# Java Classes Practical

## Objectives

The objectives of this practical session are to:

- Define a new class
- Provide a class with constructors, methods and variables
- Create objects of that class
- Test the instance methods of a class

## Overview

This practical consists of two parts plus an optional part. In Part 1, you will define a simple class called `Employee` that will model an employee of a company. You will provide that class with just two instance variables for the employee's name and age, respectively. You will also provide a constructor to initialise these variables with some values. To test this class, you will also need to define another class called `EmployeeTest`. Here, you will create two objects of the `Employee` class and display the details of each employee. Since, the `Employee` class does not (as yet) have any instance methods, you will have to access its instance variables directly.

As you have learnt, it is bad practice to allow a user of your class to access your instance variables directly. So, in Part 2 of this practical, you will provide the `Employee` class with two instance methods to return the values of the employee's name and age, respectively. Then, you will modify the `EmployeeTest` class to make use of these new instance methods rather than accessing the instance variables directly. You will also add another instance method to the `Employee` class to allow a user of the class to increment the employee's age. The company isn't really interested in an employee once he or she reaches retirement age, but, to be kind, you are going to freeze their age once they reach retirement age. In other words, your method will not increment the employee's age beyond, say, 65.

## Practical

### Part 1. Defining a Class and Creating Objects of that Class

1. In IntelliJ, reopen the first project you created, the one with the "Employee Details:" print statement. Alternatively, create it afresh.
2. You should have two classes in your package, one called `EmployeeTest`, and another called `Employee`.
3. In the `Employee` class definition, declare two `private` instance variables: a `String` variable to hold the employee's last name and an `int` variable to hold the employee's age. Note that code assist is available through **CTRL+SPACE** (e.g. when you're halfway through typing 'String'!).

4. Use the top menu **Code** > **Generate** > **Getter and Setter**, to get the editor to create accessor methods for both variables.
5. Use the **Code** menu again to generate a constructor that has two arguments to initialise both of the employee's instance variables.
6. Save the project and fix any compiler errors before proceeding.
7. Return to the class definition for `EmployeeTest`. In the `main()` method, declare two variables of the `Employee` class. Then, use the `new` operator to create *two* objects of this class. Choose suitable names and ages for each employee.
8. Add code to display the details of each employee (using `System.out.println()`). Since you haven't provided any instance methods yet, you'll have to access the instance variables directly. Use the '`\t`' escape character to insert one or more tabs between the employee's name and age.
9. Save the project and fix any compiler errors. Execute the program `EmployeeTest`. Check that it displays the correct details of each employee.

## Part 2. Defining Instance Methods in a Class

1. In the `Employee` class definition, provide another public instance method, `incAge()`, that increments the employee's age. Do a "sanity check" before performing the operation, e.g. check that the employee's age is less than the retirement age of 65.
2. In the `EmployeeTest` class, after displaying the initial details of each employee, call the `incAge()` method of one of the employees and then redisplay the details of both employees.
3. Save and run, to check that your `incAge()` method performs correctly.
4. Enclose your call to `incAge()` in a `for` loop, and verify that the age doesn't increase beyond 65.
5. In the `Employee` class definition, declare another `private` instance variable to hold the employee's first name. Modify the `getName()` method so that it returns a string containing the employee's full name, i.e. first name concatenated with last name.
6. In `EmployeeTest`, modify the creation of your two `Employee` objects to use the new overloaded constructor that allows you to assign a full name to an employee. Execute the `EmployeeTest` class and check that the full name of each employee is displayed.
7. Add a class variable to the `Employee` class for employees' retirement age. This needs to be a class variable because there is only one retirement age and it's the same for all employees. You could initialise this variable to some reasonable value.

8. Modify the `incAge()` method so that it won't increment an employee's age beyond the value held in the retirement age class variable.
9. Define a class method to allow this retirement age to be changed.
10. In the `main()` method of the `EmployeeTest` class, call your class method to set the initial retirement age.
11. To test all this, perhaps you could have two loops, changing the retirement age between them, to see how the behaviour of one of your employees is different!

# Java Inheritance Practical

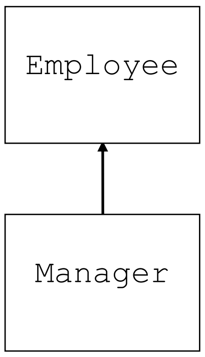
## Objectives

The objective of this session is to define a new class by extending an existing class. Then make use of some classes in the Java standard libraries.

## Overview

In this practical, you will *extend* your existing class called `Employee` to define a new class called `Manager`. Then, you will modify the `EmployeeTest` code to make use of this inheritance structure.

You need to define a class, `Manager`, where `Manager` extends the `Employee` class as shown below:



## Practical

### Part 1. Defining an Extended Class

1. In the project you created earlier, define a new class `Manager`, without a `main()` method, belonging to the same package as before.
2. Modify the class definition so that `Manager` is a subclass of `Employee`.
3. You must define a constructor for this class, so provide one which takes similar arguments as for `Employee`, and using the `super(...)` syntax, invoke the superclass constructor to initialise the variables for the name and age.
4. In `EmployeeTest`, create a third object, but this time a `Manager`, with suitable constructor arguments. Print out the details, just as you did for the `Employee`s. (You can safely comment out the looping code for testing `incAge()` etc. – we won't be needing that now.)

5. Refactor the code in `EmployeeTest`. Instead of the three separate declarations, create an array of three `Employee` references, and initialise two `Employee`s and a `Manager` for the array elements.
6. Use a loop to print out the details for each object in the array.
7. Edit the `Manager` class to provide an instance variable called '`manages`' of type `Employee[]`. This variable should of course be `private`! In the constructor, initialise this giving the array a capacity of 100. Provide another instance variable for denote the last free position of the array. (According to an EU directive, a manager can only manage a maximum of 100 employees, something we will abolish later when we cover Collections!).
8. Define an instance method `addEmployee()`, taking an `Employee` as a parameter and returning `void`. Implement this method to add the employee object to the array, in the last free position, and remember to increment this index.
9. Define another method `getEmployeeNames()`, with no arguments, but which returns a `String`. Implement this method to loop around the collection (using a `for` loop), concatenating the names from each `Employee`. Finally return this `String`. (You might like to do some String handling, like adding commas between names!)
10. Enhance `EmployeeTest`, so that you have a `Manager` who 'manages' the other two `Employee`s, by calling the `addEmployee()` method.
11. In `Employee`, provide a new instance method `getDetails()`, which takes no arguments but returns a `String`. Implement the code to return the name and age of the `Employee`, concatenated as a `String`. Modify the loop in `EmployeeTest` to use this method, rather than print the name and age separately. Test you program.
12. Now in `Manager`, *override* the `getDetails()` method to return not only the name and age, but also the names of `Employee`s who are managed, e.g.

Sally, 39; manages: Fred, James

Make use of your `getEmployeeNames()` method to do this.

13. Save and run your application. Does it work correctly? Note the use of polymorphism.
14. You might want create more objects in the `main()` method, perhaps some `Manager`s. Does it make sense to have a `Manager` who manages other `Manager`s? Is there anything you need to do in the `Manager` class to allow this?
15. What happens if you print out the managed employees' full details? Can anything go wrong with this?