* Main method can’t be Overridden as Static methods can’t be overridden.
* Static methods can’t be overridden because “static method are bonded during compile time using Type of reference variable, and not Object.
* Final is used to apply restrictions on class, method and variable. Final class can't be inherited, final method can't be overridden and final variable value can't be changed.
* Blank Final Keyword can be initiated only in the Constructor.
* Reference Variable of a Parent class refers to the object of the child class is known as upcasting – A a = new B();
* Checked Exceptions – IOExceptions, SQLExceptions, ClassNotFoundExceptions.
* Unchecked Exceptions – ArthemeticExceptions, NullPointerExceptions etc….
* Final variable in Java cannot be changed. Once if we have assigned the final variable it cannot be changed it is fixed. but if you have declare a blank final variable then you can assign value to it only in constructor.+
* You can still access any non static variable inside any static method or block by creating an instance of class in Java and using that instance to reference instance variable. This is the only legitimate way to access non static variable on static context. here is a code example of accessing non static variable inside static context:
* public class StaticTest {
* private int count=0;
* public static void main(String args[]) throws IOException {
* StaticTest test = new StaticTest(); //accessing static variable by creating an instance of class
* test.count++;
* }
* }

Difference between throw and throws in Java

There are many differences between throw and throws keywords. A list of differences between throw and throws are given below:

No. throw throws

1) Java throw keyword is used to explicitly throw an exception. Java throws keyword is used to declare an exception.

2) Checked exception cannot be propagated using throw only. Checked exception can be propagated with throws.

3) Throw is followed by an instance. Throws is followed by class.

4) Throw is used within the method. Throws is used with the method signature.

5) You cannot throw multiple exceptions. You can declare multiple exceptions e.g.public void method()throws IOException, SQLException.

Difference between final, finally and finalize

There are many differences between final, finally and finalize. A list of differences between final, finally and finalize are given below:

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **final** | **finally** | **finalize** |
| 1) | Final is used to apply restrictions on class, method and variable. Final class can't be inherited, final method can't be overridden and final variable value can't be changed. | Finally is used to place important code, it will be executed whether exception is handled or not. | Finalize is used to perform clean up processing just before object is garbage collected. |
| 2) | Final is a keyword. | Finally is a block. | Finalize is a method |

## **Difference between constructor and method in Java**

There are many differences between constructors and methods. They are given below.

|  |  |
| --- | --- |
| **Java Constructor** | **Java Method** |
| A constructor is used to initialize the state of an object. | A method is used to expose the behavior of an object. |
| A constructor must not have a return type. | A method must have a return type. |
| The constructor is invoked implicitly. | The method is invoked explicitly. |
| The Java compiler provides a default constructor if you don't have any constructor in a class. | The method is not provided by the compiler in any case. |
| The constructor name must be same as the class name. | The method name may or may not be same as the class name. |

**Static Polymorphism:**

* Any entity which shows polymorphism during compile time is called static polymorphism. Operator Overloading, Constructor Overloading and method overloading are best examples of static polymorphism. Because, they show polymorphism during compilation.
* If there is any private, final or static method in a class, there is static binding.

**Dynamic Polymorphism:**

* Any entity which shows polymorphism during run time is called dynamic polymorphism. Method Overriding is the best example of dynamic polymorphism. It is also called dynamic binding or late binding, because type of the object used will be determined at run time only.
* A method is overridden, not the data members, so runtime polymorphism can't be achieved by data members.

**Method Overloading:**

* When a class has more than one method with same name, then we call that method is overloaded. The overloaded methods will have different number of arguments or different types of arguments, but name of the methods remains same.
* If two methods have same signature and different return types, then those methods will not be treated as two different methods or methods overloaded. For duplication, compiler checks only method signature not return types. If method signature is same, straight away it gives duplicate method error.
* Overloaded methods may have same access modifiers or different access modifiers. It also does not effect method overloading.
* compiler will check only method signature for method overloading or for duplicate methods. It does not check return types, access modifiers and static or non-static.
* Overloaded methods can be synchronized.
* we can overload main() method. A class can have any number of main() methods but execution starts from public static void main(String[] args) only.
* we can declare overloaded methods as final.

**Method Overriding:**

* **Name of the overrided method** must be same as in the super class. You can’t change name of the method in subclass.
* If super class method has primitive data type as its return type, then overrided method must have same return type in sub class also. If super class method has derived or user defined data type as its return type, then return type of sub class method must be of same type or its sub class.
* You can keep same visibility or increase the visibility of overrided method, but you can’t reduce the visibility of overrided methods in the subclass. For example, default method can be overided as default or protected or public method but not as private.
* We can change an exception from checked to unchecked but reverse is not possible.
* If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception.
* If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception but can declare unchecked exception.
* If the superclass method declares an exception, subclass overridden method can declare same, subclass exception or no exception but cannot declare parent exception.

Difference between method overloading and method overriding in java

There are many differences between method overloading and method overriding in java. A list of differences between method overloading and method overriding are given below:

|  |  |  |
| --- | --- | --- |
| **No.** | **Method Overloading** | **Method Overriding** |
| 1) | Method overloading is used *to increase the readability* of the program. | Method overriding is used *to provide the specific implementation* of the method that is already provided by its super class. |
| 2) | Method overloading is performed *within class*. | Method overriding occurs *in two classes* that have IS-A (inheritance) relationship. |
| 3) | In case of method overloading, *parameter must be different*. | In case of method overriding, *parameter must be same*. |
| 4) | Method overloading is the example of *compile time polymorphism*. | Method overriding is the example of *run time polymorphism*. |
| 5) | In java, method overloading can't be performed by changing return type of the method only. *Return type can be same or different* in method overloading. But you must have to change the parameter. | *Return type must be same or covariant* in method overriding. |

**Constructors**

* The main purpose of a constructor is to initialize the object variables.
* Must be with same name as class name because we are creating object by new <Classname>
* Constructor cannot have a return type.
* Why Constructor is not Static? --- We know that static methods, block or variables belong to the class. Whereas a Constructor belongs to the object and called when we use the new operator to create an instance. Since a constructor is not class property, it makes sense that it’s not allowed to be static.  
    
  Now, the main purpose of a constructor is to initialize the object variables. So if we make constructor as static then it won’t be able to initialize the object variables. That will defeat the whole purpose of having a constructor for creating the object. So it is justified to have the constructor as non-static.
* Constructors cannot be final because as per the Java Specifications, Constructors cannot be inherited to any sub class so it doesn’t make any sense to make the Constructors as Final.
* Duplicate constructors are not allowed in a class.
* Why constructors cannot be Inherited -- So if constructors were inherited in child class then child class would contain a parent class constructor which is against the constraint that constructor should have same name as class name.
* First statement in a constructor must be either super() or this()
  + Q) Does constructor return any value?
  + Yes, it is the current class instance (You cannot use return type yet it returns a value).
  + Q) What is the purpose of Constructor class?
  + Java provides a Constructor class which can be used to get the internal information of a constructor in the class. It is found in the java.lang.reflect package.

**1) Java static variable**

* If you declare any variable as static, it is known as a static variable.
* The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc.
* The static variable gets memory only once in the class area at the time of class loading.

**2) Java static method**

* If you apply static keyword with any method, it is known as static method.
* A static method belongs to the class rather than the object of a class.
* A static method can be invoked without the need for creating an instance of a class.
* A static method can access static data member and can change the value of it.

**Restrictions for the static method**

* There are two main restrictions for the static method. They are:
* The static method can not use non static data member or call non-static method directly.
* this and super cannot be used in static context.

Q) Why is the Java main method static?

Ans) It is because the object is not required to call a static method. If it were a non-static method, JVM creates an object first then call main() method that will lead the problem of extra memory allocation.

**Note: super() is added in each class constructor automatically by compiler if there is no super() or this().**

**this KeyWord:**

* this: to refer current class instance variable
* this: to invoke current class method
* this() : to invoke current class constructor

**super keyword:**

* super is used to refer immediate parent class instance variable.
* super can be used to invoke parent class method
* super() is used to invoke parent class constructor.

### **Understanding Java Access Modifiers**

Let's understand the access modifiers in Java by a simple table.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Modifier** | **within class** | **within package** | **outside package by subclass only** | **outside package** |
| **Private** | Y | N | N | N |
| **Default** | Y | Y | N | N |
| **Protected** | Y | Y | Y | N |
| **Public** | Y | Y | Y | Y |

* Encapsulation in Java is a process of wrapping code and data together into a single unit, for example, a capsule which is mixed of several medicines.

**Inheritance**

* Private Methods cannot be Inherited.
* Default member can be inherited within package.
* protected member can be inherited to any subclass
* static method, Constructor cannot be Inherited, But static methods and static data members can be called from the subclass.
* Static methods cannot be Override because, Because, when you override a method you are changing its property or not allowing its preference. As static method is shared by all objects of that class, Java do not allow to override it.
* super keyword is used to access super class members inside the sub class. Using super keyword, we can access super class methods, super class fields and super class constructors in the sub classes.
* this keyword is used to access other members of the same class. Using this keyword, you can access methods, fields and constructors of the same class within the class. this refers to current instance of the class.
* You can’t use super and this keywords in a static method and in a static initialization block even though you are referring static members.
* You should call super() and this() calling statements inside the constructors only and they must be first statement in the constructors.

**Abstraction:**

* **Abstraction** is a process of hiding the implementation details and showing only functionality to the user.
* We can’t create objects to those classes which are declared as abstract.
* It is not compulsory that abstract class must have abstract methods. It may or may not have abstract methods. But the class which has at least one abstract method must be declared as abstract.
* You can’t create objects to abstract class even though it does not contain any abstract methods.
* Any class extending an abstract class must implement all abstract methods. If it does not implement, it must be declared as abstract.
* Abstract methods cannot be private. Because, abstract methods must be implemented somehow in the sub classes. If you declare them as private, then you can’t use them outside the class.
* Constructors and fields cannot be declared as abstract.
* Abstract methods cannot be static.
* Abstract class contain the static methods.

**Interfaces**

* An interface in java is a blueprint of a class. It has static constants and abstract methods.
* The Java compiler adds public and abstract keywords before the interface method. Moreover, it adds public, static and final keywords before data members.
* we can have method body in interface. But we need to make it default method.
* While implementing any interface methods inside a class, that method must be declared as public. Because, according to method overriding rule, you can’t reduce visibility of super class method. By default, every member of an interface is public and while implementing you should not reduce this visibility.
* Static Initialization Block and IIB – Instance Initialization Block are not allowed in interfaces.
* Q) What is marker or tagged interface?
  + An interface which has no member is known as a marker or tagged interface, for example, Serializable, Cloneable, Remote, etc. They are used to provide some essential information to the JVM so that JVM may perform some useful operation.

Difference between abstract class and interface

Abstract class and interface both are used to achieve abstraction where we can declare the abstract methods. Abstract class and interface both can't be instantiated.

But there are many differences between abstract class and interface that are given below.

|  |  |
| --- | --- |
| **Abstract class** | **Interface** |
| 1) Abstract class can **have abstract and non-abstract** methods. | Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also. |
| 2) Abstract class **doesn't support multiple inheritance**. | Interface **supports multiple inheritance**. |
| 3) Abstract class **can have final, non-final, static and non-static variables**. | Interface has **only static and final variables**. |
| 4) Abstract class **can provide the implementation of interface**. | Interface **can't provide the implementation of abstract class**. |
| 5) The **abstract keyword** is used to declare abstract class. | The **interface keyword** is used to declare interface. |
| 6) An **abstract class** can extend another Java class and implement multiple Java interfaces. | An **interface** can extend another Java interface only. |
| 7) An **abstract class** can be extended using keyword "extends". | An **interface** can be implemented using keyword "implements". |
| 8) A Java **abstract class** can have class members like private, protected, etc. | Members of a Java interface are public by default. |
| 9)**Example:** public abstract class Shape{ public abstract void draw(); } | **Example:** public interface Drawable{ void draw(); } |

**Strings**

* We can create an String Object in two ways, String Literal and by using new Keyword.
* Creating an Object using String Literal, it stores the data in the String Constant Pool memory., with new Keyword it stores the data in the heap memory.
* Once string object is created its data or state can't be changed but a new string object is created. -- Immutable
* After a string literal, all the + will be treated as string concatenation operator.
* public String substring(int startIndex): This method returns new String object containing the substring of the given string from specified startIndex (inclusive).
* public String substring(int startIndex, int endIndex): This method returns new String object containing the substring of the given string from specified startIndex to endIndex. ---- endIndex is exclusive.
* Intern method Explanation - Whenever we create a String Object, two objects will be created i.e. One in the Heap Area and One in the String constant pool and the String object reference always points to heap area object. When line-1 execute, it will create two objects and pointing to the heap area created object. Now when line-2 executes, it will refer to the object which is in the SCP. Again when line-3 executes, it refers to the same object which is in the SCP area because the content is already available in the SCP area. No need to create a new one object.
* class GFG {
* public static void main(String[] args)
* {
* // S1 refers to Object in the Heap Area
* String s1 = new String("GFG"); // Line-1
* // S2 refers to Object in SCP Area
* String s2 = s1.intern(); // Line-2
* // Comparing memory locations
* // s1 is in Heap
* // s2 is in SCP
* System.out.println(s1 == s2);
* // Comparing only values
* System.out.println(s1.equals(s2));
* // S3 refers to Object in the SCP Area
* String s3 = "GFG"; // Line-3
* System.out.println(s2 == s3);
* }
* }
* The string valueOf() method coverts given type such as int, long, float, double, boolean, char and char array into string.
* A string that can be modified or changed is known as mutable string. StringBuffer and StringBuilder classes are used for creating mutable string.
* The capacity() method of StringBuffer class returns the current capacity of the buffer. The default capacity of the buffer is 16. If the number of character increases from its current capacity, it increases the capacity by (oldcapacity\*2)+2. For example if your current capacity is 16, it will be (16\*2)+2=34.
* Difference between String and String Buffer class.

|  |  |  |
| --- | --- | --- |
| **No.** | **String** | **StringBuffer** |
| 1) | String class is immutable. | StringBuffer class is mutable. |
| 2) | String is slow and consumes more memory when you concat too many strings because every time it creates new instance. | StringBuffer is fast and consumes less memory when you cancat strings. |
| 3) | String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method. | StringBuffer class doesn't override the equals() method of Object class. |

* Difference between StringBuffer and StringBuilder

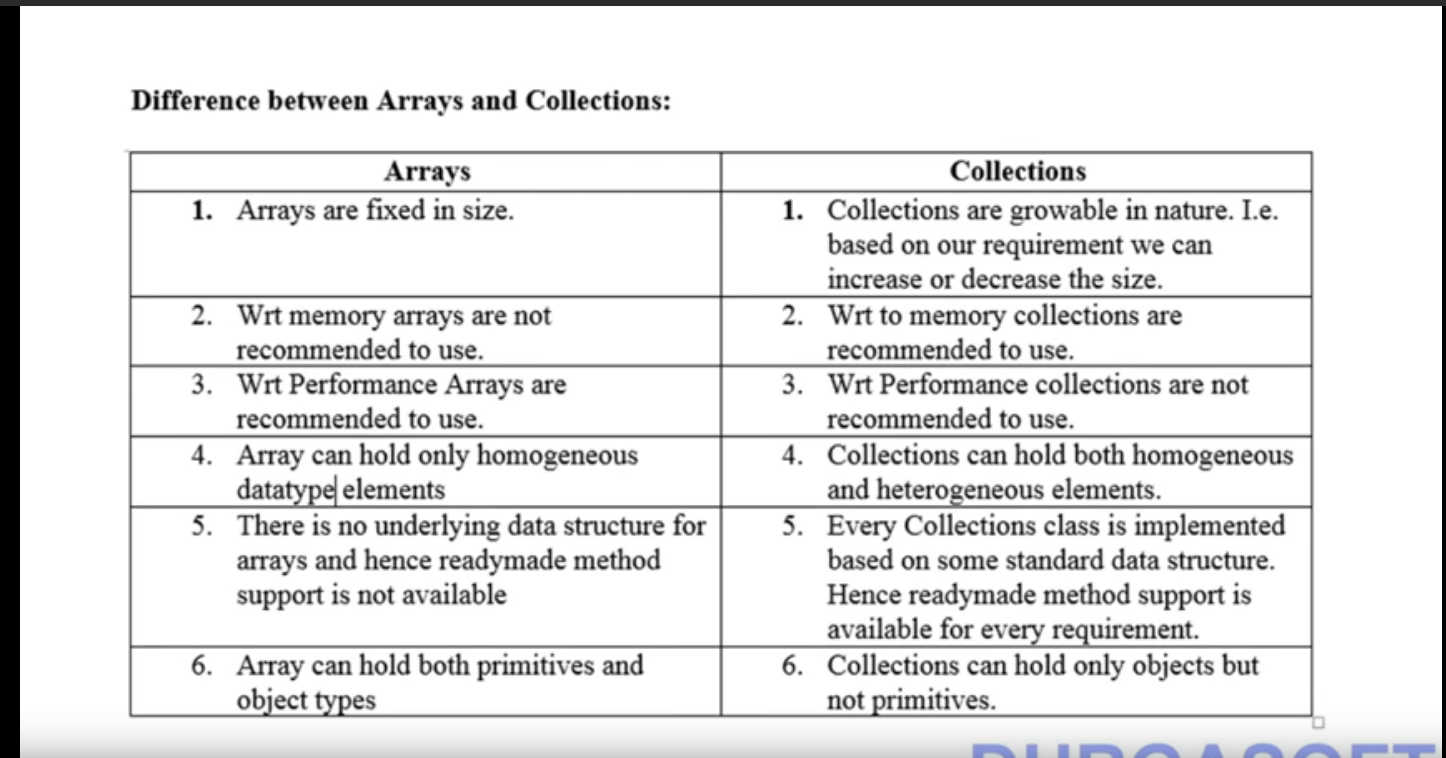
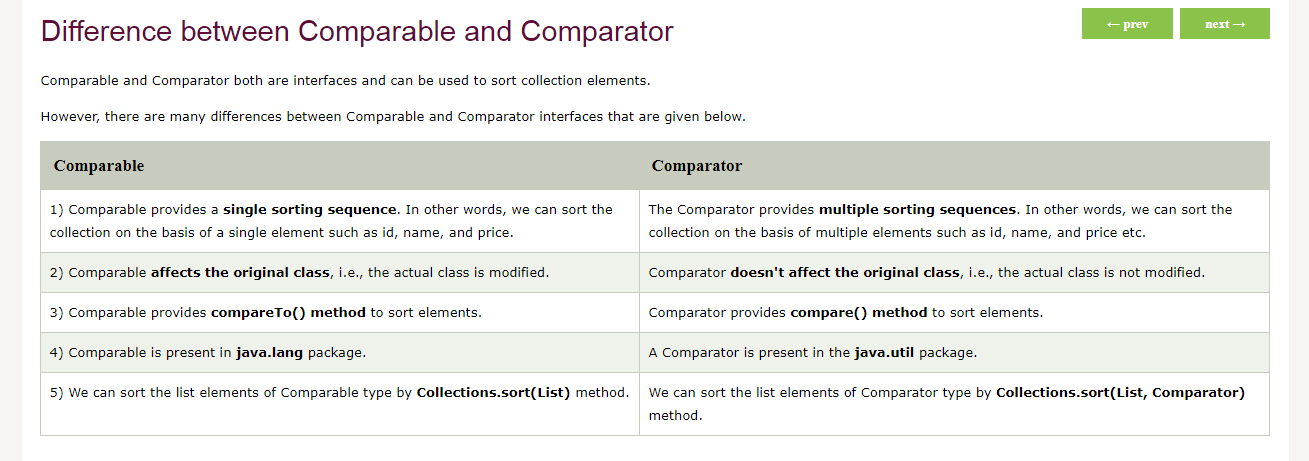
|  |  |  |
| --- | --- | --- |
| **No.** | **StringBuffer** | **StringBuilder** |
| 1) | StringBuffer is *synchronized* i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously. | StringBuilder is *non-synchronized* i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously. |
| 2) | StringBuffer is *less efficient* than StringBuilder. | StringBuilder is *more efficient* than StringBuffer. |

**Inner Class in Java**

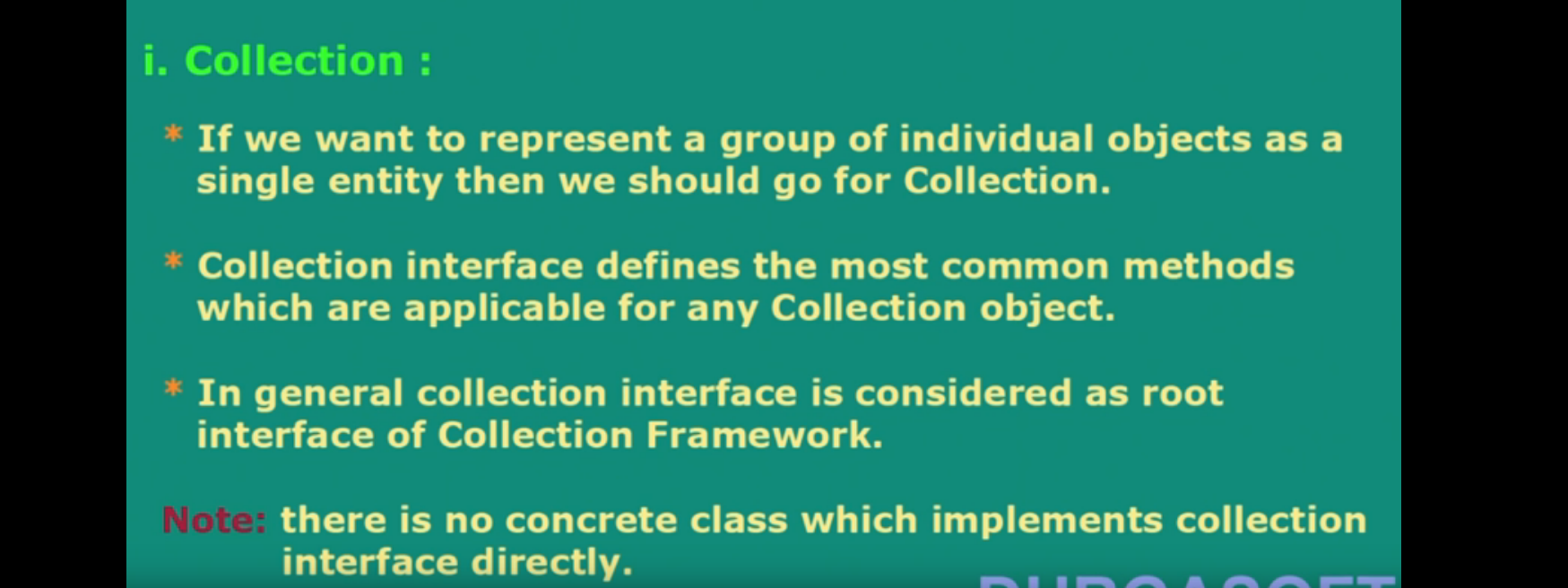
* Writing a class inside an another class is known as a Inner Class in Java.
* Three types of methods in Inner Class
  + Member Class
  + Static Class
  + Anonymous class
* Inner Class can be accessed as Below.
  + <ParentClass><InnerClassName> < InnerClassNameObj> = <ParentClassObj>.new< InnerClassName>;
* Static Class ---- Adding the static keyword to one of the Inner Class is known as a Static Class.
  + <ParentClass><InnerClassName> < InnerClassNameObj> = new <ParentClassObj>.< InnerClassName>;

**Collections**

**Differences between Arrays and Collections.**

* 
* Collection is representing a group of individual objects as a single entity.
* Collection Framework -- It defines several classes and Interfaces which can be used as a group of objects as a single entity.
* Difference between Collection and Collections:
  + **Collection** is an Interface while **Collections** is a Class.
  + **Collection** is used to represent group of individual objects as a Single entity, **Collections** is a utility class which is used to define utility methods (Searching, Sorting etc) for any collection Object.
* Difference between List and Set:
  + **List** allows duplicate values and **Set** doesn’t allow duplicate values.
  + **List** preserve Insertion Order, **Set** doesn’t preserve Insertion Order.
* Differences between Comparable and comparator   
  

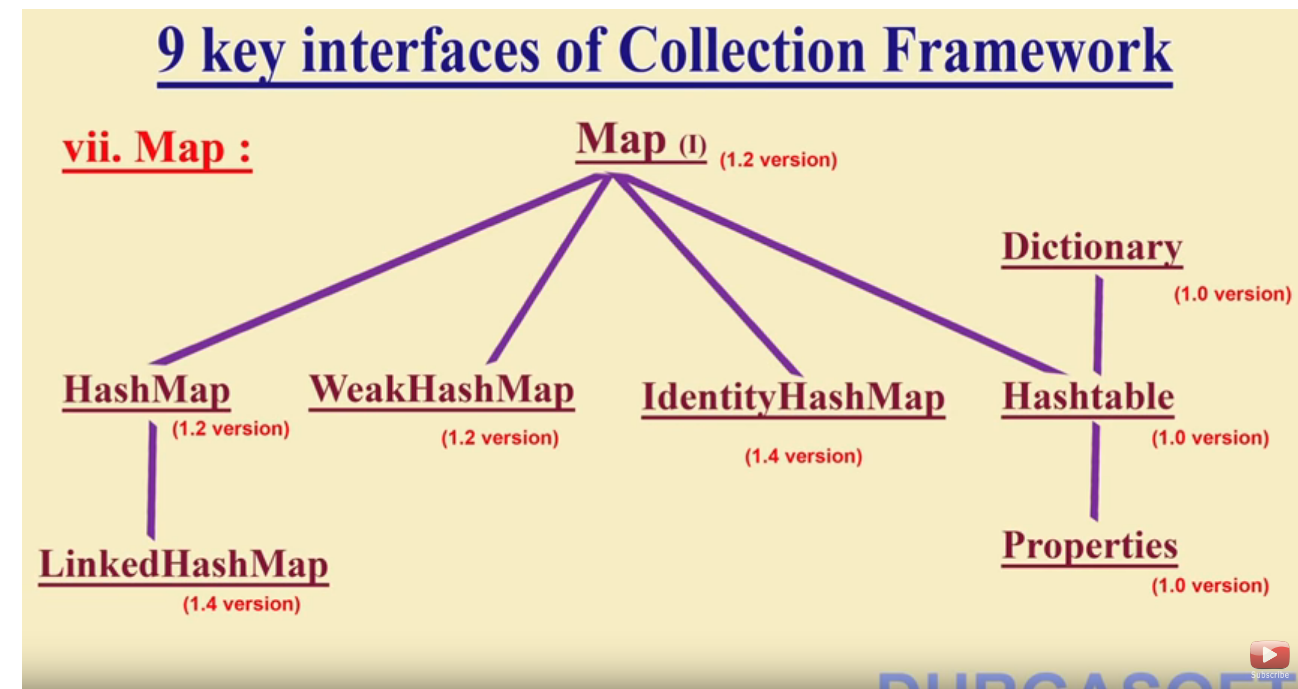
**9 Key Interfaces of Collection Framework:**

1. **Collection -   
   **
2. **List –**

* List is a child Interface of collection Interface, It allows duplicates and it follows the Insertion order.
* List contain ArrayList, LinkedList, Vector and Stack. (Stack implements Vector)

1. **Set –**

* List is a child Interface of collection Interface, It does not allows duplicates and it does not follows the Insertion order.
* Set contains HashSet and LinkedHashSet (LinkedHashSet extends HashSet)

1. **SortedSet –**
   * SortedSet is a child Interface of Set.
   * SortedSet is which duplicates are not allowed and insertion is based on some sorting order.
2. **NavigableSet –**
   * It is a child Interface of SortedSet and it defines several methods for the Navigable purposes.
   * It’s implementation class is TreeSet.
3. **Queue –**
   * It is a child Interface of Collection.
   * If we want to represent group of Individual Objects prior to processing them then we should go for Queue. [Eg – Storing of Email IDs before Sending them]
   * It follows First In First Out Insertion Order.
4. **Map –**
   * Map is not the child Interface of collection.
   * Map stores the data in the [Key, Value] pair, Key doesn’t store duplicate data while values can have the duplicate data.  
     
5. **SortedMap –**
   * It is a child Interface of Map.
   * We can store the data in by Sorting the Key value data.
6. **NavigableMap –**
   * It is a child Interface of SortedMap.
   * Its implementation class is TreeMap.

**ArrayList:**

1. Method to add elements to the list – add()
2. Method to add element at particular Index - public void add(int index, Object element)
3. Method to add all elements in an list - public boolean addAll(Collection c)
4. Method to add all elements in an list at particular Index - public boolean addAll(int index, Collection c)
5. Method to remove an element at particular Index - public Object remove(int index)
6. Method to get subList of an List - List subList(int fromIndex, int toIndex) --- fromIndex is included and toIndex is excluded.
7. Method to get last Occurrence index from an List - public int lastIndexOf(Object obj)
8. Method to get element from list – get()
9. Method to get particular index of an element present in an list - public int indexOf(Object o)
10. Method to verify whether element present in the list or not – contains();
11. Swap the elements in the list - public static void swap(List list, int i1, int i2)
12. Clone one ArrayList into second ArrayList - Eg --- ArrayList<String> al2 = (ArrayList<String>)al.clone();
13. Method to remove all the elements from the List – clear() or removeAll()
14. Method to verify whether elements present in list or not – isEmpty()
15. Method to replace the element in the list – set()
16. Method to convert Array to ArrayList – ArrayList<String> al = new ArrayList<String> (Arrays.toList(“Array Variable type.”))

**LinkedList:**

1. Method to add elements to the list – add()
2. Method to add element at particular Index - public void add(int index, Object element)
3. Method to add all elements in an list - public boolean addAll(Collection c)
4. Method to add all elements in an list at particular Index - public boolean addAll(int index, Collection c)
5. Method to add element at first position - void addFirst(Object item)
6. Method to add element at last position - void addLast(Object item)
7. Method to remove all the elements from the List – clear()
8. Clone one LinkedList into second LinkedList - Eg --- LinkedList<String> al2 = (LinkedList<String>)al.clone();
9. Method to verify whether element present in the list or not – contains();
10. Method to get element from list – get()
11. Method to get firstItem and LastItem from the List - Object get(int index) &&& Object getLast()
12. Method to get the index of item n List - int indexOf(Object item)
13. Method to get last occurrence of the specified element. - int lastIndexOf(Object item)
14. Iterator used to reverse the elements in the List -- Iterator it = list.descendingIterator();
15. Method to replace the element in the list – set()
16. Method to add element at first position of the list - public void push(E e)
17. Method to remove first element in the List and should return the removed element - public E pop()
18. Method to Retrieves and removes the head (first element) of this list. - LinkedList.poll()
19. Method that Retrieves and removes the first element of this list, or returns null if this list is empty. - public E pollFirst()
20. Method that Retrieves and removes the last element of this list, or returns null if this list is empty. - public E pollLast()
21. Method that Retrieves, but does not remove, the head (first element) of this list. -- public E peek()
22. Method that Retrieves, but does not remove, the first element of this list, or returns null if this list is empty. - public E peekFirst()
23. Method that Retrieves, but does not remove, the last element of this list, or returns null if this list is empty. - public E peekLast()

**Set: --------------- SAME APPLICABLE TO MAP ALSO**

1. HashSet doesn’t maintain any kind of order of its elements.
2. TreeSet sorts the elements in ascending order.
3. LinkedHashSet maintains the insertion order. Elements gets sorted in the same sequence in which they have been added to the Set.

**HashSet:**

1. HashSet doesn’t maintain any order, the elements would be returned in any random order.
2. HashSet doesn’t allow duplicates. If you try to add a duplicate element in HashSet, the old value would be overwritten.
3. HashSet allows null values however if you insert more than one nulls it would still return only one null value.
4. HashSet is non-synchronized.
   1. Method to clear the data – clear()

**Map:**

1. The main difference between them is that HashMap is an unordered collection while TreeMap is sorted in the ascending order of its keys. TreeMap is unsynchronized collection class which means it is not suitable for thread-safe operations until unless synchronized explicitly.
2. HashMap doesn’t maintain any order.
3. TreeMap sort the entries in ascending order of keys.
4. LinkedHashMap maintains the insertion order.

**Vector vs ArrayList in Java**

1. Vector is Synchronized
   1. ArrayList is non-Synchronized.
2. Vector Performance is slow
   1. ArrayList Performance is fast.
3. Vector Increases the size by 100%
   1. ArrayList Increases the size by 50%
4. Vector can use both Enumeration and Iterator.
   1. ArrayList can use only Iterator.

**Difference between ArrayList and LinkedList in Java**

1. ArrayList is the resizable array implementation of list interface
   1. while LinkedList is the Doubly-linked list implementation of the list interface.
2. Get or Search operations will be performed fast in ArrayList because ArrayList uses index based system for its elements as it internally uses array data structure
   1. Get or Search operations will be performed slow in LinkedList because LinkedList does not provide index based access for its elements as it iterates either from the beginning or end (whichever is closer) to retrieve the node at the specified element index.
3. Insert or add operations will be performed slow in ArrayList because In ArrayList all the elements need to be shifted to fill out the space created by removed element.
   1. Insert or add operations will be performed fast in LinkedList because LinkedList’s each element maintains two pointers (addresses) which points to the both neighbor elements in the list. Hence removal only requires change in the pointer location in the two neighbor nodes (elements) of the node which is going to be removed
4. ArrayList doesn’t contain descendingIterator.
   1. Linked Contain decendingIterator.
5. ArrayList Memory consumption is less as ArrayList maintains indexes and element data
   1. LinkedList memory consumption is more as LinkedList maintains element data and two pointers for neighbour nodes.

**Difference between ArrayList and HashMap in Java**

1. Arrayist is a List implementation Interface
   1. HashMap is an implementation of Map Interface.
2. ArrayList Memory consumption is less as it store values alone
   1. HashMap memory consumption is more as it store Key memory for every value.
3. ArrayList allows any number of null values.
   1. HashMap stores only single null value.
4. ArrayList display the elements in the Insertion order.
   1. HashMap doesn’t display the elements in the Insertion order.
5. ArrayList allows duplicate values.
   1. HashMap doesn’t allow duplicate values.
6. In ArrayList we can retrieve the elements based on the Index Number.
   1. In HashMap we can retrieve the elements based on the Key Value.

**Difference between List and Set in Java**

1. List allows duplicate values.
   1. Set doesn’t allow the duplicate values.
2. List allows any number of null values.
   1. Set allows only one null value.
3. List Implementations are ArrayList, LinkedList and Vector
   1. Set Implementations are HashSet, LinkedHashSet, TreeSet
4. By using listIterator we can travel in both forward and backward directions.
   1. By using set we can transverse in only forward directions.
5. List has one legacy class.
   1. Set doesn’t contain any legacy classes.

**HashSet vs TreeSet**

1. HashSet is faster
   1. TreeSet is slow.
2. HashSet stores the data in the random order
   1. TreeSet stores the data in the ascending order.
3. HashSet allows null value
   1. TreeSet doesn’t allow null values.
4. HashSet uses equals method for comparison
   1. TreeSet uses the compareTo method.

**Difference between HashMap and Hashtable in Java**

1. HashMap allows null values
   1. HashTable doesn’t allow null values.
2. HashMap is non-synchronized
   1. HashTable is synchronized
3. HashMap is faster
   1. HashTable is slower
4. Iterator in the HashMap is a fail-fast iterator
   1. while the enumerator for the Hashtable is not

**Difference between HashMap and HashSet**

1. HashMap implements Map Interface
   1. HashSet implements Set Interface
2. Duplicate Key Values are not allowed in the HashMap, Any Number of duplicate keys can be added.
   1. Duplicate values are not allowed in the HashSet
3. In HashMap no concept of dummy value
   1. HashSet internally uses HashMap to add elements. In HashSet, the argument passed in add(Object) method serves as key K. Java internally associates dummy value for each value passed in add(Object) method
4. HashMap is fast
   1. HashSet is slow.
5. HashMap use put() method for storing data
   1. HashSet use add() method for add or storing data

**Converting Array to ArrayList:**

String a[] = {"James","Jack","Jonathan","Jackson"};

ArrayList<String> al = **new** ArrayList<>(Arrays.*asList*(a));

**Fail Fast and Fail Safe Iterators in Java**

Concurrent Modification in programming means to modify an object concurrently when another task is already running over it.

ArrayList<String> ll = **new** ArrayList<>();

ll.add("James");

ll.add("James");

ll.add("James");

ll.add("James4");

ll.add("James5");

Iterator itr = ll.iterator();

**while**(itr.hasNext()){

System.***out***.println(itr.next());

ll.add("YK");

}

If you remove an element via Iterator remove() method, exception will not be thrown. However, in case of removing via a particular collection remove() method, ConcurrentModificationException will be thrown.

 ArrayList<Integer> al = new ArrayList<>();

        al.add(1);

        al.add(2);

        al.add(3);

        al.add(4);

        al.add(5);

        Iterator<Integer> itr = al.iterator();

        while (itr.hasNext()) {

            if (itr.next() == 2) {

                // will not throw Exception

                itr.remove();

            }

        }

        System.out.println(al);

        itr = al.iterator();

        while (itr.hasNext()) {

            if (itr.next() == 3) {

                // will throw Exception on

                // next call of next() method

                al.remove(3);

            }

        }

Non Fail fast iterators make a copy of the internal collection (object array) and iterates over the copied collection. Any structural modification done to the iterator affects the copied collection, not original collection. So, original collection remains structurally unchanged.

These iterators require extra memory for cloning of collection. Ex : ConcurrentHashMap, CopyOnWriteArrayList

**Difference between Iterator and ListIterator in java \*\*\*\*\*\*\***

1. Iterator is used for traversing List and Set both.

We can use ListIterator to traverse List only, we cannot traverse Set using ListIterator.

2) We can traverse in only forward direction using Iterator.

Using ListIterator, we can traverse a List in both the directions (forward and Backward).

3) We cannot obtain indexes while using Iterator

We can obtain indexes at any point of time while traversing a list using ListIterator. The methods nextIndex() and previousIndex() are used for this purpose.

4) We cannot add element to collection while traversing it using Iterator, it throws ConcurrentModificationException when you try to do it.

We can add element at any point of time while traversing a list using ListIterator.

5) We cannot replace the existing element value when using Iterator.

By using set(E e) method of ListIterator we can replace the last element returned by next() or previous() methods.

**What is the difference between Iterator and Enumeration?**

No. Iterator Enumeration

1) The Iterator can traverse legacy and non-legacy elements. Enumeration can traverse only legacy elements.

2) The Iterator is fail-fast. Enumeration is not fail-fast.

3) The Iterator is slower than Enumeration. Enumeration is faster than Iterator.

4) The Iterator can perform remove operation while traversing the collection. The Enumeration can perform only traverse operation on the collection.

**Wrapper Class:**

1. A Wrapper class in Java is the type of class which contains or the primitive data types
2. The package java.util contains classes which only handles objects, so it helps in this case too.
3. The automatic conversion of primitive data type into its corresponding wrapper class is known as autoboxing, for example, byte to Byte, char to Character, int to Integer, long to Long, float to Float, boolean to Boolean, double to Double, and short to Short.
4. The automatic conversion of wrapper type into its corresponding primitive type is known as unboxing. It is the reverse process of autoboxing. Since Java 5, we do not need to use the intValue() method of wrapper classes to convert the wrapper type into primitives.
5. Another useful method is the toString() method, which is used to convert wrapper objects to strings.

**Exception Handling:**

### **1) Checked Exception**

The classes which directly inherit Throwable class except RuntimeException and Error are known as checked exceptions e.g. IOException, SQLException etc. Checked exceptions are checked at compile-time.

### **2) Unchecked Exception**

The classes which inherit RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

### **3) Error**

Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

**GainSight:**

1. What is Hashing
2. Where does the String Constant Pool memory location saves – Heap Memory
4. Memory allocation of LinkedList and ArrayList
5. What is a Wrapper Class
6. Can Boolean store null values
7. Difference between Boolean and Boolean
   1. boolean is a java primitive type whereas Boolean is an object/reference type that wraps a Boolean
   2. boolean stores yes or no
   3. Boolean stores yes, no and NULL
8. How to Print a number without using loops
9. Regex in Java
10. How to remove duplicates in the String arrays. - String a[] = {"java","java","c++","c#", null, null};
11. Sorting techniques.
12. Code to launch Mozilla and IE
13. Exceptions in the Selenium
14. Whether those Exceptions Checked or Unchecked
15. Robot Class
16. How to send data to value without using the sendKeys