Comparing Objects

Learning Objectives

- Be able to implement the `Comparable` interface
- Be able to use the `compareTo` method to compare objects

Poll:

What does the array `{2, 0, 1}` look like after we pass it in as input to `mystery()`?

```
public static void mystery(int[] arr) {
    for (int i = 0; i < arr.length; i++) {
        int min = i;
        for (int j = i + 1; j < arr.length; j++) {
            if (arr[j] < arr[min]) {
                min = j;
            }
        }
        if (i != min) {
            int temp = arr[i];
            arr[i] = arr[min];
            arr[min] = temp;
        }
    }
}
```

Common Java Object Methods

Four methods are essential for ordering & comparing Java objects

- `equals`: used for defining when two objects are *structurally* equal to each other
- `hashCode`: you'll learn about it in a future course
- `compare`, `compareTo`: for today!

Many built-in Java objects (like String) define these for you.

- For your own objects, you'll need to define them yourself.

Poll:

If we are trying to order the strings `"Film"` and `"Movie"`, which comes first?

`compareTo`

Way back when, we've already seen `compareTo` for Strings!

```
firstString.compareTo(secondString);
```

returns:

- `0` if both Strings are equal
- a negative number if `firstString` is **less than** (comes before) `secondString`
- a positive number if `firstString` is **greater than** (comes after) `secondString`

`compareTo`

`compareTo` returns:

- `0` if both Strings are equal
- a negative number if `firstString` is **less than** (comes before) `secondString`
- a positive number if `firstString` is **greater than** (comes after) `secondString`

```
"apple".compareTo("banana"); // -1
"banana".compareTo("apple"); // 1
"apple".compareTo("apple"); // 0
```

Poll:

How could we change this code so that it can sort an array of Strings (instead of ints?)

```
public static void mystery(int[] arr) {
    for (int i = 0; i < arr.length; i++) {
        int min = i;
        for (int j = i + 1; j < arr.length; j++) {
            if (arr[j] < arr[min]) {
                min = j;
            }
        }
        if (i != min) {
            int temp = arr[i];
            arr[i] = arr[min];
            arr[min] = temp;
        }
    }
}
```

Solution:

```
public static void mystery(String[] arr) {
    for (int i = 0; i < arr.length; i++) {
        int min = i;
        for (int j = i + 1; j < arr.length; j++) {
            if (arr[j].compareTo(arr[min]) < 0) {
                min = j;
            }
        }
        if (i != min) {
            String temp = arr[i];
            arr[i] = arr[min];
            arr[min] = temp;
        }
    }
}
```

Activity: Imagine that we have a `Student` class that stores a `firstName`, `lastName`, `program`, and `programGPA`.

- We want to be able to compare two `Student` objects by their last names.
- If they have the same last name, then break ties by comparing their first names.
- If they have the same first & last name, then they are the same student.
    - (Speaking as a "Harry Smith" in the US, this is a gross oversimplification...)

```
public class Student {
    private String firstName, lastName, program;
    private double programGPA;
    public int compareTo(Student other) {
        return -1; // TODO
    }
}
```

One Solution

```
public int compareTo(Student other) {
    if (this.lastName.compareTo(other.lastName) < 0) {
        return -1;
    } else if (this.lastName.compareTo(other.lastName) > 0) {
        return 1;
    } else {
        if (this.firstName.compareTo(other.firstName) < 0) {
            return -1;
        } else if (this.firstName.compareTo(other.firstName) > 0) {
            return 1;
        } else {
            return 0;
        }
    }
}
```

A Concise Solution

```
public int compareTo(Student other) {
    if (this.lastName.equals(other.lastName)) {
        return this.firstName.compareTo(other.firstName);
    } else {
        return this.lastName.compareTo(other.lastName);
    }
}
```

The `Comparable` ADT

- Built-in Java interface
- Defines a single abstract method for comparison: `compareTo`
  - By definition of interfaces, any class that implements `Comparable` must implement `compareTo`
- Objects of a class that implements `Comparable` are "sortable"
  - **If a class implements `Comparable`, other built-in Java libraries will know how to make use of it!**
  - e.g. makes `Arrays.sort(...)` possible automatically!

The `Comparable` ADT

`compareTo` compares two objects for ordering:

- returns a negative `int` if the *object on which the method is invoked* is less than the *object passed as a parameter*.
- returns `0` if the object on which the method is invoked is equal to the object passed as a parameter.
- returens a positive `int` if the object on which the method is invoked is greater than the object passed as a parameter.

```
Object obj1; // the object that the method is invoked on in this example
Object obj2; // the object passed as a parameter in this example
obj1.compareTo(obj2);
```

Making an Object Sortable

Simply implement `Comparable!`

- tells Java "*this object can be sorted!*"
- `Comparable` is generically typed, so you have to specify the type 👱

```
public class Student implements Comparable<Student> {
    private String firstName, lastName, program;
    private double programGPA;
    public int compareTo(Student other) {
        if (this.lastName.equals(other.lastName)) {
            return this.firstName.compareTo(other.firstName);
        } else {
            return this.lastName.compareTo(other.lastName);
        }
    }
    // ... other methods ...
}
```

Implementing `Comparable`

- Mark that the class `implements Comparable<ClassName>`
- Implement `compareTo` returning a -ve, `0`, or +ve value in the correct cases
- Keep in mind that the magnitude of the return value doesn't matter, just the sign!

Using `Arrays.sort()`

`Arrays` is the name of a library (static class) built-in to Java. You have to import it to use it.

```
import java.util.Arrays;

public static void main(String[] args) {
    String[] arr = {"cherry", "apple", "banana"};
    Arrays.sort(arr); // sorts in place
    System.out.println(Arrays.toString(arr)); // [apple, banana, cherry]
}
```

`Arrays.sort` works on any array of objects that implement `Comparable`.

Sorting Lists

To sort a `List`, you need to use `Collections.sort()` instead of `Arrays.sort()`.

```
import java.util.Collections;

public static void main(String[] args) {
    List<String> l = new ArrayList<String>();
    l.add("cherry");
    l.add("apple");
    l.add("banana");
    Collections.sort(l); // sorts in place
    System.out.println(l); // [apple, banana, cherry]
}
```

`equals()`

`==` is only useful for determining *referential* equality between objects

- Do these two references point to the same object?

`equals()` is a method that compares two objects for *structural* equality.

- Do these two objects represent the same thing?

The Duration Class

credit to this course website for the example idea

```
public class Duration {
    private int minutes;
    private int seconds;

    public Duration(int minutes, int seconds) {
        this.minutes = minutes;
        this.seconds = seconds;
    }
    public boolean equals(Duration other) { ... }
}
```

Poll

Which of the following `Duration` objects is not *referentially* equal to `d1`? (i.e. not equal using `==`)?

```
Duration d1 = new Duration(1, 30);
Duration d2 = new Duration(1, 30);
Duration d3 = d1;
Duration d4 = d2;
```

Poll

Which of the following `Duration` objects should be considered *structurally* equal to `d1`?

```
Duration d1 = new Duration(1, 30);
Duration d2 = new Duration(0, 90);
Duration d3 = d1;
Duration d4 = d2;
Duration d5 = new Duration(1, 20);
```

Drafting an Equals Method

How can we write the `equals()` method for the `Duration` class?

We want:

- the object to always be structurally equal to itself
    - (that is, referential equality should imply structural equality)
- a null object should never be structurally equal to anything
- two Durations should be equal if they represent the same amount of time

Poll

Is this good enough?

```
public boolean equals(Duration other) {
    if (other == null) {
        return false;
    }
    return this.minutes == other.minutes && this.seconds == other.seconds;
}
```

Activity

Write a better `equals()` method for the `Duration` class.

Solution

```
public int lengthInSeconds() {
    return this.minutes * 60 + this.seconds;
}

public boolean equals(Duration other) {
    if (other == null) { // handle null case
        return false;
    }
    if (other == this) { // handle referential equality
        return true;
    }
    return this.lengthInSeconds() == other.lengthInSeconds();
}
```

Discussion

Can you think of a place that it might be useful to define an `equals()` method in your Snake project?

Relationship Between `equals()` & `compareTo()`

Ideally, when `compareTo()` returns `0`, `equals()` should return `true`, and vice versa. It's possible, though, that you might want to compare objects in a way that's more or less "detailed" than testing for equality.

- In practice: defining custom comparators allows for flexible behavior, but that's beyond the scope of this class
- For us: it's OK if `compareTo()` might say that some objects are "equal" even if `equals()` would return `false`.

```
public class Book implements Comparable<Book> {
    private String title;
    private String author;
    private int ISBN;

    public int compareTo(Book other) { // sort by author name, then title
        if (author.equals(other.getAuthor)) {
            return title.compareTo(other.getTitle());
        }
        return author.compareTo(other.getAuthor());
    }

    // use the guaranteed unique ISBN for actual equality
    public boolean equals(Book other) {
        return this.ISBN == other.ISBN;
    }
}
```

This is a reasonable implementation of both methods, although they imply two different notions of equality.

Activity

The object `Roster` maintains an array of unique `Student` objects. Write a method that prints the array of students in sorted order.

- Note: don't print `null` Student references if they're present
- Keep in mind that `Student` implements `Comparable<Student>`, so you should offload the hard word of sorting to `Arrays.sort(...)`
- `Student` objects can be printed directly since they have a `toString()` method implemented.