# Defining and Using Classes

## Overview

In this lab, you will write an application that defines and uses an Employee class.

## Source modules

Student module: StudentClasses

Solution module: SolutionClasses

## Roadmap

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

1. Defining a class and creating objects
2. Handling initialization
3. Adding static members
4. Additional suggestions

## Exercise 1: Defining a class and creating objects

Write an application that defines a simple Employee class and creates some instances.

Suggestions and requirements:

* The Employee class needs to hold the name and salary of the employee, and the date he/she joined the company.
* The class must honour the OO principle of encapsulation, so make sure the instance variables are private. Define public getter and setter methods if you need them.
* The class needs to allow an employee to have a pay raise, so define a payRaise() method that takes the amount of the pay raise and adds it to the employee's current salary.
* The class should also have a toString() method that returns a textual representation of the employee's info.
* Define a separate test class, where you can create some Employee objects and invoke methods upon them.

## Exercise 2: Handling initialization

Add constructors to the Employee class.

Suggestions and requirements:

* Add a constructor that initializes the employee's name and salary from passed-in values.
* Add another constructor that initializes the employee's name, and sets the salary to the minimum statutory salary (e.g. £7000 as a hard-coded figure, for now). Make use of constructor chaining here.
* Make sure you do NOT have a default constructor. Why not?
* Ensure the date the employee joined the company is always set to the current date/time (regardless of which constructor was called).
* Modify your test code so that it exercises each of the constructors.

## Exercise 3: Using statics

Refactor your Employee class to make appropriate use of static data and static methods.

Suggestions and requirements:

* In the Employee class, define a static field to hold the statutory minimum salary. Set it to 7000. Use this static field in the constructor that sets the employee's salary to the statutory minimum salary.
* Define static getter / setter methods to get / set the statutory minimum salary. Call these methods from your test code.

**Exercise 4 (If time permits): Additional suggestions**

In the Employee class, define a few overloaded versions of a payBonus() method.

* One version of the method can take no parameters and add a fixed percentage of the employee's salary (e.g. a 1% bonus).
* Another version of the method can take a double parameter that specifies the percentage of the bonus.
* Yet another version of the method can take three double parameters that specify the percentage of the bonus, along with a minimum and maximum salary (such that the bonus only applies if the employee's salary is within that range).

In the Employee class, define an instance variable to hold the employee's ID number. Also add a static variable named nextEmployeeID, which will be incremented each time a new employee is created. Hint: use an initialization block to assign the employee's ID from nextEmployeeID, and to increment nextEmployeeID ready for the next employee.