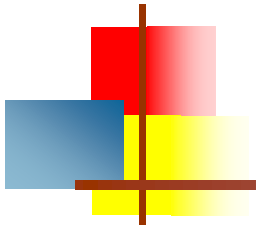
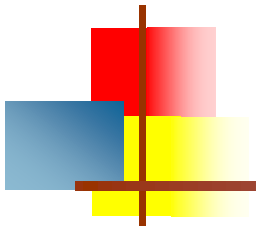
Comparable and Comparator



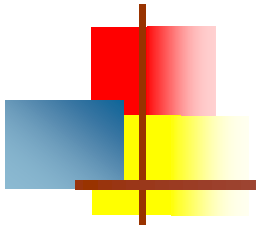
Comparing our own objects

- Four methods underlie many of Java's important Collection types: `equals`, `compare` and `compareTo`, and `hashCode`
  - To put your own objects into a Collection, you need to ensure that these methods are defined properly
  - Any collection with some sort of *membership test* uses `equals` (which, in many cases, defaults to `==`)
  - Any collection that depends on *sorting* requires larger/equal/smaller comparisons (`compare` or `compareTo`)
  - Any collection that depends on *hashing* requires both equality testing and hash codes (`equals` and `hashCode`)
  - Any time you implement `hashCode`, you *must* also implement `equals`
- Some of Java's classes, such as `String`, already define all of these properly for you
  - For your own objects, you have to do it yourself



Comparing our own objects

- The `Object` class provides `public boolean equals(Object obj)` and `public int hashCode()` methods
  - For objects that we define, the inherited `equals` and `hashCode` methods use the object's address in memory
  - We can override these methods
  - If we override `equals`, we *should* override `hashCode`
  - If we override `hashCode`, we *must* override `equals`
- The `Object` class does not provide any methods for "less" or "greater"—however,
  - There is a `Comparable` interface in `java.lang`
  - There is a `Comparator` interface in `java.util`

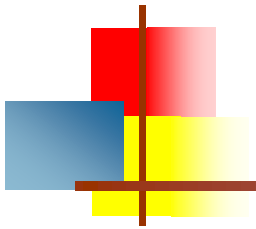


Comparable

- Implemented by a class of objects you want to compare (i.e. Students, Rectangles, Aliens, etc.)
- The interface requires one method:

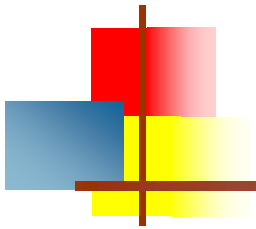
```
public int compareTo(Object o)
```

- The `compareTo` method must return
  - Negative number if the calling object "comes before" the parameter
  - Zero if the calling object "equals" the parameter other
  - Positive number if the calling object "comes after" the parameter other



Example Using `Student` Class

```
public class Student implements Comparable <Student>
{
   public Student(String name, int score) {...}

   public int compareTo(Object o) {...}

   public String getName() {. . . }
   public int getScore() { . . . }
   public void setName(String name) {. . . }
   public void setScore(int score) {. . .}

   // other methods
   . . .
}
```



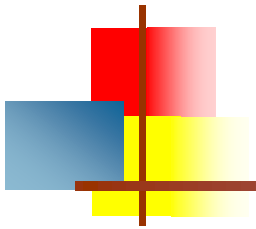
Example Using `Student` Class

- Nothing special here:

```
public Student(String name, int score)
{
        this.name = name;
        this.score = score;
}
```

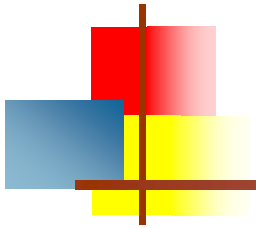
- Sort students according to score

```
public int compareTo(Student arg0)
{
        return this.name.compareTo(arg0.getName());
}
```



Using `Student` Class

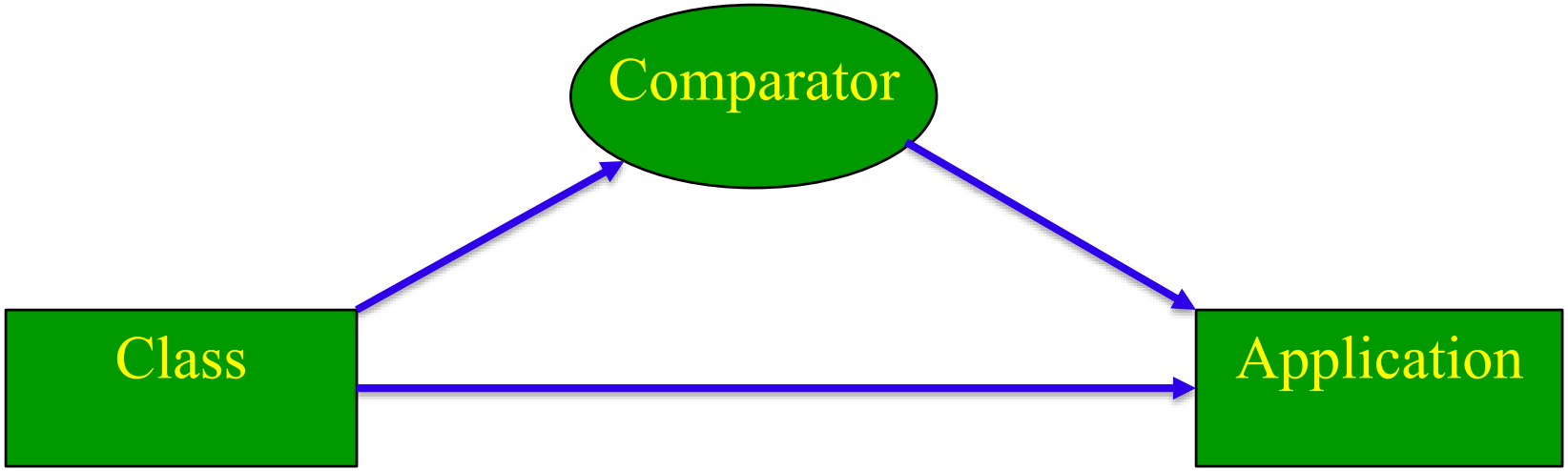
```
public static void main(String args[])
{
      List<Student> students = new ArrayList<Student>();

      students.add(new Student("Ann", 87));
      students.add(new Student("Bob", 83));
      students.add(new Student("Cat", 99));
      students.add(new Student("Dan", 25));
      students.add(new Student("Eve", 76));
      Collections.sort(students);
      Iterator<Student> itr = students.iterator();
      while (itr.hasNext())
       {
            Student s = itr.next();
            System.out.println(s.name + "   " +
                                              s.score);
       }
 }
```



Comparator

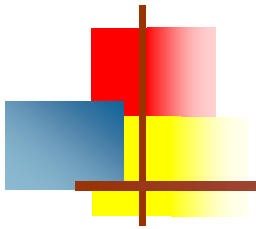
- Comparator Interface
  - Standard that is applied to describe a **problem dependent ordering** of a class.
  - Implemented **outside** the class with a class that contains a ***compare*** method that maps a total ordering onto the objects in the target class.
  - Frequently the ***`compare`*** method makes use of the ***`compareTo`*** methods of a class's instance variables

Comparator



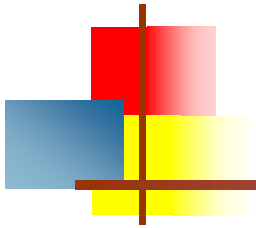
Using `Student` Class

```
public static void main(String args[])
{
      Set students = new TreeSet <new MyComparator()>();

      students.add(new Student("Ann", 87));
      students.add(new Student("Bob", 83));
      students.add(new Student("Cat", 99));
      students.add(new Student("Dan", 25));
      students.add(new Student("Eve", 76));
      Collections.sort(students);
      Iterator<Student> itr = students.iterator();
      while (itr.hasNext())
       {
            Student s = itr.next();
            System.out.println(s.name + "   " +
                                              s.score);
       }
 }
```



Using `Student` Class

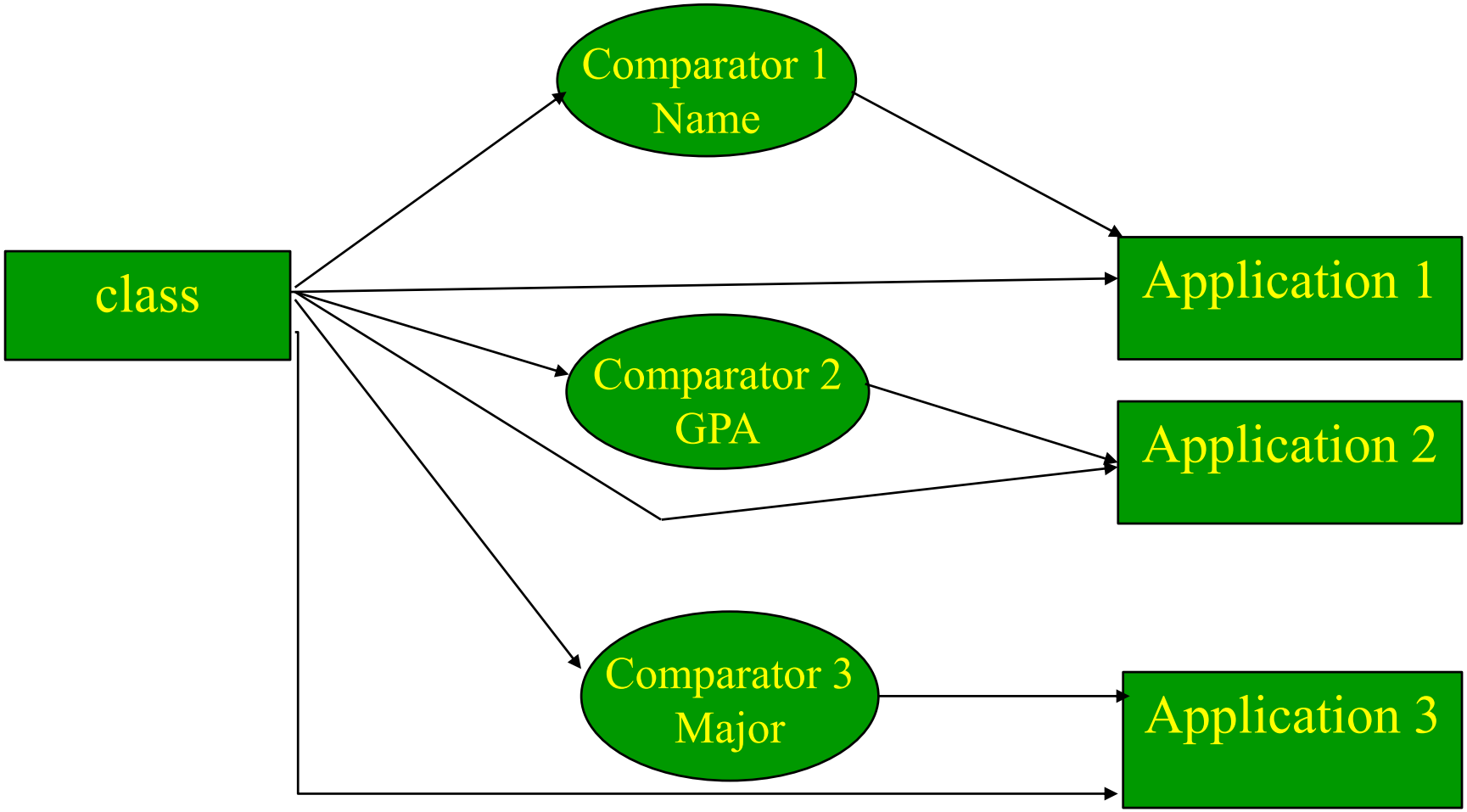
```
public class MyComparator implements Comparator <JustStudent>
{
    @Override
    public int compare(JustStudent arg0, JustStudent arg1)
    {
        // TODO Auto-generated method stub
        if (arg0.getAge() < arg1.getAge())
                return -1;
        else if (arg0.getAge() > arg1.getAge())
                return +1;
        else
                return 0;
    }
}
```



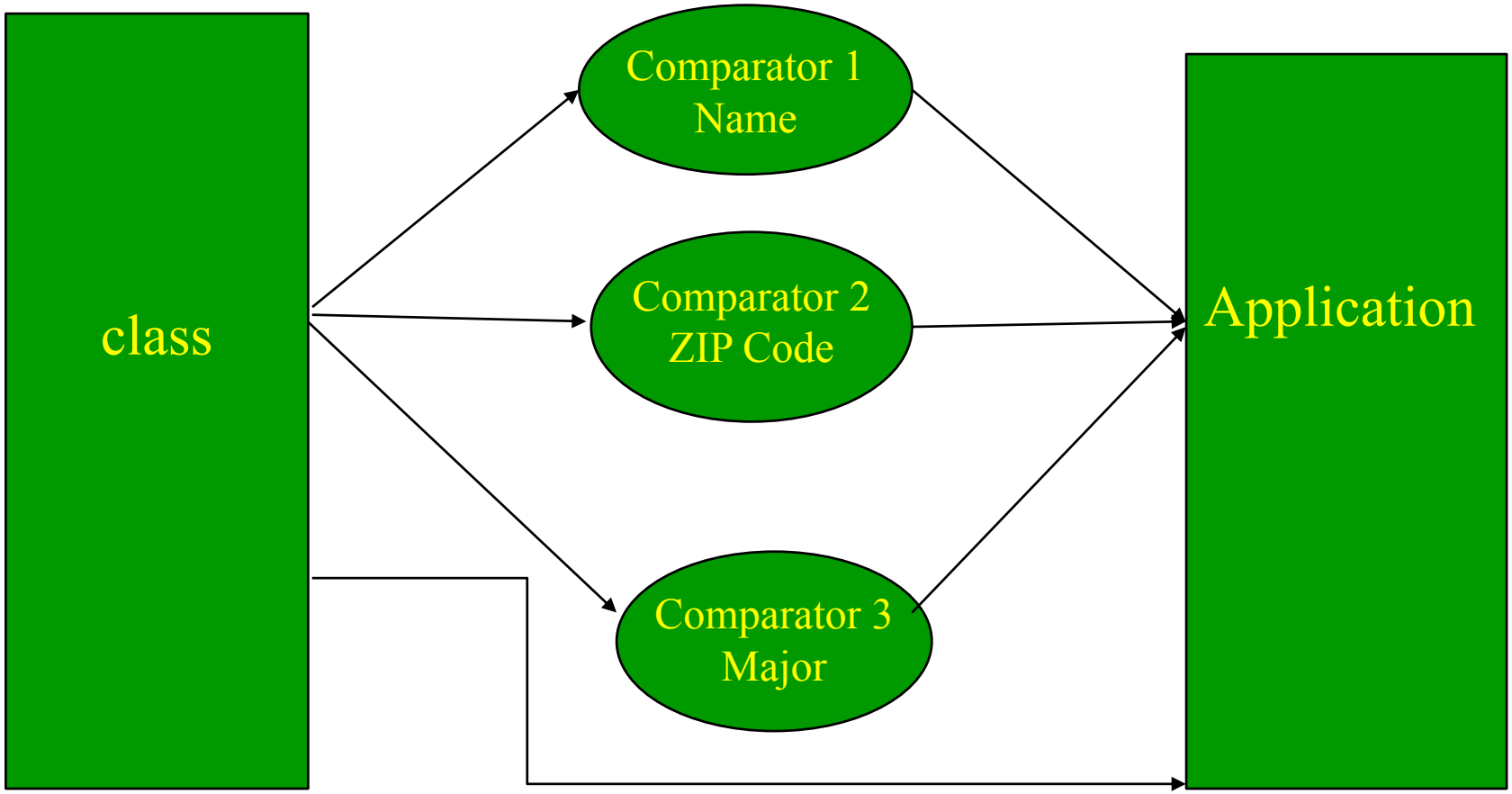
Comparator

- Student Class: Possible orderings
  - Alpha by last name
  - By class (no. of credits)
  - By ZIP
  - By major
  - By various combinations of instance variables

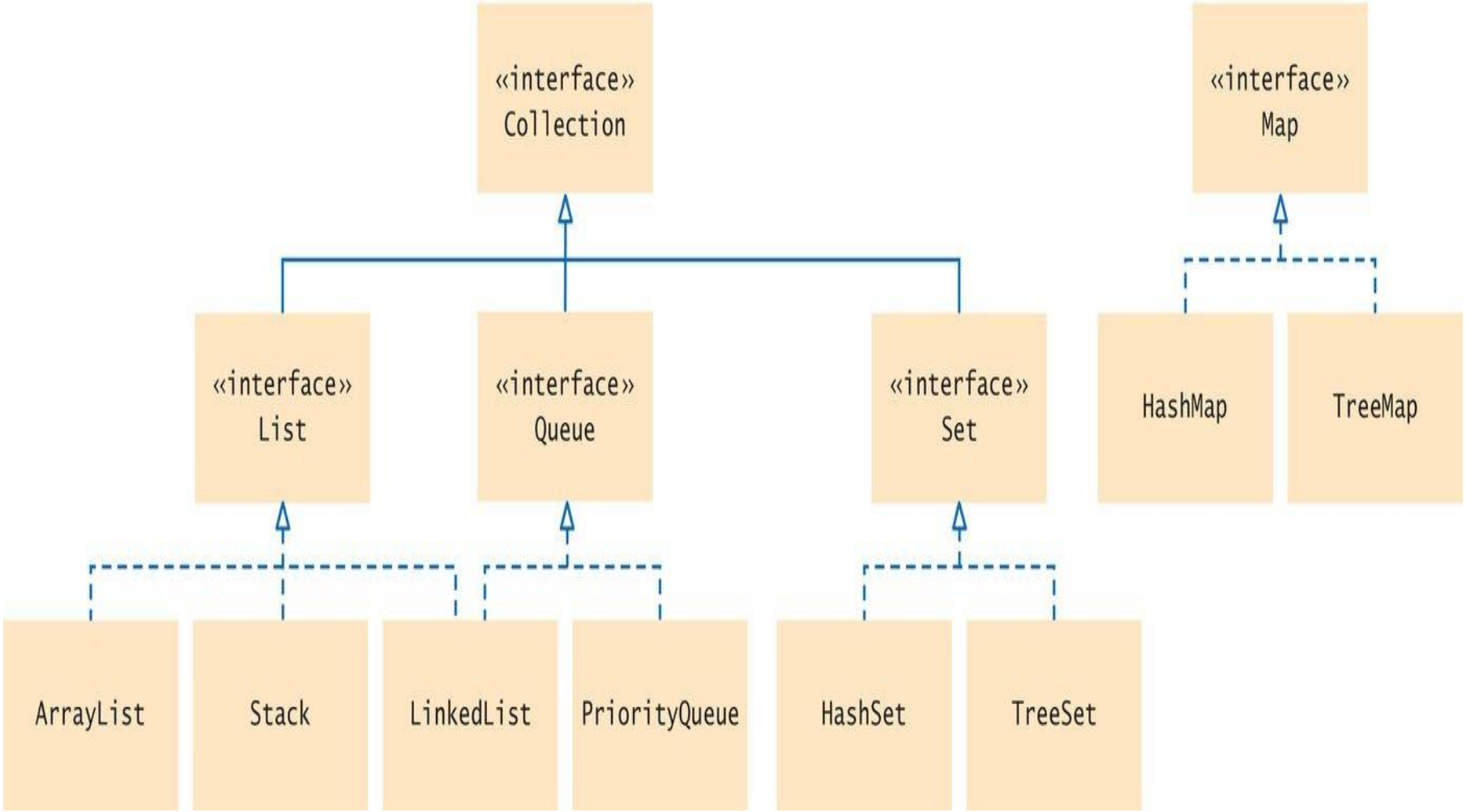
Comparator


Comparator 1
Name
class
Application 1
Comparator 2
GPA
Application 2
Comparator 3
Major
Application 3

Comparator


class
Comparator 1
Name
Comparator 2
ZIP Code
Comparator 3
Major
Application

An Overview of the Collections Framework

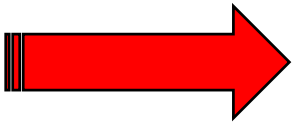


Comparator for changing the natural sorting order

- Write a program to insert integer objects into a TreeSet where the sorting order is descending order?

  [10, 0, 15, 5, 20, 20]

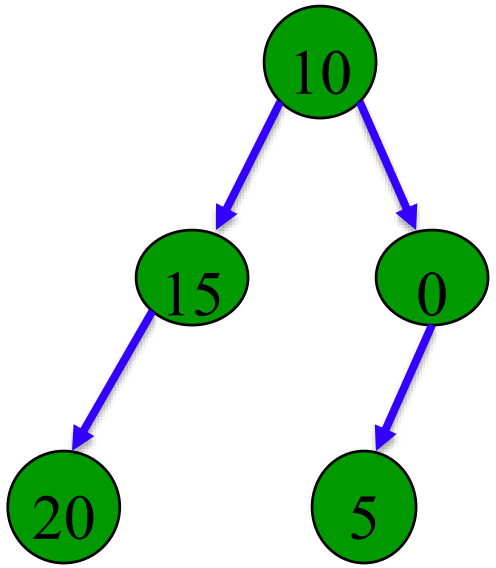
```
public class WithoutComp {
    public static void main (String arg[]){
        TreeSet tree = new TreeSet();
        tree.add(10);
        tree.add(0);
        tree.add(15);
        tree.add(5);
        tree.add(20);
        tree.add(20);
        System.out.println(tree);
    }
}
```



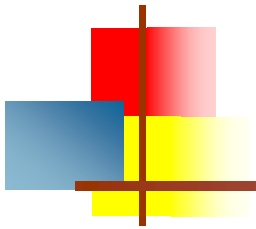
[0, 5, 10, 15, 20]

Comparator

```
class MyComprator implements Comparator{

    @Override
    public int compare(Object obje1, Object obje2) {
        // TODO Auto-generated method stub
        Integer I1 = (Integer) obje1;
        Integer I2 = (Integer) obje2;
        if (I1 < I2)
            return +1;
        else if (I1 > I2)
            return -1;
        else
            return 0;
    }
}
```



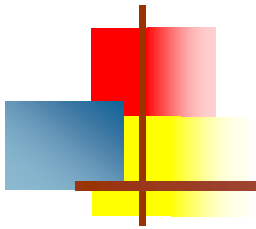
Left Root Right



Comparator

- It's possible to use the compareTo method from comparable inside the compare method.

```
public int compare(Object obje1, Object obje2) {
        Integer I1 = (Integer) obje1;
        Integer I2 = (Integer) obje2;
        return  I1.compareTo(I2); //[0, 5, 10, 15, 20]
    }
```



Comparator

```
public int compare(Object obje1, Object obje2) {
        Integer I1 = (Integer) obje1;
        Integer I2 = (Integer) obje2;
        return  I1.compareTo(I2);   [0, 5, 10, 15, 20]
        return  -I1.compareTo(I2); [20, 15, 10, 5, 0]
        return  I2.compareTo(I1);   [20, 15, 10, 5, 0]
        return  -I2.compareTo(I1); [0, 5, 10, 15, 20]
        return -1; [20, 20, 5, 15, 0, 10]
        return +1; [10, 0, 15, 5, 20, 20]
    }
```

Comparable and Comparator

Comparable	Comparator
It meant for default natural sorting order	It meant for customized sorting order
Present in Java.Lang Package	Present in Java.Util Package
The interface define only one method compareTo()	The interface define two different methods compare() and equal()