Java Practical with Collections

Objectives

The objectives of this practical session are to:

- Use Collections instead of arrays in the Employee application
- Create random test data
- Sort the collections
- Use Inner Class syntax in Java to implement custom sorting objects

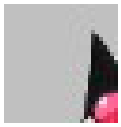
Overview

This practical consists of two parts. In the first part you will refactor your code to use the Collection API, with improved functionality over arrays. You then create random test data, which allows you to create a large number of employees and managers and then test the performance of your application to benchmark different choices of collection.

The second part uses `Comparator`s to sort collection elements. A suitable implementation of these data utilities might involve inner classes for coupling control.

Practical

Using Collections and Creating Random Data



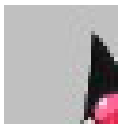
1. Create a new project in IntelliJ. Copy the `Employee` Java file from the `LoadsOfEmployees` folder in the exercises ZIP.
2. Note that there is a static method which returns a list of many Employees, with some real and random data.
3. Create a class `EmployeeTest`, with a `main()` method, and print out the details of the `Employee`s in a formatted way, using the enhanced 'for-each' loop `for( : )` syntax when iterating around the collection. Test your code.
4. Using another loop, create a list of all the `Employee`s from the SALES department. Print out their details. Do the same thing for employees under the age of 40.
5. Try out the methods `addAll()` and `removeAll()` to see how they work with your various lists.
6. Create a `HashSet`, using the constructor:

```
Set<Employee> s = new HashSet<Employee>(mylist);
```

7. Try step 5, using these sets, and see that they won't contain duplicates.

Practical

Part 2. Sorting and Comparator Classes



Objectives

The objective of this part is to define `Comparator` objects to sort your `Employee` objects in a `List`.

Overview

A `Comparator` object is an object that contains logic to sort other objects. You implement the `java.util.Comparator` generic class, and provide the method `compare()`, which takes two parameter and returns an `int`. The `int` must be a positive number or a negative number or zero, depending on whether the passed objects are in ascending order, descending order, or equal (as in the `equals()` method).

1. In the project you created earlier locate your `Employee` class. Provide a new class `EmployeeNameComparator`, which implements `java.util.Comparator`.

   ```
   public class EmployeeNameComparator
                    implements Comparator<Employee> {...

   }
   ```

2. Note the generic type parameter, which means this `Comparator` compares two `Employee` objects. Compile your code, and obey instructions to add the unimplemented methods.

3. Thus, provide the method:

   ```
   public int compare(Employee e1, Employee e2) {
         ...
   }
   ```

   This needs to return an `int`, so use the `String`'s `compareTo()` instance method on the `Employee`'s name, which sorts `String`s alphabetically, and return precisely this value.

4. In your main program, create yourself a `List` of `Employee` objects (perhaps some `Managers`), and print it out using a loop. Create an instance of your `Comparator` class. Now sort the `List` using the static method:

   ```
   Collections.sort(yourList, yourComparator);
   ```

5. Print out the contents of the `List` once more, and test to see if your sorting has worked. Run and test your application.

6. Write another `Comparator` class, similar to the other one, which sorts `Employees` by age.

7. In your `main()` method, provide another sort, via the `Employee`'s age.

8. If two `Employee`'s have the same age (quite likely), then use a secondary sort criterion of their name. What is the best way to achieve this?
9. Test your code.