Introduction to Searching and Sorting

- **Comparable Interface**

Comparator Interface

The `Comparable` Interface

- The `Comparable` interface is in the `java.lang` package, and so is automatically available to any program
- It has only the following method heading that must be implemented:

```
 public int compareTo(Object other);
```

- It is the programmer's responsibility to follow the semantics of the `Comparable` interface when implementing it

The `Comparable` Interface Semantics

- The method `compareTo` must return
  - A negative number if the calling object "comes before" the parameter other
  - A zero if the calling object "equals" the parameter other
  - A positive number if the calling object "comes after" the parameter other
- If the parameter `other` is not of the same type as the class being defined, then a `ClassCastException` should be thrown

The `Comparable` Interface Semantics

- Almost any reasonable notion of "comes before" is acceptable
  - In particular, all of the standard less-than relations on numbers and lexicographic ordering on strings are suitable
- The relationship "comes after" is just the reverse of "comes before"

The Comparable Interface

- Several core Java classes implement Comparable interface.
- It is also preferable for object1.compareTo(object2) to return 0 if and only if object1.equals(object2) is true.

The Comparable Interface (cont'd)

- Example 1: A BankAccount defining the natural ordering as the ascending order of account numbers.

```
import java.util.*;
class BankAccount implements Comparable
{
    private int accountNumber;
    private String name;
    private double balance;
```

The Comparable Interface (cont’d)

```
public int compareTo(Object object)
   {
     BankAccount account = (BankAccount) object;
     if(accountNumber < account.accountNumber)
            return -1;
     else if(accountNumber == account.accountNumber)
            return 0;
     else
            return 1;
    }
```

The Comparable Interface (cont’d)

- Assuming that account1 and account2 are BankAccount objects, a typical call to the compareTo method is:

```
int comparisonResult = account1.compareTo(account2);
 if(comparisonResult == 0)
     System.out.println(“Same account”);
else if (comparisonResult < 0)
     System.out.println(“acountNumber1 is smaller”);
else
     System.out.println(“accountNumber2 is smaller”);
```

Example

```
public class Student implements
Comparable<Student> {

		private int roll;
		private String name;
		  //getter setter

		@Override
		public int compareTo(Student o) {
				if (this.roll > o.roll)
						return 1;
		   else if (this.roll < o.roll)
						return -1;
				else
						return 0;
		}
}
```

```
public class StudentSimulator {
        public static void
main(String[] args) {
                Student st[] =
new Student[3];


                int roll[] = {
1003, 1001, 1002 };
                String name[] =
{ "javed", "sakina", "mahmud" };

ArrayList<Student> list = new
ArrayList<Student>();
```

```
for (int i = 0; i < st.length; i++) {
        st[i] = new Student();
        st[i].setRoll(roll[i]);
        st[i].setName(name[i]);
                list.add(st[i]);
                }
System.out.println("\nStudent
Information \n");
for (Student student : list) {

System.out.println(student.getRoll(
) + "\t" + student.getName());
                }
        Collections.sort(list);
```

The Comparator Interface

- If we want to sort objects of a class which does not implement Comparable interface, or the class implements Comparable but we want To order its objects in a way different from the natural ordering defined by Comparable, the **java.util.Comparator** interface should be used.
- The Comparator interface is one of the **java collections framework** interfaces.

The Comparator Interface

- The **Java collection framework** is a set of important utility classes and interfaces in the **java.util** package for working with collections.
- A **collection** is a group of objects.
- **Comparator** interface defines how collection objects are compared.

The Comparator Interface

```
public interface Comparator
{
    int compare(Object object1, Object object2);
    boolean equals(Object object);
}
```

A class that implements Comparator should implement the compare method such that its returned value is:

0	if object1 “is equal to” object2
> 0	if object1 “is greater than” object2
< 0	if object1 “is less than” object2

The Comparator Interface (cont'd)

- It is also preferable for the compare method to return 0 if and only if object1.equals(object2) is true.
- The compare method throws a ClassCastException if the type of object1 and that of object2 are not compatible for comparison.

The Comparator Interface (cont'd)

- **Example 2**: This example sorts the strings in reverse order of the alphabetical one.

```
import java.util.*;
class StringReverseComparator implements Comparator
{
   public int compare(Object object1, Object object2)
   {
      String string1 = object1.toString();
      String string2 = object2.toString();
      // Reverse the comparison
      return string2.compareTo(string1);
    }
 }
```

The Comparator Interface (cont’d)

```
class Test
{
    public static void main(String[] args)
    {
      String[] array =
    {"Ahmad","Mohammad","Ali","Hisham","Omar",
    "Bilal","Hassan"};
      Arrays.sort(array, new
    StringReverseComparator());
    System.out.println(Arrays.asList(array));
    }
}
```

The Comparator Interface (cont'd)

-The sort method ,in the Arrays class, sorts the array "array" according to the comparator object. Notice the comparator object is provided as a parameter for the sorting method; it is an object from the class StringReverseComparator .

- After printing, we get the following order:

[Omar, Mohammad, Hisham, Hassan, Bilal, Ali, Ahmad]

Example

```
public class Employee
{
        private int id;
        private String name;
        private float salary;
//getter
//setter
```

```
class EmployeeSortBySal implements
Comparator<Employee> {
        @Override
        public int compare(Employee
o1, Employee o2) {
                return new
Float(o1.getSalary()).compareTo(o2.get-
Salary());
        }
}

class EmployeeSortById implements
Comparator<Employee> {
        @Override
        public int compare(Employee
o1, Employee o2) {
                return new
Integer(o1.getId()).compareTo(o2.getId())
;
        }
}
```

```
class EmployeeSortByName
implements
Comparator<Employee> {

@Override
public int compare(Employee o1,
Employee o2) {
return
o1.getName().compareTo(o2.get-
Name());
            }
}
```

```
public class EmployeeMain {
public static void main(String[] args) {
                        int id[] = { 1003, 1001,
1002 };
String name[] = { "rahma", "sakib",
"mahmud" };
float salary[] = { 20000, 10000, 30000 };
List<Employee> elist = new
ArrayList<Employee>();

Employee emp[] = new Employee[3];
for (int i = 0; i < emp.length; i++) {
emp[i] = new Employee();

emp[i].setId(id[i]);

emp[i].setName(name[i]);

emp[i].setSalary(salary[i]);
            elist.add(emp[i]);
                        }
```

```
Collections.sort(elist, new
EmployeeSortById());

System.out.println("Sorted
Employee list");

for (Employee employee : elist) {
System.out.println(employee.ge-
tId() + "\t" + employee.getName()
+ "\t" + employee.getSalary());
                    }
```

```
Collections.sort(elist, new
EmployeeSortBySal());

System.out.println("Sorted
Employee list by salary");

for (Employee employee : elist) {
System.out.println(employee.ge-
tId() + "\t" + employee.getName()
+ "\t" + employee.getSalary());
                    }
```