1. **comparable** Implement a `compareTo` method for each of the following classes:

(a)
```
public class Shoe implements Comparable<Shoe> {
  private int size;

  @Override
  public int compareTo(Shoe other) {
    // compare shoes by size

    // the easy way
    return Integer.compare(this.size, other.size);
  }
}
```

```
public class Shoe implements Comparable<Shoe> {
  private int size;

  @Override
  public int compareTo(Shoe other) {
    // compare shoes by size

    // the hard way
    int result = 0;
    if (this.size < other.size) {
        result = -1;   // or any other negative integer
    }
    else if (this.size > other.size) {
        result = 1;    // or any other positive integer
    }
    return result;
  }
}
```

(b)
```
public class TimeOfDay implements Comparable<TimeOfDay> {
  private int hour;    // 0-23 the hour of the day
  private int minute;  // 0-59 the minute of the hour

  @Override
  public int compareTo(TimeOfDay other) {
    // compare times by hour then minute

    int result = Integer.compare(this.hour, other.hour);
    if (result == 0) {
        // times have the same the same hour; we need to
        // examine the minutes to determine the final answer
        result = Integer.compare(this.minute, other.minute);
    }
    return result;
  }
}
```

(c)
```
public class Card implements Comparable<Card> {
  private Rank rank;    // the rank of the card 2-10, J, Q, K, A
  private Suit suit;    // the suit of the card
  @Override
  public int compareTo(Card other) {
    // compare cards by their rank
    // the integer value of this.rank can be obtained
    // as this.rank.ordinal()

    // note that the suit is not used in the comparison so
    // comparing the queen of hearts and the queen of diamonds
    // would return zero
    return Integer.compare(this.rank.ordinal(), other.rank.ordinal());
  }
}
```

2. **compareTo contract**

   Consider your `compareTo` method for `Shoe`:

   (a)
   ```
   Shoe x = new Shoe(8);     // size 8
   Shoe y = new Shoe(11);    // size 11
   ```

   What is the sign of:

   i. `x.compareTo(y)`

   Solution: negative

   ii. `y.compareTo(x)`

   Solution: positive

   (b)
   ```
   Shoe x = new Shoe(7);     // size 7
   Shoe y = new Shoe(4);     // size 4
   ```

   What is the sign of:

   i. `x.compareTo(y)`

   Solution: positive

   ii. `y.compareTo(x)`

   Solution: negative

   (c)
   ```
   Shoe x = new Shoe(7);     // size 7
   Shoe y = new Shoe(7);     // size 7
   ```

   What is the value of:

   i. `x.compareTo(y)`

   Solution: zero

   ii. `y.compareTo(x)`

   Solution: zero

   (d) Analyze your answers for parts (a)–(c); does your `compareTo` method satisfy Part 1 of the compareTo contract?

   Solution: Yes, the signs of the returned values flip when the order of the two compared objects are reversed.

(e)
```
Shoe x = new Shoe(8);     // size 8
Shoe y = new Shoe(8);     // size 8
Shoe z = new Shoe(10);    // size 10
```

What is the sign of:

i. `x.compareTo(y)`

Solution: zero (the question should be what is the value of `x.compareTo(y)`)

ii. `x.compareTo(z)`

Solution: negative

iii. `y.compareTo(z)`

Solution: negative

(f)
```
Shoe x = new Shoe(8);     // size 8
Shoe y = new Shoe(8);     // size 8
Shoe z = new Shoe(4);     // size 4
```

What is the sign of:

i. `x.compareTo(y)`

Solution: zero (the question should be what is the value of `x.compareTo(y)`)

ii. `x.compareTo(z)`

Solution: positive

iii. `y.compareTo(z)`

Solution: positive

(g) Does your `compareTo` method satisfy Part 3 of the compareTo contract?

Solution: Yes, the signs of `x.compareTo(z)` and `y.compareTo(z)` are the same.

3. **Static fields**

   Modify the `Shoe` class shown below so that it keeps track of the number of shoes created. Make sure that a client is able to obtain the number of shoes created.

```
public class Shoe {
  private int size;

  private static int numberOfShoes = 0;


  public Shoe() {
    this.size = 8;
    Shoe.numberOfShoes++;
  }

  public Shoe(int size) {
    // possibly validate size here
    this.size = size;
    Shoe.numberOfShoes++;
  }

  public Shoe(Shoe other) {
    this(other.size);   // constructor chain
    // don't increment Shoe.numberOfShoes here!
    // the chained constructor already increments the number of shoes
  }

  public static int getNumberOfShoes() {
    return Shoe.numberOfShoes;
  }


}
```