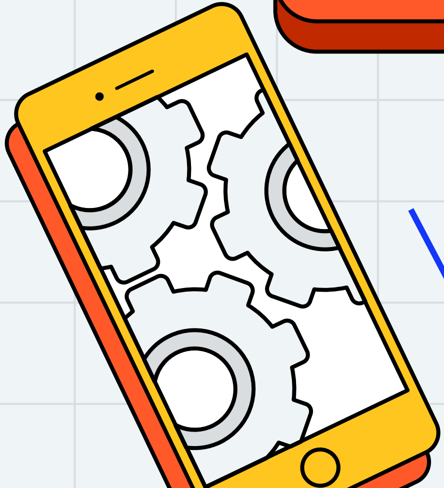
Application Programming



Hend Alkittawi

Java Comparison

Comparing objects in Java
Collections

INTRODUCTION

- Related to the discussion on Java generics and collections is the discussion on the following interfaces
  - Iterator, Iterable, Comparable, Comparator
- The Java API has a consistent approach to iterators that are implemented by nearly all collections in the class Library.
- Iterators are implemented in the Java API using two primary interfaces:
  - **Iterator**: used to define an object that can be used as an iterator.
  - **Iterable**: used to define a collection from which an iterator can be extracted.
- The **Comparable** and **Comparator** interfaces in Java facilitate comparisons between objects

COMPARING OBJECTS

- Classes that implement the Comparable and Comparator interfaces must contain certain key methods for comparing objects created by that class
  - For example, the String class implements Comparable, so sorting Strings in an array alphabetically is easy

```
String[] fruits = new String[] {"Pineapple", "Apple", "Orange", "Banana"};
Arrays.sort(fruits);
```

THE COMPARABLE INTERFACE

- The **Comparable** interface contains only one method: **compareTo()** which takes an object as a parameter and returns an integer
- The purpose of this interface is to provide a common mechanism for comparing one object to another

```
ClassName obj1 = new ClassName();
ClassName obj2 = new ClassName();
int result     = obj1.compareTo(obj2);
```

- The integer that is returned from the compareTo() method should be
  - negative if obj1 < obj2
  - positive if obj1 > obj2
  - zero   if obj1 = obj2

THE COMPARABLE INTERFACE

- If an object is **Comparable**, we can sort an array of it

```
Book book1 = new Book(...);
Book book2 = new Book(...);
```

  - For an **array** of Comparable objects, use   Arrays.sort()
  - The Arrays class provides the sorting logic for Comparable types
  - Arrays.sort()takes an array of objects which implement the Comparable interface

```
Book[] books = new Book[2];
books[0] = book1;
books[1] = book2;
Arrays.sort( books );                          // Comparable
```

THE COMPARABLE INTERFACE

- If an object is **Comparable**, we can sort a collection of it

```
Book book1 = new Book(...);
Book book2 = new Book(...);
```

- For an **arraylist** of Comparable objects use the sort() method from Collections

```
ArrayList<Book> bookList = new ArrayList<Book>();
bookList.add( book1 );
bookList.add( book2 );
Collections.sort( bookList );
```

```java
// some code is omitted, check code for
project in Canvas

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

    public Book(String title, String author) {
       this.title = title;
       this.author = author;
    }

    @Override
    public String toString() {
       return "Book [title=" + title + ", author="
                                + author + "]";
    }
    @Override
    public int compareTo(Book other) {
       return
             this.getAuthor().compareTo(

other.getAuthor());
    }
}
```

```java
public class ComparableTest {

    public static void main(String[] args) {
        Book book1 = new Book("Java The Complete Guide", "Pat Alfonso");
        Book book2 = new Book("Java for Begginers",      "Hamza Ryan");
        Book book3 = new Book("Java for Begginers",      "Daisy Mack");
        Book book4 = new Book("Java All in One",         "Carolina Minato");
        Book book5 = new Book("Java All in One",         "Carolina Aidan");

        ArrayList<Book> bookArrayList = new ArrayList<Book>();
        bookArrayList.addAll(Arrays.asList(book1, book2, book3, book4, book5));
        System.out.println("******* Unsorted Collection *******");
        System.out.println(bookArrayList);

        Collections.sort(bookArrayList);
        System.out.println("******* Sorted Collection *******");
        System.out.println(bookArrayList);

        Book[] bookArray = {book1, book2, book3, book4, book5};

        System.out.println("******* Unsorted Array *******");
        for(Book book : bookArray)
                System.out.println(book);

        Arrays.sort(bookArray);

        System.out.println("******* Sorted Array *******");
        for(Book book : bookArray)
                System.out.println(book);
        }
}
```

THE COMPARATOR INTERFACE

- The **Comparator** interface contains the **compare()** method which takes two objects as a parameter and returns an integer
- If an object is **Comparator**, we can sort an array of it

```
Book book1 = new Book(...);
Book book2 = new Book(...);
```

  - For an **array** of Comparable objects, use Arrays.sort()
  - The Arrays class provides the sorting logic for Comparable types
  - Arrays.sort()takes an array of objects which implement the Comparable interface

```
Book[] books = new Book[2];
books[0] = book1;
books[1] = book2;
Arrays.sort( books );                           // Comparable
Arrays.sort( books, Book.bookComparator ); // Comparator
```

THE COMPARATOR INTERFACE

- For an arrayList, you can use the use the sort() method from ArrayList

```
bookList.sort( Book.bookComparator );
```

where BookComparator is defined in the Book class as follows

```
public static Comparator<Book> bookComparator
    = new Comparator<Book>() {
        public int compare(Book book1, Book book2) {
            return book1.getTitle().compareTo(book2.getTitle());
        }
      };
// public static Comparator<Book> bookComparator = new Comparator<Book>();
```

- Inner classes are classes defined within another class.
- An anonymous inner class is a class without a name, for which only one object is created.

Implementing The Comparator Interface

```java
// some code is omitted, check code
for project in Canvas

public class Book {

    public static MyComparator bookComparator;

    private String title;
    private String author;

    public Book(String title, String author){
       this.title = title;
       this.author = author;
       bookComparator = new MyComparator();
    }

    @Override
    public String toString() {
       return "Book [title=" + title + ",
                 author=" + author + "]";
    }
}
```

```java
public class MyComparator implements Comparator<Book>{
    @Override
    public int compare(Book book1, Book book2){
        return book1.getAuthor().compareTo(book2.getAuthor());
    }
}
```

```java
// some code is omitted, check code for project in Canvas
public class ComparatorTest {

    public static void main(String[] args){
        Book book1 = new Book("Java The Complete Guide", "Pat Alfonso");
        Book book2 = new Book("Java for Begginers",      "Hamza Ryan");
        Book book3 = new Book("Java for Begginers",      "Daisy Mack");
        Book book4 = new Book("Java All in One",         "Carolina Minato");
        Book book5 = new Book("Java All in One",         "Carolina Aidan");

        ArrayList<Book> bookArrayList = new ArrayList<Book>();
        bookArrayList.addAll(Arrays.asList(book1, book2, book3, book4, book5));

        Collections.sort(bookArrayList, Book.bookComparator);

        // OR
        bookArrayList.sort(Book.bookComparator);
}
```

Implementing The Comparator Interface As An Inner Class

```java
// some code is omitted, check code for project
in Canvas
public class Book {
     public static MyComparator bookComparator;
     private String title;
     private String author;

     public Book(String title, String author) {
        this.title = title;
        this.author = author;
        bookComparator = new MyComparator();
     }

     @Override
     public String toString() {
        return "Book [title=" + title
        + ", author=" + author + "]";
     }


     private class MyComparator implements Comparator<Book> {
        @Override
        public int compare(Book book1, Book book2) {
                return book1.getAuthor().
                                 compareTo(book2.getAuthor());
        }
    }
}
```

```java
// some code is omitted, check code for project in Canvas
public class ComparatorTest {

     public static void main(String[] args){
         Book book1 = new Book("Java The Complete Guide", "Pat Alfonso");
         Book book2 = new Book("Java for Begginers",      "Hamza Ryan");
         Book book3 = new Book("Java for Begginers",      "Daisy Mack");
         Book book4 = new Book("Java All in One",         "Carolina Minato");
         Book book5 = new Book("Java All in One",         "Carolina Aidan");

         ArrayList<Book> bookArrayList = new ArrayList<Book>();
         bookArrayList.addAll(Arrays.asList(book1, book2, book3, book4, book5));

         Collections.sort(bookArrayList, Book.bookComparator);

         // OR
         bookArrayList.sort(Book.bookComparator);
}
```

Implementing The Comparator Interface As An Anonymous Inner Class

```java
// some code is omitted, check code for project
in Canvas
public class Book {
     public static MyComparator bookComparator;
     private String title;
     private String author;

     public Book(String title, String author) {
        this.title = title;
        this.author = author;
        bookComparator = new MyComparator();
     }

     @Override
     public String toString() {
        return "Book [title=" + title
        + ", author=" + author + "]";
     }


     private class MyComparator implements Comparator<Book> {
        @Override
        public int compare(Book book1, Book book2) {
               return book1.getAuthor().
                                compareTo(book2.getAuthor());
        }
    }
}
```

```java
// some code is omitted, check code for project in Canvas

public class Book {
       public static Comparator<Book> bookComparator = new Comparator<Book>() {
               public int compare(Book book1, Book book2) {
                       return book1.getAuthor().compareTo(book2.getAuthor());
               }
       };

       private String title;
       private String author;

       public Book(String title, String author) {
               this.title = title;
               this.author = author;
       }
}
```

THANK YOU!

DO YOU HAVE ANY QUESTIONS?

@ hend.alkittawi@utsa.edu

By Appointment

Online