# Java 8 Features

## Overview

In this lab you'll refactor some Java applications that currently use "normal" classes, so that the code makes use of lambda expressions instead.

## Source folders

Student project: StudentJavaSE8

Solution project: SolutionJavaSE8

## Roadmap

There are 4 exercises in this lab, of which the last two exercises are "if time permits". Here is a brief summary of the tasks you will perform in each exercise; more detailed instructions follow later:

1. Implementing Runnable by using lambdas
2. Implementing Callable<T> by using lambdas
3. (If Time Permits) Implementing Comparator<T> by using lambdas
4. (If Time Permits) Handling GUI events by using lambdas

## Exercise 1: Implementing Runnable by using lambdas

In the *Student* project, open Exercise1\_Runnable.java and take a look at the existing code. Note the following points:

* Part 1 of the program creates an instance of a "normal" class that implements the Runnable interface. We pass our Runnable instance into the Thread constructor, and then start the new thread (don't worry too much about the mechanics of multithreading here). This causes our run() method to execute in the new thread.
* Part 2 of the program is similar, except that it creates the Runnable instance inline (i.e. within the call to the Thread constructor).

Refactor both parts of the program so that they use lambda expressions to represent the runnable code in each case, rather than implementing the Runnable interface manually as at present. The lambda expressions will represent the run() method, which doesn't take any parameters. This is the syntax for a lambda expression that doesn't take any parameters (note the empty parentheses):

() -> *your lambda expression*

## Exercise 2: Implementing Callable<T> by using lambdas

Open Exercise2\_Callable.java and take a look at the existing code. Note the following points:

* At the start of the code, we create a list of Callable<String> objects. Each object is an instance of a class that implements the Callable<String> interface, and provides a suitable call() method that returns a String result.
* Further on in the code, we create an ExecutorService to invoke the Callable<String> objects in separate threads (again, you don't need to worry about the threading details here). When each thread completes, we display its return value on the console.

Refactor the first part of the code so it creates lambda expressions rather than implementing the Callable<String> interface manually. The lambda expressions will represent the call() method, which doesn't take any parameters and returns a String result.

## Exercise 3 (If Time Permits): Implementing Comparator<T> by using lambdas

Open Exercise3\_Comparator.java and take a look at the existing code. At the start of the program, we create a list of Person objects. Each person has a name, age, and boolean flag indicating if he is Welsh ☺.

Then come the interesting parts:

* Part 1 of the code creates an instance of a class that implements Comparator<Person> (we've implemented the compare() method so that it compares Person objects by age). We then pass our Comparator<Person> instance into Collections.sort() to sort the list of persons by age.
* Part 2 of the code is similar, except that it creates the Comparator<Person> instance inline (i.e. within the call to the Collections.sort() method). Also note that this implementation of the compare() method compares Person objects by name rather than by age).

Refactor Parts 1 and 2 of the program so that they use lambda expressions to represent the comparison logic, rather than implementing the Comparator<Person> interface manually as at present.

## Exercise 4 (If Time Permits): Handling GUI events by using lambdas

Open Exercise4\_EventHandlers and take a look at the existing code. This is a simple Windows Forms application implemented using the Java Swing API. The application creates a window containing two buttons, and handles the click event for each button. Run the application and see how it works.

Refactor the code so that it uses lambda expressions to implement the event-handler logic, rather than implementing EventHandler<ActionEvent> manually as at present.