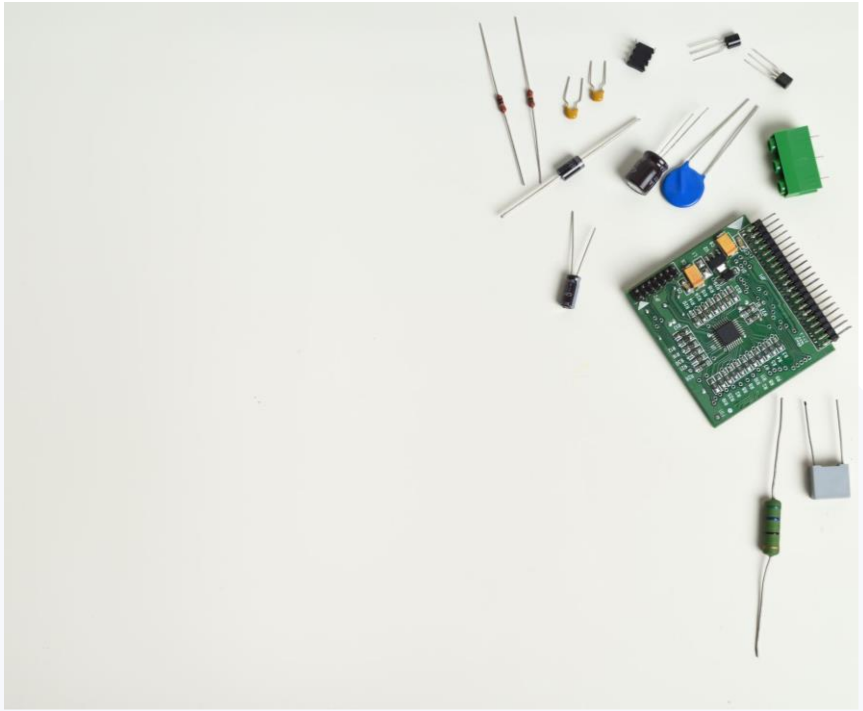
Array and Strings

Arrays

- An array is a data structure in Java that represents a collection of elements of the same data type, stored in contiguous memory locations and accessed using an index or position.



Types of Arrays

- Single

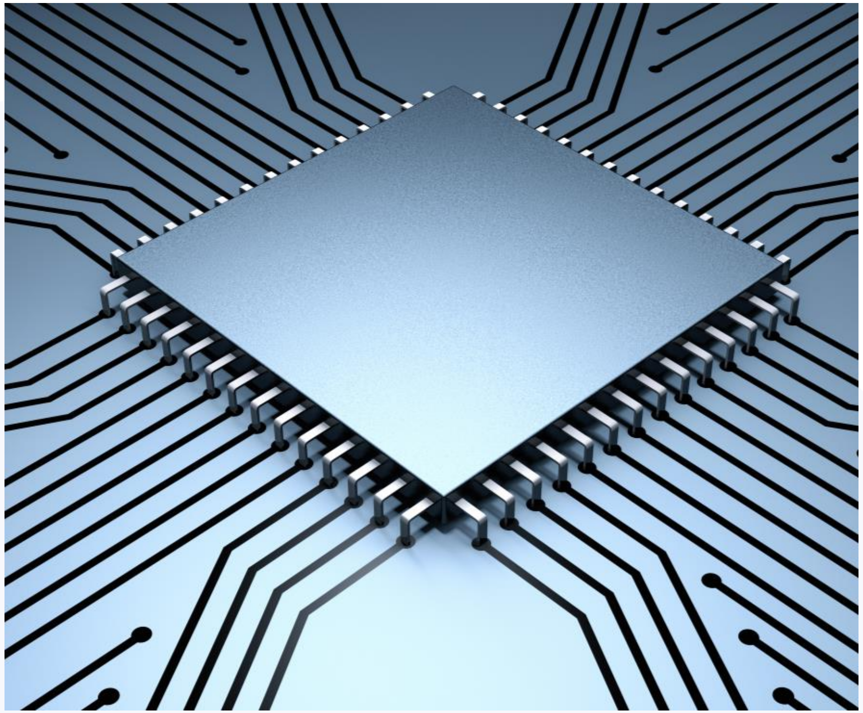
  Initialize an array by providing values within curly braces during declaration, such as int[] numbers = {1, 2, 3, 4, 5};. Access elements using their index, starting from 0, e.g., int third Element = numbers[2]; retrieves the third element (value 3) from the array.

- Multi-Dimensional Array

  A multi-dimensional array is an array of arrays, or a collection of elements organized in rows and columns. It can have two or more dimensions, allowing you to represent data in a grid-like structure, such as a matrix.

Collection

- Collection is an interface available in java.util package
- Collection is used to represent group of Object as a single entity
- Collections having so many advantage over arrays so that collection is using widely



Differences between array and Collection

Disadvantages of array

- ➢ Array hold homogenous data type
- • ex : int a[]=new int[5];
- ➢ Array size is fixed means after initializing an array you cannot change the size of an array
- ➢ Size of an array has to declare while initializing an array
- ➢ Performance will be low when operation is insertion and deletion

Advantages of array

- ➢ Collection hold heterogeneous element
- ex : Array List al=new Array List();
- ➢ Collection are growable in nature means size is not fixe
- ➢ No need to declare any size while initializing a Collection
- ex: LinkedList l=new LinkedList();
- ➢ The root class of Collection is inerrable interface

Some of the common method in Collection

1.add(Obj) : to add element

2.addAll(Collection c) : to add collection

3.remove(obj) : to remove object

4.removeAll(Collection c) : remove collection

5.retainAll(Collection c) : remove remaining element except these collection

6.size() : return size of the collection

7.clear() : remove all element from collection

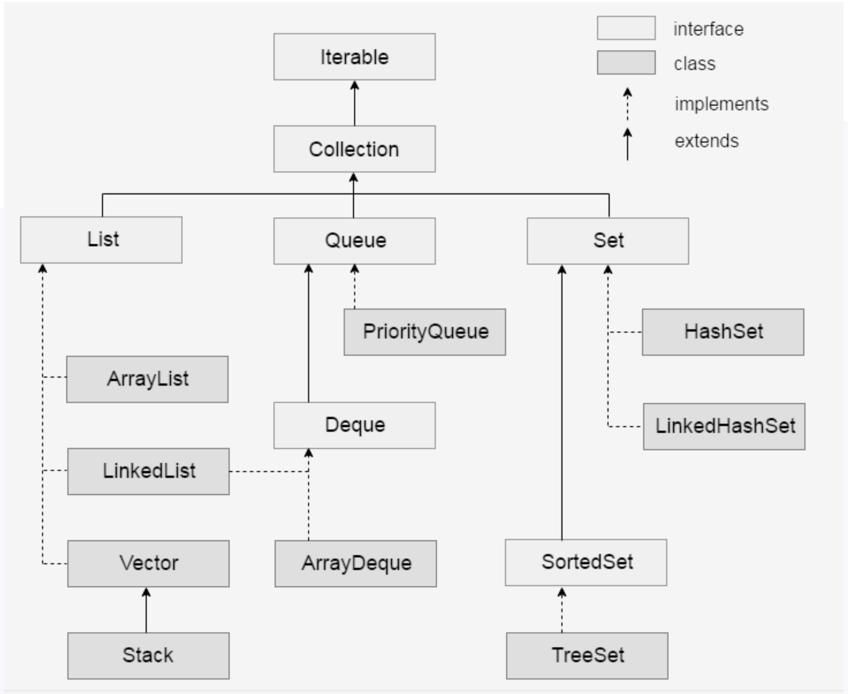
8.contains(Obj) : return true if object is present

9.containsAll(Collection c) : return true if collection is present

10.isEmpty() : return true if no element in collection

11.toArray[] : convert Collection to Object Array

Classification of Collection



Classification of Collection

- ❖ Iterable
- ❑ Collection

I. List

II. Queue

III. Set

i. List

- 1.ArrayList
- 2.LinkedList
- 3.vector
- stack

1.ArrayList

- It store element in indexing notation means in continuous memory location
- It allow duplicate element
- It store element in user added order
- Performance is good when operation is retrieval operation
- Performance is poor when operation is insert ,delete (more shifting of element)
- It is non-Synchronized

Syntax to create Array List and methods

- ArrayList al=new ArrayList();

Methods:

It implements Collection so Collection all method will be access

1.add(int a, Obj) : add element on specified index

2.remove(int a) : remove element from specified index

3.lastIndexOf(obj) : returns the last occurrence index of specified element

4.addAll(int index, Collection) : adding Collection into collection from specified index

5. **indexOf(Object o) :** return index of specified Object

➢ ArrayList implements random-access interface, so retrieval operation is fast

Difference between Array and ArrayList

Array

- it is dynamic (growable size)
- No need to specify size
- Need size() to get length of collection
- No need to specify type of elements or data type

ArrayList

- Array is static (fixed size)
- Need to specify size while creating array object
- Need length property to get size of array
- Need to specify type of element or data type

LinkedList

- It store the element in doubly linked list
- It allows duplicate element
- it is non-Synchronize
- The performance is good when Operation is insert and deletion
- The performance is poor when Operation is Retrieval
- It does not implement random-access interface

Storing of element

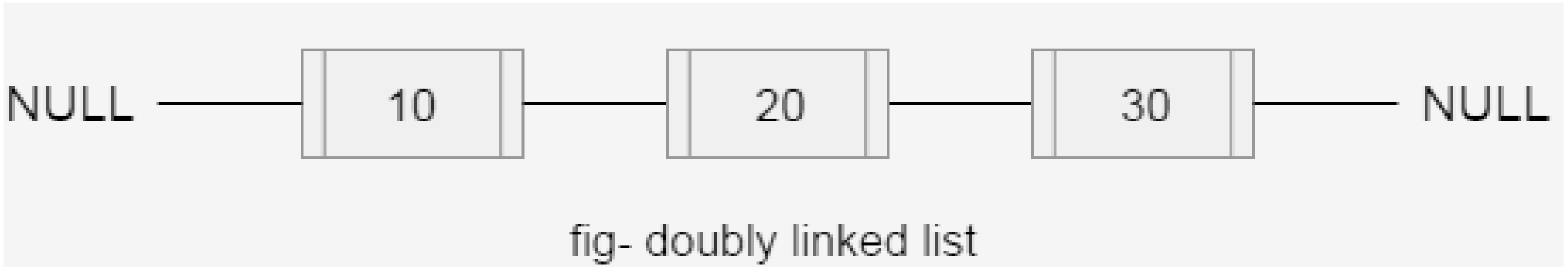


fig- doubly linked list

Syntax to create LinkedList

```
LinkedList al=new LinkedList();
```

Methods in LinkedList

1.add(int index, Obj) : add element in specified index

2.addFirst(Obj) : add element on beginning index

3.addLast(Obj : add Object on last

4. Object getFirst() : return first element

5. Object getLast() : return last element

6. int lastIndexOf(Object o) : return index of last element

Differences between ArrayList and LinkedList

ArrayList	LinkedList
• It stores elements in doubly linked list	Stores element in indexing notation
• Performance is poor when operation is retrieval	Performance is fast when operation is retrieval
• Performance is fast when operation is insert delete	Performance is poor when operation is insert delete
• Occupies more memory as links contain address of elements	Occupies less memory

Vector and stack

- Vector is a legacy class
- All methods are synchronized
- Stores element in index
- It allow duplicate element
- growable in nature
- null insertion is allowed
- Stack is not in use we other class which better then stack like HashSet

Differences between Vector and ArrayList

Vector	ArrayList
Methods are Synchronized	Methods are non-synchronized
You can use any cursor to access elements	You can't use Enumeration cursor to access elements
It is thread safe means multiple thread will not access this object	It is not threading safe means multiple thread can access this object

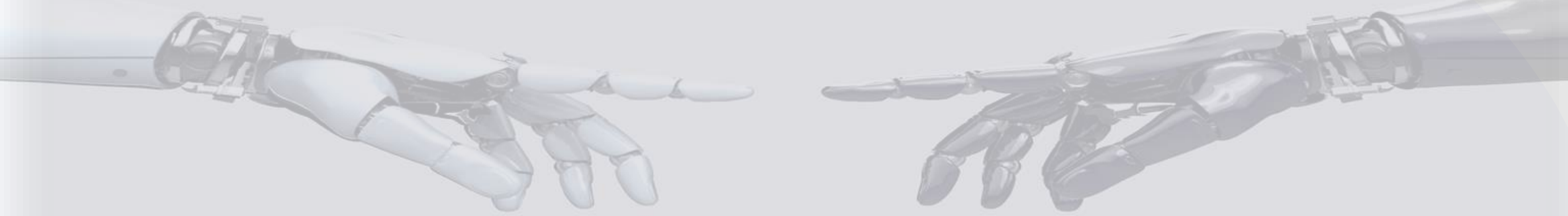
Enumeration interface

- ➢ It is interface available in java.util package
- ➢ It is a cursor to access the elements form legacy classes one by one

Methods

1.boolean hasMoreElements() :checks whether next element available or not

2.Object nextElement() : moves the pointer and returns elements from collection



Iterator interface

- ➢ This is a cursor used to access elements from collection one by one
- ➢ Invoke iterator() on collection to get instance of iterator

ex: iterator i=col.iterator()

Methods

1.hasNext() : checks whether next element available or not

2.next() : moves the pointer and returns elements from collection

3.remove() : remove the current element from the collection

Differences between Enumeration and Iterator

Enumeration	Iterator
It is legacy interface	It is collection framework interface
It is used to access elements from legacy classes(vector, stack, Hash Table)	It is universal cursor can be used to access elements from all class
You cannot remove elements using this	You can remove elements by Iterator

Strings

- String operations in Java include tasks like concatenation, substring extraction, length determination, comparison, and conversion, and these operations are performed using various built-in methods provided by the String class.

String, StringBuffer, StringBuilder

String:

- String class is present in java.lang package.
- String objects are immutable.
- When you try to modify the contents of object then new object is created.
- String class is final i.e. it cannot be extended.
- String class implements the following interfaces:
- java.io.Serializable
- java.lang.Comparable
- java.lang.CharSequence
- String object can be created in two ways:
- Without using new keyword.
- By using new keyword.

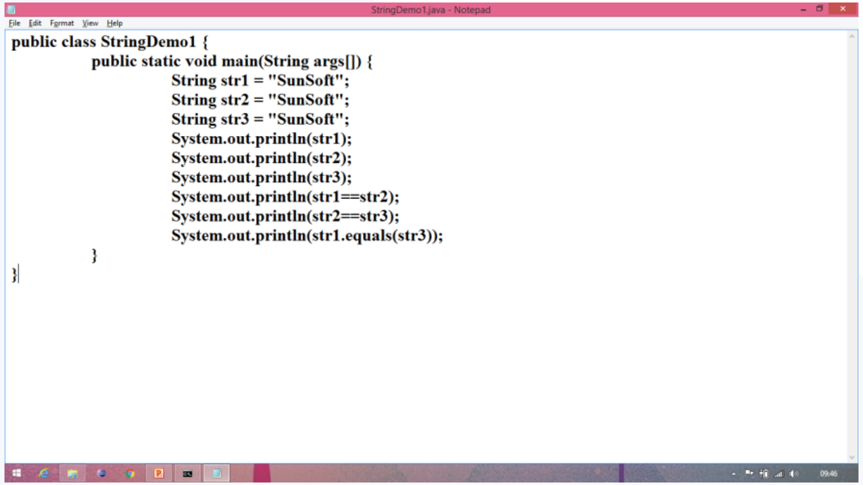
String, StringBuffer, StringBuilder (contd.)

Creating String object without new keyword:

- JVM allocates the memory for the String reference variable.
- JVM verifies the String literal in the String Constant Pool.
- If String literal is not available in the String Constant Pool, then it creates new String object inside the String Constant pool and the newly created String object address will be assigned to String reference variable.
- If String literal is available, then existing String object address will be assigned to String reference variable.

String, StringBuffer, StringBuilder (contd.)

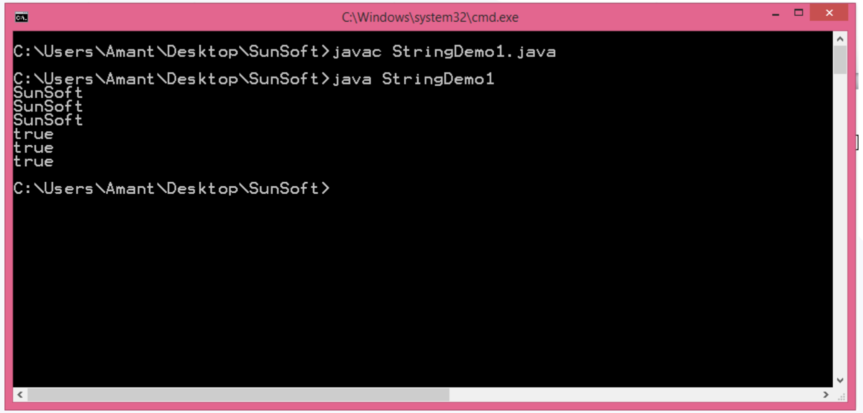
- Creating String object without new keyword Demo:



```
public class StringDemo1 {
        public static void main(String args[]) {
                String str1 = "SunSoft";
                String str2 = "SunSoft";
                String str3 = "SunSoft";
                System.out.println(str1);
                System.out.println(str2);
                System.out.println(str3);
                System.out.println(str1==str2);
                System.out.println(str2==str3);
                System.out.println(str1.equals(str3));
        }
}
```

String, StringBuffer, StringBuilder (contd.)

- **Creating String object without new keyword Demo:**



```
C:\Users\Amant\Desktop\SunSoft>javac StringDemo1.java

C:\Users\Amant\Desktop\SunSoft>java StringDemo1
SunSoft
SunSoft
SunSoft
true
true
true

C:\Users\Amant\Desktop\SunSoft>
```

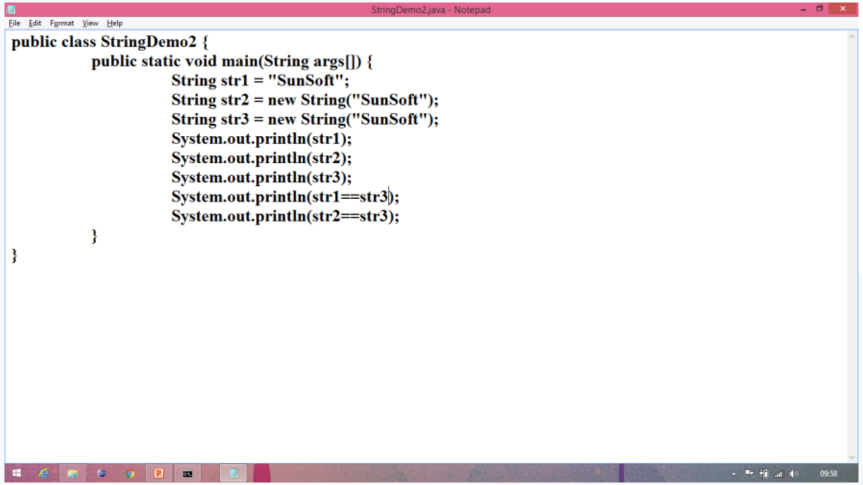
String, StringBuffer, StringBuilder (contd.)

Creating String object with new keyword:

- JVM allocates the memory for the String reference variable.
- JVM verifies the String literal in the String Constant Pool.
- If String literal is not available in the String Constant Pool then it creates new String object inside the String Constant pool.
- If String literal is available in the String Constant Pool then ignores that.
- It creates another new String object outside the constant pool and assigns address of the newly created String object outside the pool to String reference variable.

String, StringBuffer, StringBuilder (contd.)

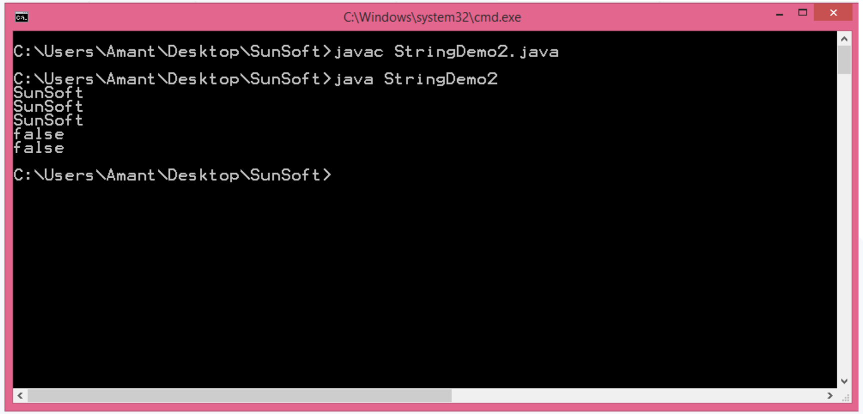
- Creating String object with new keyword Demo:



```
public class StringDemo2 {
		public static void main(String args[]) {
				String str1 = "SunSoft";
				String str2 = new String("SunSoft");
				String str3 = new String("SunSoft");
				System.out.println(str1);
				System.out.println(str2);
				System.out.println(str3);
				System.out.println(str1==str3);
				System.out.println(str2==str3);
		}
}
```

String, StringBuffer, StringBuilder (contd.)

- **Creating String object with new keyword Demo:**



```
C:\Users\Amant\Desktop\SunSoft>javac StringDemo2.java

C:\Users\Amant\Desktop\SunSoft>java StringDemo2
SunSoft
SunSoft
SunSoft
false
false

C:\Users\Amant\Desktop\SunSoft>
```

String, StringBuffer, StringBuilder (contd.)

Some useful methods in String class:

- public int length()
- public boolean isEmpty()
- public String concat(String)
- public String toLowerCase()
- public String toUpperCase()
- public String trim()

String, StringBuffer, StringBuilder

StringBuffer:

- StringBuffer is a final class in java.lang package.
- StringBuffer is mutable i.e. the content of StringBuffer can be modified after creation.
- Every StringBuffer object has a capacity associated with it.
- The capacity of StringBuffer is the number of characters it can hold.
- The capacity increases automatically as more contents added to it.
- The methods available in StringBuffer class are synchronized.
- In StringBuffer class multiple threads cannot access simultaneously.
- To sole this problem StringBuilder is added from Java 5.

String, StringBuffer, StringBuilder (contd.)

Some useful methods in StringBuffer class:

StringBuffer append(X value)

int capacity()

int length()

void setLength(int len)

String, StringBuffer, StringBuilder

- **StringBuilder:**
- StringBuilder is a final class in java.lang package.
- StringBuilder is added from Java 5.
- StringBuilder class functionality is same as StringBuffer only except that the methods available in StringBuilder class are non synchronized.
- In StringBuilder class multiple threads can execute simultaneously.

String, StringBuffer, StringBuilder (contd.)

Some useful methods in StringBuilder class:

StringBuffer append(X value)

int capacity()

int length()

void setLength(int len)

xebia.com