```java
// ComparatorExample1

import java.util.*;
//class student whose objects are to be sorted
class Student{
    int rollno;
    String name;
    int age;
    Student(int rollno,String name,int age){
        this.rollno=rollno;
        this.name=name;
        this.age=age;
    }
}
//AgeComparator class implementing Comparator to compare objects
class AgeComparator implements Comparator <Student> {
    public int compare(Student s1,Student s2){
        if(s1.age==s2.age)
            return 0;
        else if(s1.age>s2.age)
            return 1;
        else
            return -1;
    }
}



class ComparatorExample1{
    public static void main(String args[]){
        //create an ArrayList of Students
        ArrayList<Student> myList=new ArrayList<Student>();
        myList.add(new Student(101,"Ram",25));
        myList.add(new Student(106,"Hari",22));
        myList.add(new Student(105,"Gita",27));

        //call Collections.sort method with AgeComparator object to sort ArrayList
        Collections.sort(myList,new AgeComparator());
        System.out.println("Sorted Student objects by Age are as follows:");

        for(Student stud: myList) {
            System.out.println(stud.age + " --> " + stud.name);
        }
    }
}
```

Sorted Student objects by Age are as follows:

106 Hari 22

101 Ram 25

105 Gita 27

```java
package Unit10;


import java.util.*;

class Student1{

    String name;
    float percentage;

    Student1(String name, float percentage){
        this.name = name;
        this.percentage = percentage;
    }

}


//PercentageComparator class implementing Comparator to compare objects
class PercentageComparator implements Comparator<Student1>{
    public int compare(Student1 obj1, Student1 obj2){
        if(obj1.percentage<obj2.percentage){
            return 1;
        }
        return -1;
    }
}


public class ComparatorExample2{

    public static void main(String args[]) {

        ArrayList<Student1> studList = new ArrayList<Student1>();

        studList.add(new Student1("Gouthami", 90.61f));
        studList.add(new Student1("Raja", 83.55f));
        studList.add(new Student1("Honey", 85.55f));
        studList.add(new Student1("Teja", 77.56f));
        studList.add(new Student1("Varshith", 80.89f));

        Comparator<Student1> com = new PercentageComparator();

        Collections.sort(studList, com);

        System.out.println("Avg % --> Name");
        System.out.println("---------------------");
        for(Student1 stud:studList) {
            System.out.println(stud.percentage + " --> " + stud.name);
        }
    }
}
```

Avg % --> Name

90.61 --> Gouthami

85.55 --> Honey

83.55 --> Raja

80.89 --> Varshith

77.56 --> Teja

Collections in Java

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

Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.

Java Collection means a single unit of objects. Java Collection framework provides many interfaces Set, List, Queue, Deque and classes ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet.

What is Collection in Java ?

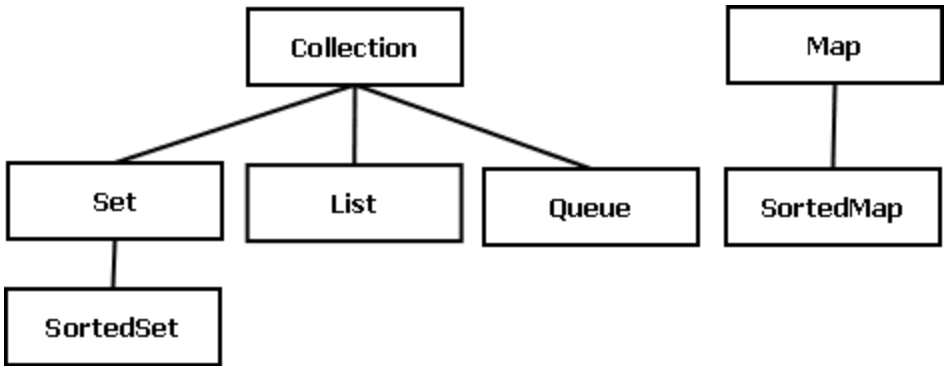
A Collection represents a single unit of objects, i.e., a group.

What is a framework in Java ?

Framework represents a set of classes and interfaces. It provides ready made architecture.

What Is a Collections Framework ?

A collections framework is a unified architecture for representing and manipulating collections. All collections frameworks contain the following:



- **Interfaces :** These are abstract data types that represent collections. Interfaces allow collections to be manipulated independently of the details of their representation. In object-oriented languages, interfaces generally form a hierarchy.
- **Implementations :** These are the concrete implementations of the collection interfaces. In essence, they are reusable data structures.

- **Algorithms :** These are the methods that perform useful computations, such as searching and sorting, on objects that implement collection interfaces. The algorithms are said to be polymorphic: that is, the same method can be used on many different implementations of the appropriate collection interface. In essence, algorithms are reusable functionality.

How Many Types of Collections in Java ?

There are three generic types of collections in Java.

1. Ordered Lists
2. Dictionaries or Maps
3. Sets

What is the use of Java collections ?

The Java collections framework gives the programmer access to prepackaged data structures as well as to algorithms for manipulating them. A collection is an object that can hold references to other objects. The collection interfaces declare the operations that can be performed on each type of collection.

What is the difference between Java collection and Java collections ?

Collections is an utility class present in java.util package to define several utility method like sorting, searching for collection object. Collections is a class which has some static methods and that method returns the collection. Collection is an interface, rather than root interface in collection hierarchy.

What are collection APIs give me an example ?

The Collection API is a set of classes and interfaces that support operation on collections of objects.

- **Example of Classes** : HashSet, HashMap, ArrayList, LinkedList, TreeSet and TreeMap.
- **Example of Interfaces** : Collection, Set, List and Map.

What is the hierarchy of collections framework in Java ?

The hierarchy of the entire collection framework consists of four core interface such as Collection, List, Set, Map, and two specialized interfaces named SortedSet and SortedMap for sorting. All the interfaces and classes for the collection framework are located in java.util package.

Why do we need Java collections ?

Collections are used almost in every programming language and when Java arrived, it also came with Collection classes. Collections are used in situations where data is dynamic. Collections allow adding an element, deleting an element and host of other operations. There are a number of Collections in Java allowing to choose the right Collection for the right context. You can play with data structure and algorithms.

Benefits of Java Collections Framework

The Java Collections Framework provides the following benefits :

1. **Reduces programming effort :** By providing useful data structures and algorithms, the Collections Framework frees you to concentrate on the important parts of your program rather than on the low-level "plumbing" required to make it work. By facilitating interoperability among unrelated APIs, the Java Collections Framework frees you from writing adapter objects or conversion code to connect APIs.
2. **Increases program speed and quality :** The Collections Framework provides high-performance, high-quality implementations of useful data structures and algorithms. The various implementations of each interface are interchangeable, so programs can be easily tuned by switching collection implementations. Because you're freed from the drudgery of writing your own data structures, you'll have more time to devote to improving programs' quality and performance.
3. **Allows interoperability among unrelated APIs :** The collection interfaces are the vernacular by which APIs pass collections back and forth. If my network administration API furnishes a collection of node names and if your GUI toolkit expects a collection of column headings, APIs will interoperate seamlessly, even though they were written independently.
4. **Reduces effort to learn and to use new APIs :** Many APIs naturally take collections on input and furnish them as output. In the past, each such API had a small sub-API devoted

to manipulating its collections. There was little consistency among these ad hoc collections sub-APIs, so you had to learn each one from scratch, and it was easy to make mistakes when using them. With the advent of standard collection interfaces, the problem went away.

5. **Reduces effort to design new APIs :** This is the flip side of the previous advantage. Designers and implementers don't have to reinvent the wheel each time they create an API that relies on collections; instead, they can use standard collection interfaces.
6. **Fosters software reuse :** New data structures that conform to the standard collection interfaces are by nature reusable. The same goes for new algorithms that operate on objects that implement these interfaces.

Which collections are thread safe in Java ?

ArrayList, LinkedList, HashSet, LinkedHashset and TreeSet in Collection Interface and HashMap, LinkedHashMap and Treemap are all non-synchronized. All collection classes (except Vector and Hashtable) in the java.util package are not thread-safe. The only two legacy collections are thread-safe: Vector and Hashtable.

What is a set in Java collections ?

A Set is a Collection that cannot contain duplicate elements. It models the mathematical set abstraction. The Set interface contains only methods inherited from Collection and adds the restriction that duplicate elements are prohibited. Set also adds a stronger contract on the behavior of the equals and hashCode operations, allowing Set instances to be compared meaningfully even if their implementation types differ. Two Set instances are equal if they contain the same elements.

What is the advantage of generic collection ?

Code that uses generics has many benefits over non-generic code: Stronger type checks at compile time. A Java compiler applies strong type checking to generic code and issues errors if the code violates type safety. Fixing compile-time errors is easier than fixing runtime errors, which can be difficult to find.

What is collection class in Java ?

Java collection class is used exclusively with static methods that operate on or return collections. It inherits Object class. The important points about Java Collections class are: Java Collection class supports the polymorphic algorithms that operate on collections.

Frequently Asked Questions

Q #1) Is comparable a functional interface?

Answer: Yes, comparable is a functional interface. It declares a single abstract method, compareTo ().

Q #2) How do you make a class Comparable?

Answer: We make a class comparable by implementing the Comparable interface. Inside the class, we override the compareTo () method to sort the object. We pass an object and compare it with the current object.

The method compareTo () returns 0 if two objects are equal. It returns a negative number if the first object is less than the second and positive if the first object is greater than the second object.

Q #3) What is the main purpose of the Comparator interface?

Answer: Comparator interface is mainly used to sort the custom objects. A comparator interface can be used to compare objects of different classes as well. Also, the comparator interface can be used to sort objects in multiple fields.

Q #4) Why is Comparator used in Java?

Answer: Comparator interface is mainly used when we want a different sorting order for our custom objects other than natural ordering based on multiple fields.

Q #5) What implements the Comparable interface?

Answer: Comparable interface is implemented by all the wrapper classes and String class. Custom objects also use the comparable interface for sorting.