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 `Employees`. (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 `Employees` 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 `Employees`, 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 `Employees` 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 `Managers`. Does it make sense to have a `Manager` who manages other `Managers`? 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?