1 Exercises

Exercise 1. Implement a comparable data type called `Card` that represents a playing card. Each card is represented using an integer *rank* (0 for 2, 1 for 3, . . . , 11 for King, and 12 for Ace) and an integer *suit* (0 for Clubs, 1 for Diamonds, 2 for Hearts, and 3 for Spades). Your data type must support the following API:

`Card implements Comparable<Card>`	
`Card(int suit, int rank)`	constructs a card given its suit and rank
`boolean equals(Object other)`	returns `true` if this card and `other` are the same, and `false` otherwise
`String toString()`	returns a string representation of this card; for example "Ace of Hearts"
`int compareTo(Card that)`	returns a comparison† of this card with `other`
`static Comparator<Card> suitOrder()`	returns a comparator for comparing two cards by suit
`static Comparator<Card> reverseRankOrder()`	returns a comparator for comparing two cards in reverse order of rank

† A card c_1 is less than a card c_2 if either the suit of c_1 is less than the suit of c_2 *or* c_1 and c_2 are of the same suit *and* the denomination of c_1 is less than the denomination of c_2. We refer to this as the *natural order*.

Exercise 2. Suppose `deck` is an array of `Card` (the data type from Problem 1) objects.

a. Write down a statement that uses `Arrays.sort()` to sort `deck` by the natural order.

b. Write down a statement that uses `Arrays.sort()` to sort `deck` by suit order.

c. Write down a statement that uses `Arrays.sort()` to sort `deck` by reverse rank order.

Exercise 3. Complete the implementation of the iterable data type called `Range` that allows clients to iterate over a range of integers from the interval $[lo, hi]$ in increments specified by *step*. For example, the following code

```
for (int i : new Range(1, 10, 3)) {
    StdOut.println(i);
}
```

should output

```
1
4
7
10
```

Range.java

```
import java.util.Iterator;

public class Range implements Iterable<Integer> {
    private int lo;
    private int hi;
    private int step;

    // Constructs a range given the bounds and the step size.
    public Range(int lo, int hi, int step) {
        ...
    }

    // Returns a range iterator.
    public Iterator<Integer> iterator() {
        ...
    }

    // Range iterator.
    private class RangeIterator implements Iterator<Integer> {
        private int i; // the current number in the range

        // Constructs an iterator.
        public RangeIterator() {
            ...
        }

        // Returns true if there are more numbers in the range, and false otherwise.
        public boolean hasNext() {
            ...
        }

        // Returns the next number in the range.
        public Integer next() {
            ...
        }
    }
}
```

2 Solutions

Solution 1.

Card.java

```java
import java.util.Comparator;

public class Card implements Comparable<Card> {
    private static String[] RANKS = {"2", "3", "4", "5", "6", "7", "8", "9",
                                     "10", "Jack", "Queen", "King", "Ace"};
    private static String[] SUITS = {"Clubs", "Diamonds", "Hearts", "Spades"};
    private int suit;
    private int rank;

    public Card(int suit, int rank) {
        this.suit = suit;
        this.rank = rank;
    }

    public boolean equals(Card other) {
        return compareTo(other) == 0;
    }

    public String toString() {
        return RANKS[rank] + " of " + SUITS[suit];
    }

    public int compareTo(Card that) {
        if (suit < that.suit || suit == that.suit && rank < that.rank) {
            return -1;
        }
        else if (suit == that.suit && rank == that.rank) {
            return 0;
        }
        else {
            return 1;
        }
    }

    public static Comparator<Card> suitOrder() {
        return new SuitOrder();
    }

    public static Comparator<Card> reverseRankOrder() {
        return new ReverseRankOrder();
    }

    private static class SuitOrder implements Comparator<Card> {
        public int compare(Card c1, Card c2) {
            return c1.suit - c2.suit;
        }
    }

    private static class ReverseRankOrder implements Comparator<Card> {
        public int compare(Card c1, Card c2) {
            return c2.rank - c1.rank;
        }
    }
}
```

Solution 2.

a. `Arrays.sort(deck);`

b. `Arrays.sort(deck, Card.suitOrder());`

c. `Arrays.sort(deck, Card.reverseRankOrder());`

Solution 3.

Range.java

```java
import java.util.Iterator;

public class Range implements Iterable<Integer> {
    private int lo;
    private int hi;
    private int step;

    // Constructs a range given the bounds and the step size.
    public Range(int lo, int hi, int step) {
        this.lo = lo;
        this.hi = hi;
        this.step = step;
    }

    // Returns a range iterator.
    public Iterator<Integer> iterator() {
        returns new RangeIterator();
    }

    // Range iterator.
    private class RangeIterator implements Iterator<Integer> {
        private int i; // the current number in the range

        // Constructs an iterator.
        public RangeIterator() {
            i = lo;
        }

        // Returns true if there are more numbers in the range, and false otherwise.
        public boolean hasNext() {
            i <= hi;
        }

        // Returns the next number in the range.
        public Integer next() {
            int next = i;
            i += step;
            return next;
        }
    }
}
```