Imperial College London – Department of Computing

MSc in Computing Science

536: Introduction to Java
Tutorial 2 — Collections

In this exercise you will create a Java class to represent people, make a collection of people objects, and use the features of the Java Collections framework to sort the people in various ways.

1 Aims

After completing this tutorial you will be able to:

- Use the Java API documentation
- Use Eclipse's wizards and other features to speed up your programming
- Use Java interfaces to implement multiple inheritance
- Use the `List<T>` type from the Java Collections framework
- Implement sorting of collections of objects on multiple attributes

This tutorial aims to guide you through your first steps to using the Java API documentation. The single most important website to learn Java the language and the Java API is http://docs.oracle.com/javase/tutorial/. The API Documentation can be found under Other Resources, on the right side of the web page.

2 What To Do

2.1 Create an Eclipse Project

- Create a new Java Project in Eclipse. Name it whatever you want. Either accept the default location or choose your own.
- Eclipse will create a `/src` folder for your code, a `/bin` folder for compiled `.class` files, and two project configuration files: `.classpath` and `.project`.

2.2 Create a Program Class

First, create a new Java class as the starting point for your program. The class should be called `Program`, in package `people` and have a single method `main`.

- Select the `src` folder in the Eclipse window and click on **File** > **New** > **Class**, or click the New Class button on the toolbar.
- Fill in the class name (`Program`), the package name (`people`) and check the box to create a main method.
- Once your new class is created, add the following line to its `main` method:

  ```
  Person ann = new Person("Ann", "Aarvark", 22);
  ```

  You will see that Eclipse shows an error because there is no such class as `Person`. Do not worry, this is easily fixed!

2.3 Create a Person Class

Eclipse has a very helpful "Quick Fix" feature that suggests ways to fix problems in your code. This feature can be used to help write your program faster. Let's use this to create a `Person` class.

- Position the cursor somewhere on the line in `Program.main()` with the error, and press **Ctrl-1**.
- You will see a list of Quick Fix options. Choose "Create class 'Person' ".
- The new class form is already filled in with the correct package and class name, so just click Finish to create the class.
- Now press **Alt-LeftArrow** to go back to where you just came from. You will see there is still an error in `Program.main` because the correct `Person` constructor does not yet exist.
- Call up the Quick Fix options again. You will see new options including "Create constructor Person(String, String, int)". Choose this option.

The three member variables of `Person` are a `String surname`, a `String givenNames` and an `int age`. These can also be created using Quick Fix.

- Add a line to the `Person` constructor to assign the value of the first `String` parameter to `givenNames`. (Do not declare `givenNames` as a local variable, just assign to the name.) Now use Quick Fix to create the declaration of the field `givenNames`.
- Repeat this procedure for the other two fields.
- Finally, go to the Source menu and click "Generate Getters and Setters". Click the arrows next to each field to expand the choices, check each of the "get" options and click OK.
- Save both files.

2.4 Run the Program

Your program does very little at the moment, but it is always worth checking that there are no errors.

- Select `Person.java` and click on the `Run` button on the Eclipse toolbar.
- You should see an empty console window in Eclipse and a message that `Program` terminated. If there is no such message, then you will have to click the Terminate button (red square) in the toolbar of the console window to stop the program.

2.5 Override `toString()`

`Person` has a `toString()` method that it inherits from `Object`, but it is not very useful.

- Add the following line to `Program.main()`:

  ```
  System.out.println(ann);
  ```

  This line will call `Person.toString()` to get a string representation of `ann` to print.
- Run the program. As you should see, `Object.toString()` just returns the memory address of the object.
- Override `toString()` to provide a better string representation of a `Person`. Follow the format: "`givenNames surname (age)`". You can use the **Override/Implement Methods** wizard on the **Source** menu to avoid too much strenuous typing.

2.6 Make `Person` Comparable

Now, look up the `Comparable<T>` interface in the `java.lang` (i.e. default) package in the API Documentation. As described in the documentation, Lists (and arrays) of objects that implement `Comparable` can be sorted automatically by classes in the Java Collections framework. This is because a class that implements `Comparable` guarantees to define its single method: `compareTo()`, which should define a *natural ordering* for elements of the class. Study the description of the `compareTo()` method carefully. Your `compareTo()` method needs to follow the given requirements.

- Change the heading of the `Person` class to declare that it implements `Comparable<Person>`.
- Use the Quick Fix options to create the `compareTo()` method.
- Your `compareTo()` method should compare the current object to the other `Person` by `surname` in lexical order, then `givenNames` in lexical order, then `age` in ascending order. **Hint:** the class `String` is also `Comparable`.
- Add some lines to `Program.main()` to test `compareTo()`.

2.7 Override `equals()`

The documentation tells you that the natural ordering defined by `compareTo()` should be consistent with `equals()`. So, given two non-null `Person` objects `p1` and `p2`, `p1.compareTo(p2) == 0` should be true if and only if `p1.equals(p2)` is true.

- Override `equals()` in `Person` so that this assertion is satisfied.
- According to the API documentation for `Object.equals()` "For any non-null reference value `x`, `x.equals(null)` should return `false`", so make sure you check for that case.
- The argument of `equals()` is of type `Object`, so you also need to check that it is a `Person` with `if (o instanceof Person)` and return `false` if it is not.
- Assuming these initial checks pass, you will need to type cast the argument to a `Person`, and then use the `compareTo()` method you have just written to test for equality.

2.8 Make a `Person` List

Now create a `Person` list in your main program. `List` is a type (in fact an interface) in the Collections framework. It is a generic type, so what you need is a `List<Person>`.

- Declare a variable in `Program.main` of type `List<Person>`.
- Use Quick Fix to add the required import statement to the file. You will see there is more than one `List` type available. You want the one in the `java.util` package.
- Now create a new object and assign it your `List` variable. Remember, `List` is an interface, so you cannot create objects of class `List`. The API documentation for `java.util.List` tells you which library classes implement `List`.

2.9 Populate the List

Add code to your main program that prompts the user for some people data, like so:

```
Enter a name: Justin Case
How old is Justin Case? 35
Enter a name: Jo King
How old is Jo King? 25
```

and create Person objects accordingly, putting them into the list.

- When dealing with a name, the part up to the last space is treated as the given names, and the part after this space is the surname. For example, "Jay Eff Kay" has a surname "Kay" and given names "Jay Eff". Do not worry about extra spaces for the moment. Hint: study the String class for useful methods (finding characters, getting substrings etc.) for this subtask. If the user input has no spaces, treat it as the end-of-list signal.
- When dealing with an age, you need to make sure that you get a non-negative integer from the user, e.g.

```
How old is Justin Case? 35q
That's not an integer.
How old is Justin Case? -17
That's negative.
How old is Justin Case? 35
```

2.10 Refactor Your Code

At this point you might find that the code to prompt the user and collect the input is rather long with deeply nested loops and conditionals. If so, you should extract some of the code into helper methods to avoid this.

- Try using Eclipse's extract method feature. Select the code you want to put into its own method and press **Shift-Alt-M**. Have a look at the **Refactor** menu to see what else Eclipse can do.

2.11 Sort and Print the List

If you have properly implemented `Person.toString()`, then you only need to pass the list object itself to `System.out.println()` to print its contents. The built-in code will go through all `Person` objects automatically.

- Print the unsorted list.
- Sort the list according to the natural ordering established in Section 2.6 above. Hint: the documentation of `Comparable<T>` has a hint for you. Print the sorted list.
- Sort the list by given names. Print the list again. Do not change the `compareTo()` method. Hint: the method you used to perform the previous sort is overloaded.