COM2104: Advanced Programming

LECTURE 6: FURTHER KNOWLEDGE ABOUT ARRAYLIST

Objectives

- Understand advanced sorting
- Know how to define your own rule for sorting objects
- Know how to apply lambda for functional interface and arraylist
- Know how to get min/max of an arraylist with numbers

Advanced Sorting

Java Advanced Sorting

- In the previous lecture, you learned how to sort lists alphabetically and numerically, but what if the list has objects in it?
- To sort objects you need to specify a rule that decides how objects should be sorted.
  - For example, if you have a list of cars you might want to sort them by year, the rule could be that cars with an earlier year go first.
- The Comparator and Comparable interfaces allow you to specify what rule is used to sort objects.

Comparators

- An object that implements the Comparator interface is called a comparator.
- The Comparator interface allows you to create a class with a compare() method that compares two objects to decide which one should go first in a list

Comparators

- The compare() method should return a number which is:
  - Negative if the **first** object should go first in a list.
  - Positive if the **second** object should go first in a list.
  - Zero if the order ***does not matter***.

One example about Comparator

- A class that implements the Comparator interface might look something like this:

```
// Sort Car objects by year
class SortByYear implements Comparator {
  public int compare(Object obj1, Object obj2) {
    // Make sure that the objects are Car objects
    Car a = (Car) obj1;
    Car b = (Car) obj2;

    // Compare the objects
    if (a.year < b.year) return -1; // The first car has a smaller year
    if (a.year > b.year) return 1;  // The first car has a larger year
    return 0; // Both cars have the same year
  }
}
```

One basic Knowledge you should know

- For object obj1, we could use a class name to specify:
  - Like Car obj1 means obj1 is an instance of class Car.
    - Car a = (Car) obj1;
  - Like Integer obj1 means obj1 is an integer.
    - Integer a = (Integer) obj1;
  - Like Double obj1 means obj1 is a double.
    - Double a = (Double) obj1;

A complete example using a comparator

```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;

// Define a Car class
class Car {
  public String brand;
  public String model;
  public int year;

  public Car(String b, String m, int y) {
    brand = b;
    model = m;
    year = y;
  }
}

// Create a comparator
class SortByYear implements Comparator {
  public int compare(Object obj1, Object obj2) {
    // Make sure that the objects are Car objects
    Car a = (Car) obj1;
    Car b = (Car) obj2;

    // Compare the year of both objects
    if (a.year < b.year) return -1; // The first car has a smaller year
    if (a.year > b.year) return 1;  // The first car has a larger year
    return 0; // Both cars have the same year
  }
}
```

```java
public class Main {
  public static void main(String[] args) {
    // Create a list of cars
    ArrayList<Car> myCars = new ArrayList<Car>();
    myCars.add(new Car("BMW", "X5", 1999));
    myCars.add(new Car("Honda", "Accord", 2006));
    myCars.add(new Car("Ford", "Mustang", 1970));

    // Use a comparator to sort the cars
    Comparator myComparator = new SortByYear();
    Collections.sort(myCars, myComparator);

    // Display the cars
    for (Car c : myCars) {
      System.out.println(c.brand + " " + c.model + " " + c.year);
    }
  }
}
```

```
Ford Mustang 1970
BMW X5 1999
Honda Accord 2006
```

The explanation for the above example

```java
// Sort Car objects by year
class SortByYear implements Comparator {
  public int compare(Object obj1, Object obj2) {
    // Make sure that the objects are Car objects
    Car a = (Car) obj1;
    Car b = (Car) obj2;

    // Compare the objects
    if (a.year < b.year) return -1; // The first car has a smaller year
    if (a.year > b.year) return 1;  // The first car has a larger year
    return 0; // Both cars have the same year
  }
}
```

```java
// Use a comparator to sort the cars
Comparator myComparator = new SortByYear();
Collections.sort(myCars, myComparator);
```

We define our own rule for sorting objects

When using our own rule, we need to create one instance about defined comparator first, then pass it to the sorting method.

Special Sorting Rules

- Comparators can also be used to make special sorting rules for strings and numbers.
- For example, we could use a comparator to list all of the even numbers before the odd ones. (Next slide)

For Example

```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;

class SortEvenFirst implements Comparator {
  public int compare(Object obj1, Object obj2) {
    // Make sure the objects are integers
    Integer a = (Integer)obj1;
    Integer b = (Integer)obj2;

    // Check each number to see if it is even
    // A number is even if the remainder when dividing by 2 is 0
    boolean aIsEven = (a % 2) == 0;
    boolean bIsEven = (b % 2) == 0;

    if (aIsEven == bIsEven) {

      // If both numbers are even or both are odd then use normal sorting rules
      if (a < b) return -1;
      if (a > b) return 1;
      return 0;

    } else {

      // If a is even then it goes first, otherwise b goes first
      if (aIsEven) {
        return -1;
      } else {
        return 1;
      }
    }
  }
}
```

```java
public class Main {
  public static void main(String[] args) {
    ArrayList<Integer> myNumbers = new ArrayList<Integer>();
    myNumbers.add(33);
    myNumbers.add(15);
    myNumbers.add(20);
    myNumbers.add(34);
    myNumbers.add(8);
    myNumbers.add(12);

    Comparator myComparator = new SortEvenFirst();
    Collections.sort(myNumbers, myComparator);

    for (int i : myNumbers) {
      System.out.println(i);
    }
  }
}
```

```
8
12
20
34
15
33
```

A further explanation for the above example

```
public int compare(Object obj1, Object obj2) {
  // Make sure the objects are integers
  Integer a = (Integer)obj1;
  Integer b = (Integer)obj2;

  // Check each number to see if it is even
  // A number is even if the remainder when dividing by 2 is 0
  boolean aIsEven = (a % 2) == 0;
  boolean bIsEven = (b % 2) == 0;

  if (aIsEven == bIsEven) {

    // If both numbers are even or both are odd then use normal sorting rules
    if (a < b) return -1;
    if (a > b) return 1;
    return 0;

  } else {

    // If a is even then it goes first, otherwise b goes first
    if (aIsEven) {
      return -1;
    } else {
      return 1;
    }
  }
}
```

We first judge whether a or b are both even or odd. Then, using if-else statements to realize even number going first.

Comparable Interface

- The Comparable interface allows an object to specify its own sorting rule with a compareTo() method.
- The compareTo() method takes an object as an argument and compares the comparable with the argument to decide which one should go first in a list

Comparable Interface

- Like the comparator, the compareTo() method returns a number which is:
  - Negative if the comparable should go first in a list.
  - Positive if the other object should go first in a list.
  - Zero if the order does not matter.

One example about using Comparable interface

```
class Car implements Comparable {
  public String brand;
  public String model;
  public int year;

  // Decide how this object compares to other objects
  public int compareTo(Object obj) {
    Car other = (Car)obj;
    if(year < other.year) return -1; // This object is smaller than the other one
    if(year > other.year) return 1;  // This object is larger than the other one
    return 0; // Both objects are the same
  }
}
```

The complete example of the above one

```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;

// Define a Car class which is comparable
class Car implements Comparable {
  public String brand;
  public String model;
  public int year;

  public Car(String b, String m, int y) {
    brand = b;
    model = m;
    year = y;
  }

  // Decide how this object compares to other objects
  public int compareTo(Object obj) {
    Car other = (Car)obj;
    if(year < other.year) return -1; // This object is smaller than the other one
    if(year > other.year) return 1;  // This object is larger than the other one
    return 0; // Both objects are the same
  }
}
```

```java
public class Main {
  public static void main(String[] args) {
    // Create a list of cars
    ArrayList<Car> myCars = new ArrayList<Car>();
    myCars.add(new Car("BMW", "X5", 1999));
    myCars.add(new Car("Honda", "Accord", 2006));
    myCars.add(new Car("Ford", "Mustang", 1970));

    // Sort the cars
    Collections.sort(myCars);

    // Display the cars
    for (Car c : myCars) {
      System.out.println(c.brand + " " + c.model + " " + c.year);
    }
  }
}
```

```
Ford Mustang 1970
BMW X5 1999
Honda Accord 2006
```

A further explanation for the above example

Here, the class implements the Comparable interface and overwrites the compareTo() method.

```
class Car implements Comparable {
  public String brand;
  public String model;
  public int year;

  // Decide how this object compares to other objects
  public int compareTo(Object obj) {
    Car other = (Car)obj;
    if(year < other.year) return -1; // This object is smaller than the other one
    if(year > other.year) return 1;  // This object is larger than the other one
    return 0; // Both objects are the same
  }
}
```

Comparator vs. Comparable

- A comparator is an object with one method that is used to compare two different objects.
- A comparable is an object which can compare itself with other objects.
- It is easier to use the Comparable interface when possible, but the Comparator interface is more powerful because it allows you to sort any kind of objects.

Lambda

AddALL and forEach of arraylist

1. Collections.addAll() method could add multiple elements together to an arraylist.
2. forEach method could allow us to iterate over each element in the arraylist.

For Example:

```
ArrayList<string> list = new ArrayList<>();
Collections.addAll(list, "Hello", "How", "дела?");


list.forEach( (s) -> System.out.println(s) );
```

Java Lambda Expressions

- **Lambda expressions in Java**, represent instances of functional interfaces (interfaces with a single abstract method).
- They provide a concise way to express instances of single-method interfaces using a block of code.

Functional Interface

- A **functional interface** **in Java** is an interface that contains only one **abstract** method. Functional interfaces can have multiple **default** or **static** methods, but **only one abstract method.**

```
// Define a functional interface
@FunctionalInterface

interface Square {
    int calculate(int x);
}
```

Syntax for lambda expression

```
(argument list) -> { body of the expression }
```

Components:

- **Argument List:** Parameters for the lambda expression
- **Arrow Token (->):** Separates the parameter list and the body
- **Body:** Logic to be executed.

Lambda Expression Parameters

- There are three Lambda Expression Parameters are mentioned below:
  - Zero Parameter
  - Single Parameter
  - Multiple Parameters

Lambda Expression with Zero parameter

```
() -> System.out.println("Zero parameter lambda");
```

Zero parameter

Body of expression

Example

Functional Interface

Lambda Expression

```
// Java program to demonstrates Lambda expression with
zero parameter

@FunctionalInterface
interface ZeroParameter {
    void display();
}

public class Geeks {
    public static void main(String[] args)
    {
        // Lambda expression with zero parameters
        ZeroParameter zeroParamLambda = ()
            -> System.out.println(
                "This is a zero-parameter lambda
expression!");

        // Invoke the method
        zeroParamLambda.display();
    }
}
```

Output

```
This is a zero-parameter lambda expression!
```

Lambda Expression with Single parameter

```
(p) -> System.out.println("One parameter: " + p);
```

Single parameter

Body of expression

Example

```java
import java.util.ArrayList;
class Test {
    public static void main(String args[])
    {
        // Creating an ArrayList with elements
        // {1, 2, 3, 4}
        ArrayList<Integer> arrL = new ArrayList<Integer>();
        arrL.add(1);
        arrL.add(2);
        arrL.add(3);
        arrL.add(4);

        // Using lambda expression to print all elements
        // of arrL

        System.out.println("Elements of the ArrayList : ");
        arrL.forEach(n -> System.out.println(n));

        // Using lambda expression to print even elements
        // of arrL
        System.out.println(
            "Even elements of the ArrayList : ");
        arrL.forEach(n -> {
            if (n % 2 == 0)
                System.out.println(n);
        });
    }
}
```

Lambda Expression

Lambda Expression

Output

```
Elements of the ArrayList :
1
2
3
4
Even elements of the ArrayList :
2
4
```

A further explanation for the above example

```
arrL.forEach(n -> {
        if (n % 2 == 0)
            System.out.println(n);
});
```

For the right side of arrow, we could write an if statement.

Lambda Expression with Multiple parameters

```
(p1, p2) -> System.out.println("Multiple parameters: " + p1 + ", " + p2);
```

Multiple parameter

Body of expression

Example

Functional Interface

Lambda Expression

```
@FunctionalInterface
interface Functional {
    int operation(int a, int b);
}

public class Test {

    public static void main(String[] args) {
        // Using lambda expressions to define the
operations
        Functional add = (a, b) -> a + b;
        Functional multiply = (a, b) -> a * b;

        // Using the operations
        System.out.println(add.operation(6, 3));  //
Output: 9
        System.out.println(multiply.operation(4, 5));
// Output: 20
    }
}
```

Output

```
9
20
```

Min/Max of an arraylist

Method 1

```java
import java.util.*;
public class Max {
    public static void main(String args[])
    {
        // initializing the ArrayList elements
        ArrayList<Integer> arr = new ArrayList<>();
        arr.add(10);
        arr.add(20);
        arr.add(8);
        arr.add(32);
        arr.add(21);
        arr.add(31);
        int min = arr.get(0);
        int max = arr.get(0);
        // store the length of the ArrayList in variable n
        int n = arr.size();
        // loop to find minimum from ArrayList
        for (int i = 1; i < n; i++) {
            if (arr.get(i) < min) {
                min = arr.get(i);
            }
        }
        // loop to find maximum from ArrayList
        for (int i = 1; i < n; i++) {
            if (arr.get(i) > max) {
                max = arr.get(i);
            }
        }
        // The result will be printed
        System.out.println("Maximum is : " + max);
        System.out.println("Minimum is : " + min);
    }
}
```

Min

Max

Output

```
Maximum is : 32
Minimum is : 8
```

Method 2

Max:
Collections.max(arraylist)

Min:
Collections.min(arraylist)

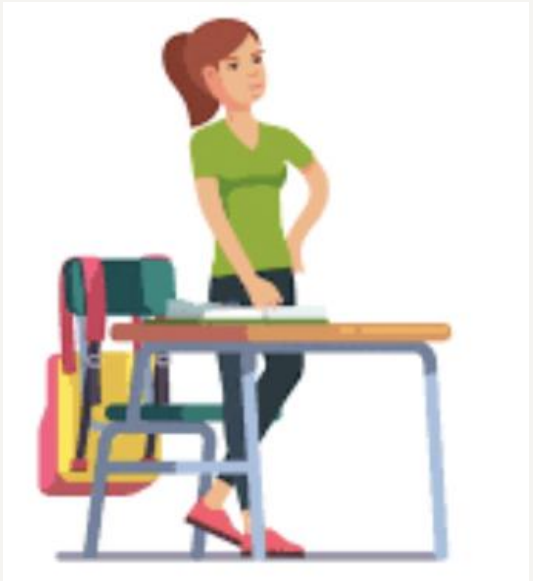
```java
import java.util.ArrayList;
import java.util.Collections;
public class MinMax {
    public static void main(String args[])
    {
        // Creating arraylist
        ArrayList<Integer> arr = new ArrayList<Integer>();

        // Adding object in arraylist
        arr.add(10);
        arr.add(20);
        arr.add(5);
        arr.add(8);

        // Finding the size of ArrayList
        int n = arr.size();

        // printing the ArrayList elements
        System.out.println("ArrayList elements are :");

        for (int i = 0; i < n; i++) {
            System.out.print(arr.get(i) + " ");
        }

        System.out.println();

        // Finding the minimum and maximum from the
        // arraylist using min and max method of collection
        // class

        int max = Collections.max(arr);
        System.out.println("Maximum is : " + max);

        int min = Collections.min(arr);
        System.out.println("Minimum is : " + min);
    }
}
```

Output

```
Array elements are :
10 20 5 8
Maximum is : 20
Minimum is : 5
```

Method 3

- You could sort the arraylist first.
- Applying get() and size() method.
- After sorting, the first element is the minimal value. The last element is the maximal value.




End