Ulster University

COM410 Programming in Practice

B2.3 Sorting in Java

Java Provision for Sorting

- Java provides comprehensive support for sorting through built-in methods
- Underlying sorting algorithms are **O(n log n)**
- Mixture of Merge Sort, QuickSort and TimSort (depending on use and version of JDK)
- Four cases to consider
    - Arrays of primitives
    - Collections
    - Sorting custom objects on fields
    - Flexible sorting

Arrays of Primitives

- Using method `Arrays.sort()`
- Works on numeric primitive types (`int`, etc.) and `String`s

```
int[] intArray = { 3, 6, 2, 9, 1 };
Arrays.sort( intArray );

String strArray = { "one", "two", "three", "four", "five" };
Arrays.sort( strArray );
```

Arrays of Primitives

- Descending order by providing a `Comparator` as the 2nd parameter
- Must be sorting an `Object` type

```
Integer[] integerArray = { 3, 6, 2, 9, 1 };
Arrays.sort( integerArray, Comparator.reverseOrder() );

String strArray = { "one", "two", "three", "four", "five" };
Arrays.sort( strArray, Comparator.reverseOrder() );
```

Collections

- Sort collections such as `ArrayList` and `LinkedList` using the `Collections.sort()` method
- Must be sorting an `Object` type

```
ArrayList<Integer> integerList = new ArrayList<Integer>();
integerList.add( 40 );
integerList.add( 20 );
integerList.add( 30 );

Collections.sort( integerList );
Collections.sort( integerList, Comparator.reverseOrder() );
```

Custom Objects

- Class to be sorted must implement the **`Comparable`** interface and implement the method **`compareTo()`**

```
public class Person implements Comparable<Person> {
  private String name;
  private int age;

  ...

  public int compareTo(Person anotherPerson) {
     return this.age - anotherPerson.getAge();
  }
}
```

```
ArrayList<Person> people = new ArrayList<Person>();
...
Collections.sort( people );
```

Flexible Sorting

- Provide different comparators to sort on different object fields

```
public class Person implements Comparable<Person> {

  public static Comparator<Person> ageComparator = new Comparator<Person>() {

     public int compare(Person p1, Person p2) {
        return p1.getAge() - p2.getAge();
     }
  }
}
```

```
ArrayList<Person> people = new ArrayList<Person>();
...
Collections.sort( people, Person.ageComparator );
```

Challenge

- **Cardball** is a sports simulation game in which each player is dealt 5 playing cards from a shuffled pack. To play the game, each player is required to sort the cards in their hand and play them in descending order of rank value. After each player has played a card, the player who played the card with the highest value is awarded a goal. If more than one player plays the highest valued card, then no goal is scored in that round.
- Develop a class `Cardball` within your `DataStructures` project to play an automated 2-player game (i.e. one with no user input) according to the rules above
    - The game should run in the `main()` method, while the class can contain any supporting variables or methods that are required.