1) What do you understand by Collection Framework in Java?

The Java Collection framework provides an architecture to store and manage a group of objects. It permits the developers to access prepackaged data structures as well as algorithms to manipulate data. The collection framework includes the following:

- Interfaces
- Classes
- Algorithm

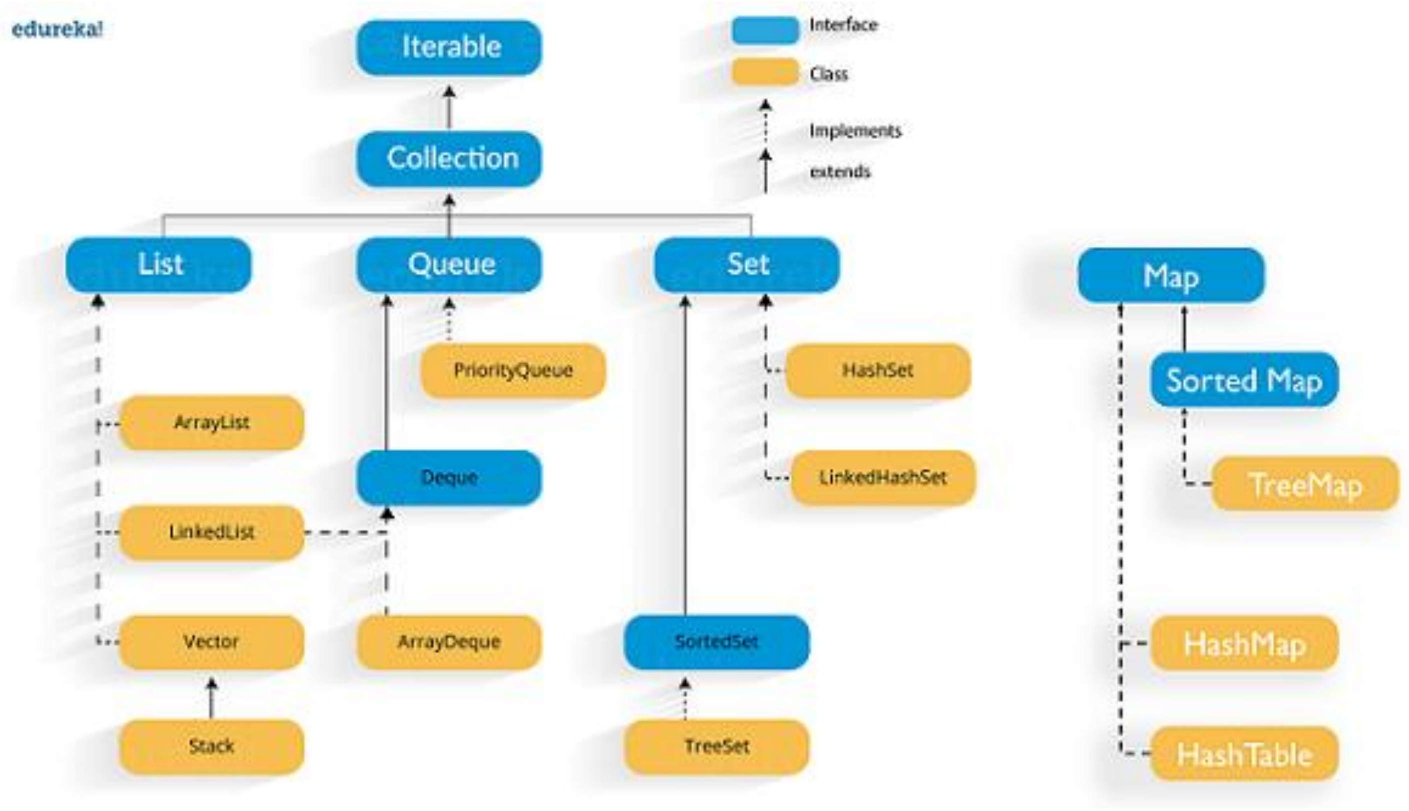
All these classes and interfaces support various operations such as Searching, Sorting, Insertion, Manipulation, and Deletion which makes the data manipulation really easy and quick.

2. What are the advantages of the Collection Framework in Java?

Below table contains the major advantages of the Java Collection Framework:

Feature	Description
Performance	The collection framework provides highly effective and icient data structures that result in enhancing the speed and curacy of a program.
Maintainability	The code developed with the collection framework is easy maintain as it supports data consistency and interoperability thin the implementation.
Reusability	The classes in Collection Framework can effortlessly mix th other types which results in increasing the code usability.
Extensibility	The Collection Framework in Java allows the developers to stomize the primitive collection types as per their quirements.

3. Describe the Collection hierarchy in Java.



4. List down the primary interfaces provided by Java Collections Framework?

Collection is the root of the collection hierarchy. A collection represents a group of objects known as its elements. The Java platform doesn't provide any direct implementations of this interface.

Set is a collection that cannot contain duplicate elements. This interface models the mathematical set abstraction and is used to represent sets, such as the deck of cards.

List is an ordered collection and can contain duplicate elements. You can access any element from its index. The list is more like an array with dynamic length.

A Map is an object that maps keys to values. A map cannot contain duplicate keys: Each key can map to at most one value.

Some other interfaces are Queue, Dequeue, Iterator, SortedSet, SortedMap and ListIterator.

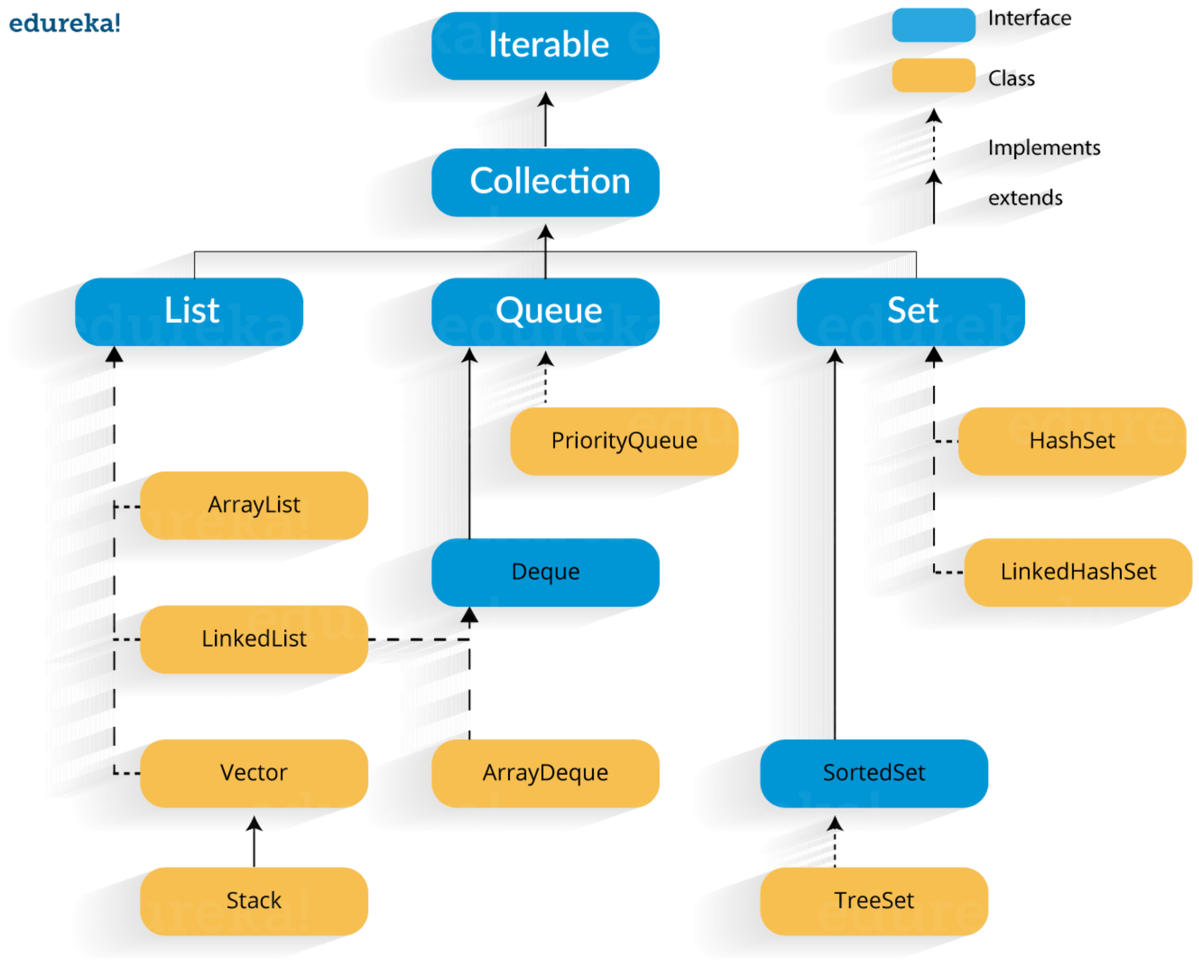
What is Generics in Java?

Generics is a term that denotes a set of language features related to the definition and use of Generic types and methods. Java Generic methods differ from regular data types and methods. Before Generics, we used the *collection* to store any type of objects i.e. *non-generic*. Now, Generics force the Java programmer to store a specific type of objects.

Now that you know what is Generics in Java, let's move further and understand why do you need Java Generics.

Why Java Generics?

If you look at the Java collection framework classes, then you will observe that most classes take parameter/argument of type Object. Basically, in this form, they can take any Java type as argument and return the same object or argument. They are essentially *heterogeneous* i.e. not of a similar type.



Sometimes in the Java application, the data type of the input is not fixed. The input can be an *integer*, a *float* or [*Java string*]. In order to assign the input to the [variable] of the right datatype, prior checks had to be conducted. In the traditional approach, after taking the input, the datatype of the input was checked and then was assigned to the variable of the right datatype. When this logic was used, the length of the code and execution time was increased. To avoid this, ***Generics were introduced***. When you use Generics, the parameters in the code is checked at compile time automatically and it sets the datatype by default. So this is where you need the concept of generics in Java.

Now that you have gained some insights on Generics, let's move ahead and look at the various ways how Generics can be applied to the source code.

Types of Java Generics

There are 4 different ways Generics can be applied to in Java and they are as follows:

1. Generic Type Class
2. Generic Interface
3. Generic Method
4. Generic Constructor

Now let’s understand how generics can be applied to type class in depth.

1. Generic Type Class

A class is said to be Generic if it declares one or more type variables. These variable types are known as the type parameters of the Java Class. Let’s understand this with the help of an example. In the below example, I will create a class with one property ***x*** and type of the property is an object.

```
class Genericclass{
private Object x;
public void set(Object x) { this.x = x; }
public Object get() { return x; }
}
```

Here, once you initialize the class with a certain type, the class should be used with that particular type only. E.g. If you want one instance of the class to hold the value ***x*** of type ‘**String**‘, then programmer should set and get the only String type. Since I have declared property type to Object, there is no way to enforce this restriction. A programmer can set any object and can expect any return value type from ***get method*** since all Java types are subtypes of Object class.

To enforce this type of restriction, we can use generics as below:

```
class Genericclass<X> {
//T stands for "Type"
private T x;
public void set(T x) { this.x = x; }
public T get() { return x; }
}
```

Now you can be assured that class will not be misused with wrong types. A simple example of “*Genericclass*” looks like as shown below:

```
Genericclass<String> instance = new Genericclass<String>();
instance.set("Edureka");
instance.set(10); //This will raise compile time error
```

So that’s how it works. This analogy is true for the interface as well. Let’s quickly look at an example to understand, how generics type information can be used in interfaces in java.

2. Generic Interface

An interface in Java refers to the abstract data types. They allow Java collections to be manipulated independently from the details of their representation. Also, they form a hierarchy in *object-oriented programming* languages. Let’s understand how generic type can be applied to interfaces in Java.

```
//Generic interface definition
interface GenericInterface<T1, T2>
{
T2 PerformExecution(T1 x);
T1 ReverseExecution(T2 x);
}

//A class implementing generic interface
```

```
class Genericclass implements GenericInterface<String, Integer>
{
public Integer PerformExecution(String x)
{
//execution code
}
public String ReverseExecution(Integer x)
{
//execution code
}
}
```

I hope you were able to understand how Generics can be applied to type class and interfaces. Now let's delve deeper into this article and understand how it is helpful for methods and constructors.

3. Generic Methods

Generic methods are much similar to generic classes. They differ from each other in only one aspect that the *scope or type information is inside the method only.* Generic methods introduce their own type parameters.

Let's understand this with an example. Below is an example of a generic method which can be used to find all the occurrences of type parameters in a list of variables.

```
public static <T> int countAllOccurrences(T[] list, T element) {
int count = 0;
if (element == null) {
for ( T listElement : list )
if (listElement == null)
count++;
}
else {
for ( T listElement : list )
if (element.equals(listElement))
count++;
}
return count;
}
```

If you pass a list of String to search in this method, it will work fine. But if you try to find a Number in the list of String, it will give compile time error.

This analogy is similar to the constructor as well. Let's take an example for the generic constructor and understand how it works.

4. Generic Constructor

A constructor is a block of code that initializes the newly created object. A **constructor** resembles an instance method in **Java** but it's not a method as it doesn't have a return type. **The constructor** has the same name as the class and looks like this in **Java** code. Now let's take an example and understand how it works.

```
class Dimension<T>
{
private T length;
private T width;
private T height;

//Generic constructor
public Dimension(T length, T width, T height)
{
super();
this.length = length;
this.width = width;
this.height = height;
}
}
```

In the above example, Dimension class's constructor has the type information. So you can have an instance of dimension with all attributes of a single type only. So this is how you can use generics type constructors. I hope you understood the types of generics in Java.

Now let's move further and look at the advantages of Generics in Java.

Advantages of Generics in Java

1. Code Reusability

You can compose a strategy or a class or an interface once and use for any type or any way that you need.

2. Individual Types Casting isn't required

Basically, you recovered information from ArrayList every time, you need to typecast it. Typecasting at each recovery task is a major migraine. To eradicate that approach, generics were introduced.

3. Implementing a non-generic algorithm

It can calculate the algorithms that work on various sorts of items that are type safe as well.

21. What is Set in Java Collections framework and list down its various implementations?

A Set refers to a collection that cannot contain duplicate elements. It is mainly used to model the mathematical set abstraction. The Java platform provides three general-purpose Set implementations which are:

1. HashSet
2. TreeSet
3. LinkedHashSet

6. List down the major advantages of the Generic Collection.

Below are the main advantages of using the generic collection in Java:

- Provides stronger type checks at the time of compilation
- Eliminates the need for typecasting
- Enables the implementation of generic algorithms which makes the code customizable, type-safe and easier to read

8. What do you understand by Iterator in the Java Collection Framework?

Iterator in Java is an interface of the Collection framework present in java.util package. It is a Cursor in Java which is used to iterate a collection of objects. Below are a few other major functionalities provided by the Iterator interface:

- Traverse a collection object elements one by one

- Known as Universal Java Cursor as it is applicable for all the classes of the Collection framework
- Supports READ and REMOVE Operations.
- Iterator method names are easy to implement

10. How the Collection objects are sorted in Java?

Sorting in Java Collections is implemented via Comparable and Comparator interfaces. When Collections.sort() method is used the elements get sorted based on the natural order that is specified in the compareTo() method. On the other hand when Collections.sort(Comparator) method is used it sorts the objects based on compare() method of the Comparator interface.

1. **What is difference between Iterator and ListIterator?**

    - We can use Iterator to traverse Set and List collections whereas ListIterator can be used with Lists only.
    - Iterator can traverse in forward direction only whereas ListIterator can be used to traverse in both the directions.
    - ListIterator inherits from Iterator interface and comes with extra functionalities like adding an element, replacing an element, getting index position for previous and next elements.