retrieval times are quiet fast.

o Java TreeSet class doesn't allow null element.

o Java TreeSet class is non synchronized.

o Java TreeSet class maintains ascending order.

(numbers🡪ascending, String🡪alphabatically)

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

* Iterator- (package – java.util)

is an object that allows you to traverse the elements of a collection (like a List, Set, or Map) one by one without exposing the underlying implementation. It implements the Iterator interface.

Key Features of an Iterator

1. Universal Traversal: Works on all collections that implement the Iterable interface.
2. Forward Traversal Only: Allows only forward movement through the collection.
3. No Index Access: Unlike loops with indices, iterators don't use indices to access elements.
4. Fail-Fast: Iterators throw a ConcurrentModificationException if the collection is modified during iteration.

A screenshot of a computer

Description automatically generated

*Syntax-*

*// Get an iterator*

*Iterator<String> iterator = list.iterator();*

*// Iterate through the collection*

*while (iterator.hasNext()) {*

*String fruit = iterator.next();*

*System.out.println(fruit);*

*}*

* ListIterator- (package -> java.util)

A ListIterator in Java is a specialized version of the Iterator interface that is used for traversing the elements of a List in both forward and backward directions. It provides more functionality compared to a standard Iterator.

Key Features of ListIterator

1. Bidirectional Traversal: Can traverse a list in both forward and backward directions.
2. Modify Elements: Allows updating or adding elements during iteration.
3. Index Access: Provides access to the current index of the iterator.
4. Available Only for Lists: Works only with collections that implement the List interface (e.g., ArrayList, LinkedList).

A screenshot of a computer

Description automatically generated

*Syntax-*

*// Get a ListIterator*

*ListIterator<String> iterator = list.listIterator();*

*// Forward traversal System.out.println("Forward Traversal:");*

*while (iterator.hasNext()) { System.out.println(iterator.next()); }*

* Fail-Fast –

ConcurrentModifyException occurs if we try to manipulate data (add,remove,update) at the time of iteration.

Key Features of Fail-Fast Iterators:

1. Immediate Error: They detect structural modifications during iteration and fail fast by throwing a ConcurrentModificationException.
2. Structural Modification: This refers to operations that add, remove, or rearrange elements in the collection (e.g., add(), remove()).
3. Iterator Scope: Fail-fast behavior applies to iterators and enhanced for-loops, as they use iterators internally.
4. Non-Synchronized: Fail-fast iterators are not thread-safe; concurrent modifications in multiple threads can result in exceptions or unpredictable behavior.



Collections with Fail-Fast Behavior-

Fail-fast iterators are typically used in non-thread-safe collections like:

* ArrayList
* LinkedList
* HashSet
* TreeSet
* HashMap
* TreeMap
* Fail-Safe in java-

fail-safe refers to a property of certain iterators that allow modifications to the collection during iteration without throwing a ConcurrentModificationException. These iterators operate on a separate copy of the collection (a snapshot), which ensures that structural changes do not affect the iteration process.

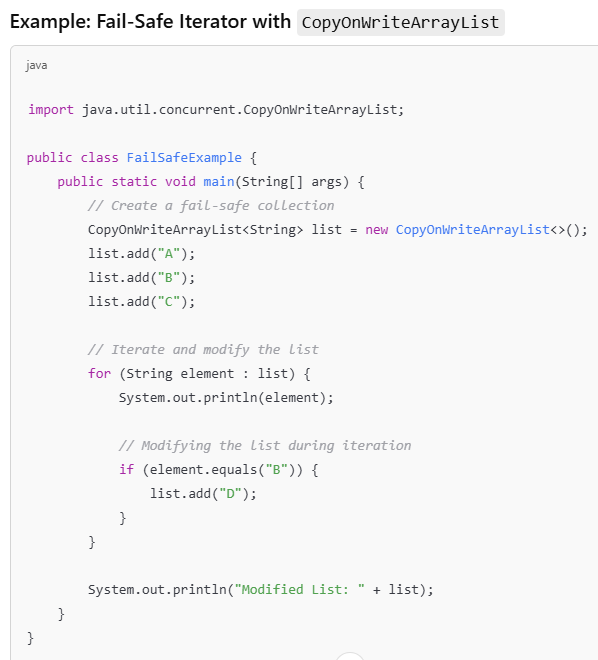
Key Features of Fail-Safe Iterators:

1. Snapshot-Based: Fail-safe iterators work on a clone or snapshot of the original collection, meaning they are unaffected by structural modifications.
2. Thread-Safe: These iterators are often used in concurrent environments where multiple threads might modify the collection simultaneously.
3. No ConcurrentModificationException: Modifications to the collection during iteration do not throw exceptions.

Examples of Fail-Safe Collections:

Fail-safe behavior is implemented in concurrent collections like:

* CopyOnWriteArrayList
* CopyOnWriteArraySet
* ConcurrentHashMap
* ConcurrentSkipListMap



A screenshot of a computer

Description automatically generated

* Comparable & Comparator in java-

A close-up of a text

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

3. Maps- (package-> java.util)

HashMap, LinkedHashMap, and TreeMap are classes in Java's java.util package that implement the Map interface. They store key-value pairs but differ in performance, ordering, and underlying data structures.

- A Map doesn't allow duplicate keys, but you can have duplicate values.

I. HashMap- (package 🡪java.util)

Java HashMap class implements the Map interface which allows us to store key and value pair, where keys should be unique. If you try to insert the duplicate key, it will replace the element of the corresponding key. It is easy to perform operations using the key index like updation, deletion, etc.

- It inherits the AbstractMap class and implements the Map interface.

Points to remember -

o It contains values based on the key.

o It contains only unique keys.

o It may have one null key and multiple null values.

o It is non synchronized.

o It maintains no order.

o The initial default capacity of Java HashMap class is 16 with a load factor of 0.75.

A computer screen shot of a code

AI-generated content may be incorrect.

II. LinkedHashMap- ()

Java LinkedHashMap class is Hashtable and Linked list implementation of the Map interface, with predictable iteration order. It inherits HashMap class and implements the Map interface.

Points to remember-

o Java LinkedHashMap contains values based on the key.

o Java LinkedHashMap contains unique elements.

o Java LinkedHashMap may have one null key and multiple null values. o Java LinkedHashMap is non synchronized.

o Java LinkedHashMap maintains insertion order.

o The initial default capacity of Java HashMap class is 16 with a load factor of 0.75.

A screenshot of a computer code

AI-generated content may be incorrect.

III. TreeMap- ()

Java TreeMap class is a red-black tree based implementation. It provides an efficient means of storing key-value pairs in sorted order.

The important points about Java TreeMap class are-

o It contains values based on the key. It implements the NavigableMap interface and extends AbstractMap class.

o It contains only unique elements.

o It cannot have a null key but can have multiple null values.

o It is non synchronized.

o It maintains ascending order.

A computer code with text

AI-generated content may be incorrect.

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

A screenshot of a computer

AI-generated content may be incorrect.

* Internal Working of HashMap-

Basics of HashMap

* Key-Value Pair Storage: Each entry in a HashMap is a key-value pair. Keys are unique, but values can be duplicated.
* Underlying Data Structure: HashMap uses a combination of an array and linked list (or a binary tree in some cases) to store entries.
* Hash Function: A hash function is used to calculate the index of the key-value pair in the internal array.

Buckets-

* the storing data in index that index is nothing but array bucket.
* In bucket data is stored in linkedlist.
* Since Java 8, when the size of a linked list at a bucket exceeds a threshold (8 by default), the list is converted into a binary search tree/ balance tree to improve search performance

Steps in HashMap Operations

1. Insertion (put() method)

* Step 1: Compute the hash code of the key.

Whenever put() method added it generates the hashcode

* Step 2: Calculate the index of the bucket using the hash function.

Formula - index = hash(key) % (16-1) 🡪(it will gave index to store that key & values)

* Step 3: Check the bucket at the calculated index:
  + If the bucket is empty: Create a new Node and store the key-value pair.
  + If the bucket is not empty:
    - Traverse the linked list (or tree) at the bucket.
    - Hash Collison occurs If the key already exists it checks by using .equals method that value is same or not if same then override the new value.
    - If the key does not exist, append the new Node to the list.

A diagram of a bucket with a linklist

AI-generated content may be incorrect.

* Here is some question-

SOP(map.put(101,”java”)); 🡪 null

SOP(map.put(101,”CPP”)); 🡪 java

SOP(map.put(102,”Python”)); 🡪 null

* Because put method return type is value
* At the time of putting values in sop if there is no content at that index then its previous value is null so it will print null.
* & if there is same key present & in sop we are putting new value with same key then it will print previous value.

1. How it knows the values are null or not

Set.add(11);

Set.add(12);

Checks with –

Returns Boolean type - method

map.put(11, PRESENT) == null

null = null 🡪(true)

or

11=null 🡪 (false)

* true 🡪 then that index is empty & it will insert that value on that index
* false 🡪 then that index is contains any value & it will override with new value
* JDBC in Java-

JDBC (Java Database Connectivity) is an API in Java that provides a way to connect and interact with relational databases. It allows Java applications to execute SQL queries, retrieve data, and update databases.

Key Components of JDBC

1. JDBC Drivers

* A JDBC driver is a software component that facilitates communication between Java applications and the database.
* Types of JDBC drivers:
  + Type-1: JDBC-ODBC Bridge Driver (deprecated).
  + Type-2: Native-API Driver (uses native library of the database).
  + Type-3: Network Protocol Driver (uses middleware server).
  + Type-4: Thin Driver (pure Java driver that communicates directly with the database).

Most modern databases, such as MySQL, PostgreSQL, and Oracle, use Type-4 drivers.

2. Key Interfaces

* DriverManager: Manages a list of database drivers and establishes a connection.
* Connection: Represents a connection to the database.

Statement: Used to execute static SQL queries.

* PreparedStatement: Used to execute precompiled SQL queries (safer against SQL injection).
* ResultSet: Represents the result set of a query.

Steps to Connect to a Database Using JDBC

1. Load the JDBC Driver

Class.forName("com.mysql.cj.jdbc.Driver");

1. Establish a Connection

Connection connection = DriverManager.getConnection( "jdbc:mysql://localhost:3306/mydatabase", "username", "password");

1. Create a Statement

Statement statement = connection.createStatement();

1. Execute SQL Queries

ResultSet resultSet = statement.executeQuery("SELECT \* FROM employees");

1. Process the Results

while (resultSet.next()) {

System.out.println(resultSet.getString("name"));

}

1. Close the Connection

resultSet.close();

statement.close();

connection.close();

JDBC query always throws exception that’s why we need to write jdbc code in try catch block or thows

A screenshot of a computer program

AI-generated content may be incorrect.

