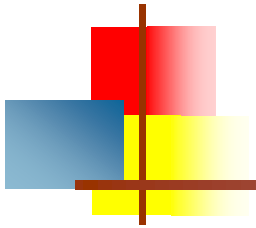
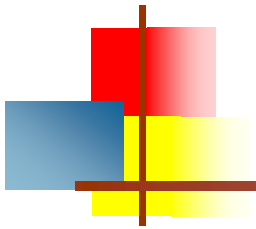
Comparable and Comparator



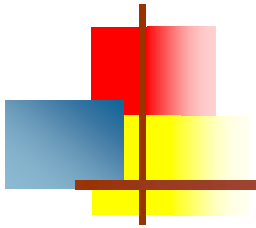
Comparing our own objects

- 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`

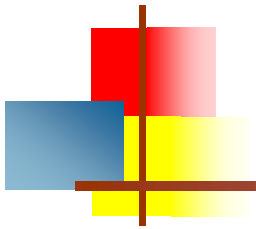


Comparable

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

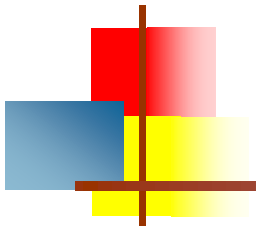
```
public int compareTo(Object o)
```

- 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
   . . .
}
```



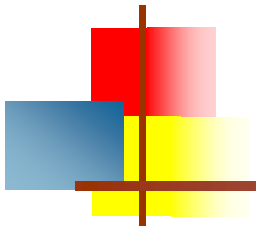
Example Using `Student` Class

- Nothing special here:

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

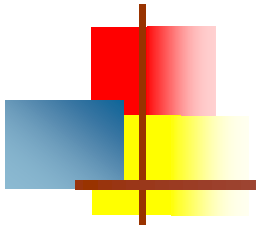
- Sort students according to score

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



Using `Student` Class

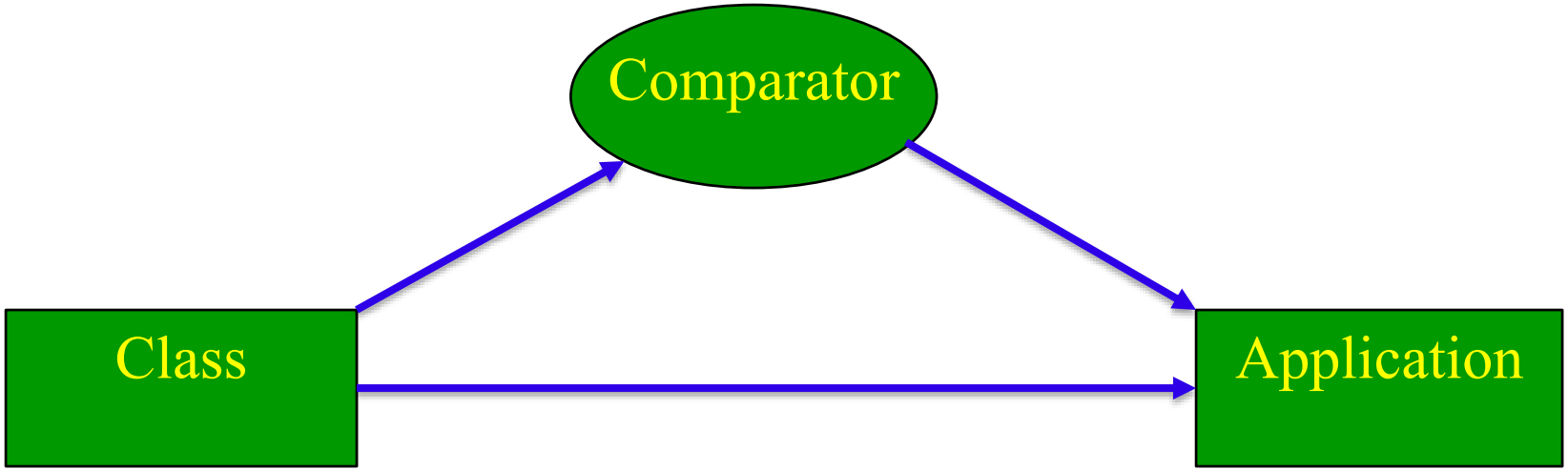
```
public static void main(String args[])
{
      List<Student> students = new ArrayList<Student>();

      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

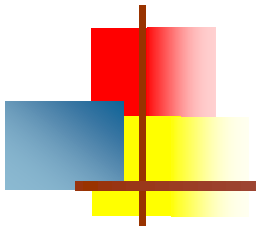
- Comparator Interface
  - Standard that is applied to describe a **problem dependent ordering** of a class.
  - Implemented **outside** the class with a class that contains a ***compare*** method that maps a total ordering onto the objects in the target class.
  - Frequently the ***`compare`*** method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

Comparator



Using `Student` Class

```
public static void main(String args[])
{
      Set <Student> 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;
    }
}
```