Collections Class

Collections class is utility class from java.util package which is specially design for perform the operations on List Collection or Collection. Collections class provides the some static method to us to perform operation on Collection.

Example: suppose consider we have the ArrayList and we want to perform sorting with ArrayList Then Collections class provide the sort () method to us to perform sorting on ArrayList.

Q. What is the diff between Collection and Collections?

1) Collection is interface from java.util package and Collections is class from java.util package.

2) Collection interface provide the implementation of data structure algorithm and Collections class is utility class which provide the static method to us to perform operation Collection Framework like as finding max element from collection, min element from collection, sort the collection etc these operation perform by Collections class.

Methods of Collections class

public static void sort(List): this method is used for sort the data from list collection. This is the overloaded method

public static void sort(List, Comparator): this method is used for sort the user defined objects .

public static void reverse(List): this method is used for reverse the list collection

public T min(java.util. Collection): this method can find the minimum element from collection

public T max(java.util. Collection) : this method is used for find the max element from collection.

Etc

Now we want to use the Collections.sort () method

This method is used for perform the sorting on List Collection.

Now we have the simple example we have the List collection with some values and we want to perform sorting on List Collection

Example

```
package org.techhub;
import java.util.*;
public class CollectionsApplication {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		List list = new ArrayList();
		list.add(100);
		list.add(400);
		list.add(300);
		list.add(200);
		list.add(900);
		list.add(700);
		list.add(50);
		System.out.println("Before Sorting");
		for(Object obj:list)
		{
			System.out.println(obj);
		}
		Collections.sort(list); //this method perform sorting operation automatically on Collection Framework
		System.out.println("After Sorting");
		for(Object obj:list)
		{
			System.out.println(obj);
		}
	}
}
```

We want to find the max element from List collection

For that we have the max() method of Collections class and it is also static method of Collections class.

```
package org.techhub;
import java.util.*;
public class CollectionsApplication {
```

```
	public static void main(String[] args) {

		List list = new ArrayList();
		  list.add(100);
		  list.add(400);
		  list.add(300);
		  list.add(200);
		  list.add(900);
		  list.add(700);
		  list.add(50);
		  System.out.println("Before Sorting");
		  for(Object obj:list)
		  {
			System.out.println(obj);
		  }
		  System.out.println("Max Element is "+Collections.max(list));
		  System.out.println("Min Element is "+Collections.min(list));
	}
}
```

WAP to create the collection of Employee Data and sort the all employee by using its id.
We want to employee detail empid, empname, empsal

Steps to implement the above program

1) Create the POJO class name as Employee with three fields id , name and sal

```
package org.techhub;
public class Employee {
	private int id;
	private String name;

	public int getId() {
```

```
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public int getSal() {
		return sal;
	}
	public void setSal(int sal) {
		this.sal = sal;
	}
	private int sal;
}
```

2) Create the class with main method and create the object of ArrayList

```
package org.techhub;
import java.util.*;
public class EmployeeStorageApplication {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		ArrayList al =new ArrayList();

	}
}
```

3) Create the objects of Employee class and store in ArrayList

```
package org.techhub;
import java.util.*;
public class EmployeeStorageApplication {
```

```
    public static void main(String[] args) {
            // TODO Auto-generated method stub
            ArrayList al =new ArrayList();
   Employee emp[]=new Employee[5]; //array of reference
   for(int i=0; i<emp.length;i++)
   {  Scanner xyz  = new Scanner(System.in);
     System.out.println("Enter the name id and salary of employee");
       String name=xyz.nextLine();
       int id=xyz.nextInt();
       int sal=xyz.nextInt();
     emp[i]=new Employee(name,id,sal); //Array of objects
     al.add(emp[i]);
   }
   System.out.println("Employee Records before sorting");
   for(Object obj:al)
   {
     Employee e =(Employee)obj;
     System.out.println(e.getId()+"\t"+e.getName()+"\t"+e.getSal());
   }
   Collections.sort(al);
   System.out.println("Employee Records After sorting");
   for(Object obj:al)
   {
     Employee e =(Employee)obj;
     System.out.println(e.getId()+"\t"+e.getName()+"\t"+e.getSal());
   }

  }
}
```

Note: above code generate the runtime exception to us
Why?

Because we use the Collections.sort() method in above code and we store the Employee class object in ArrayList so Collections.sort() method by default sort the list collection if collection contain primitive data type but here in our above code store the employee class object in ArrayList so Employee class

object is customize object it is not primitive data type so we get the exception at run time

Why Collections.sort () method not sort the by default user defined class object?

User defined object contain different type of data so Collections class cannot predict the sorting technique on object so it will raise at program run time means as per our example we have ArrayList and in ArrayList we store Employee class objects and Employee contain the three different type of data means it contain String ,integer and salary so we cannot predict sorting perform by using id or sorting perform by using salary or sorting perform by name so Collections.sort() method not perform sorting directly on User defined objects.

How to perform sorting on user defined by using Collections.sort () method

If we want to perform sorting on user defined objects we have the two interfaces in java
1) Comparable interface
2) Comparator interface.

Comparable interface

Comparable interface is used for perform the sorting with user defined objects with the help of Colloections.sort() method.

Steps to work with Comparable interface

1) add the java.lang package in application

Note: java.lang is default package of java so have to no need to import it

2) Create the POJO class and implement the Comparable interface in it.

```
package org.techhub;
public class Employee implements Comparable {
```

```
        private int id;
        private String name;

         public Employee()
         {

         }
        public Employee(String name,int id,int sal)
        {
                this.name=name;
                this.id=id;
                this.sal=sal;
        }
        public int getId() {
                return id;
        }
        public void setId(int id) {
                this.id = id;
        }
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }
        public int getSal() {
                return sal;
        }
        public void setSal(int sal) {
                this.sal = sal;
        }
        private int sal;
}
```

3) override the compareTo() method of Comparable interface in POJO class and perform the comparision

If we want to work with compareTo() method we have the some following rules.

1) If current object is greater than specified object then return positive integer

2) If current object is less than specified object then return negative integer

3) if current object is equal to specified object then return zero

Example

```
package org.techhub;
public class Employee implements Comparable  {
	private int id;
	private String name;

	 public Employee()
	 {

	 }
	public Employee(String name,int id,int sal)
	{
		this.name=name;
		this.id=id;
		this.sal=sal;
	}
	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public int getSal() {
		return sal;
```

```
	}
	public void setSal(int sal) {
		this.sal = sal;
	}
	private int sal;

	@Override
	public int compareTo(Object o) {
		// TODO Auto-generated method stub
		Employee e=(Employee)o;
		if(this.id>e.id)
		{
			return 1;
		}
		else if(this.id<e.id)
		{return -1;
		}
		else
		{
			return 0;
		}
		return 0;
	}
}
```

Main method class

```
package org.techhub;
import java.util.*;
public class EmployeeStorageApplication {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		ArrayList al =new ArrayList();
	Employee emp[]=new Employee[5]; //array of reference
	for(int i=0; i<emp.length;i++)
	{  Scanner xyz  = new Scanner(System.in);
	  System.out.println("Enter the name id and salary of employee");
```

```
            String name=xyz.nextLine();
            int id=xyz.nextInt();
            int sal=xyz.nextInt();
          emp[i]=new Employee(name,id,sal); //Array of objects
          al.add(emp[i]);
        }
        System.out.println("Employee Records before sorting");
        for(Object obj:al)
        {
          Employee e =(Employee)obj;
          System.out.println(e.getId()+"\t"+e.getName()+"\t"+e.getSal());
        }
        Collections.sort(al);
        System.out.println("Employee Records After sorting");
        for(Object obj:al)
        {
          Employee e =(Employee)obj;
          System.out.println(e.getId()+"\t"+e.getName()+"\t"+e.getSal());
      }

   }

}
```

Note: The major limitation of Comparable interface is to can't perform the sorting using more than one attribute of object.

Comparator interface

Comparator interface is used for perform the sorting with multiple attribute of objects

Suppose consider in above program we sort the employee record by using id of employee using Comparable interface by developer A and we have one more developer B and B want to sort the employee records by using its salary And if B developer try to modify the logic in compareTo() method of Comparable interface so Developer A logic may get vanish and if we want to solve this problem we have the one more interface for sorting purpose name as Comparator interface.

Steps to works with Comparator interface

1) add the java.util package in application
2) create the one more class and implement the Comparator interface in it and override its compare() method and write the comparison logics

```
package org.techhub;
import java.util.Comparator;
public class SortEmployeeBySal implements Comparator {
	@Override
	public int compare(Object o1, Object o2) {
		// TODO Auto-generated method stub
		Employee emp1 = (Employee) o1;
		Employee emp2 = (Employee) o2;
		if (emp1.getSal() > emp2.getSal()) {
			return 1;
		} else if (emp1.getSal() < emp1.getSal()) {
			return -1;
		} else {
			return 0;
		}
	}

}
```

3) use the Collections.sort() in following fashion

When we use the Comparator interface we have to use the Collections.sort () method in given fashion.

Syntax: Collections.sort (List,Comarator): as per this syntax we have to pass first parameter as list collection and second parameter child class object of Comparator interface.

```
package org.techhub;
import java.util.*;
public class EmployeeStorageApplication {
	public static void main(String[] args) {
		// TODO Auto-generated method stub
```

```
            ArrayList al =new ArrayList();
      Employee emp[]=new Employee[5]; //array of reference
      for(int i=0; i<emp.length;i++)
      {  Scanner xyz  = new Scanner(System.in);
        System.out.println("Enter the name id and salary of employee");
          String name=xyz.nextLine();
          int id=xyz.nextInt();
          int sal=xyz.nextInt();
        emp[i]=new Employee(name,id,sal); //Array of objects
        al.add(emp[i]);
      }
      SortEmployeeBySal s=new SortEmployeeBySal();
      Collections.sort(al,s);
      System.out.println("Employee Records After sorting");
      for(Object obj:al)
      {
        Employee e =(Employee)obj;
        System.out.println(e.getId()+"\t"+e.getName()+"\t"+e.getSal());
      }
  }

}
```