Interfaces IIII

Comparator continued

Another use case for comparators

What if we make CDs comparable on, say, the title? Then we could sort an array of CDs using the first version of Arrays.sort() which only takes one argument, the reference to the array.

Fine. But what if we want to add the flexibility of having two ways of comparing to CDs? One which is the default, looking at title, and an optional way which uses only the year?

Then we can use the CDYearComparator again!

CD as Comparable

```
public class CD implements Comparable<CD>{
  private String title;
  private String year;
  public CD(String title, String year){
    this.title=title;
    this.year=year;
  }
  public String title(){return title;}
  public String year(){return year;}


  @Override public int compareTo(CD other){
    return title.compareTo(other.title);
  }
  @Override
  public String toString(){
    return String.format("%s, %s",title, year);
  }
}
```

A closer look at CD as Comparable

```
public class CD implements Comparable<CD>{
  //... constructors etc

  @Override
  public int compareTo(CD other){
    return title.compareTo(other.title); // Strings are comparable!
  }
}
```

Some test cases

```
CD first= new CD("Violator", "1990");
CD second = new CD("On the beach", "1969");
CD third = new CD("Forever young", "1984");
CD fourth = new CD("Diamond dogs", "1974");
CD[] cds = new CD[4];
cds[0]=first;
cds[1]=second;
cds[2]=third;
cds[3]=fourth;
System.out.print("Unsorted: ");
System.out.println(java.util.Arrays.toString(cds));
java.util.Arrays.sort(cds);
System.out.print("Default sorted (title): ");
System.out.println(java.util.Arrays.toString(cds));
java.util.Arrays.sort(cds,new CDYearComparator());
System.out.print("Sorted on year: ");
System.out.println(java.util.Arrays.toString(cds));
```

Some test cases - a closer look

```
CD first= new CD("Violator", "1990");
CD second = new CD("On the beach", "1969");
CD third = new CD("Forever young", "1984");
CD fourth = new CD("Diamond dogs", "1974");
CD[] cds = new CD[4];
cds[0]=first;
cds[1]=second;
cds[2]=third;
cds[3]=fourth;
System.out.print("Unsorted: ");
System.out.println(java.util.Arrays.toString(cds));
java.util.Arrays.sort(cds); // default sort
System.out.print("Default sorted (title): ");
System.out.println(java.util.Arrays.toString(cds));
java.util.Arrays.sort(cds,new CDYearComparator()); // using a comparator
System.out.print("Sorted on year: ");
System.out.println(java.util.Arrays.toString(cds));
```

Bonus - alternative ways to create arrays

```
public class TestCDYearComp2{
  public static void main(String[] args){
    CD[] cds = {
      new CD("Violator", "1990"),
      new CD("On the beach", "1969"),
      new CD("Forever young", "1984"),
      new CD("Diamond dogs", "1974")
    };
    System.out.print("Unsorted: ");
    System.out.println(java.util.Arrays.toString(cds));
    java.util.Arrays.sort(cds);
    System.out.print("Default sorted (title): ");
    System.out.println(java.util.Arrays.toString(cds));
    java.util.Arrays.sort(cds,new CDYearComparator());
    System.out.print("Sorted on year: ");
    System.out.println(java.util.Arrays.toString(cds));
  }
}
```

Arrays and lists only store references to objects

There is no need to have reference variables in order to store an object reference in an array (or some kind of list).

The call to new is evaluated to a reference, which can be stored directly in the array:

```
CD[] cds = {
  new CD("Violator", "1990"),        // note the comma!
  new CD("On the beach", "1969"),
  new CD("Forever young", "1984"),
  new CD("Diamond dogs", "1974")     // no comma for last element
};  // note the semicolon!
```