# CSC 251 Project 1 – HR Application for Employees and Payroll

## Overview and Requirements

You are to write an application to allow a Human Resources department enter Employee, Manager and Executive data, generate payroll information, and list various subsets of employees. This program uses enhanced versions of the Employee/Manager/Executive program in the Inheritance lab, and enhances DataSetGeneric. Reusing code from those labs can significantly help in completing this project.

In particular, the user of this application will be able to

* Create employees, managers, and executives and store them in a data store.
* List all employees,
* List only regular employees (i.e., omitting managers and executives)
* List only Managers
* List only Executives
* Show the highest paid employee
* Display Employees sorted by name
* Display Employees sorted by salary
* Generate a payroll

## Objectives

After completing this project, students will have

* Enhanced a previously written generic data store to provide a generalized sub setting mechanism based on the Screener interface
* Enhanced a previously written hierarchy of classes for Employee/Manager/Executive to implement Measurable
* Completed an application that uses the data store to keep specific application objects
* Written a generic sub setting interface, *Screener*, and associated classes to subset the contents of the data store using various criteria
* Written JUnit tests for several methods

## Organization of Project Resources

This document, the Javadoc web pages and starter code are organized more as reference material than a tutorial on what you should do. An unspoken part of the project is connecting the dots to develop a full understanding of the whole program.

I suggest making a list of the concepts and components of the project, the locations they are referenced, and your questions associated with them. As you work through this document, the Javadoc and starter code, you should be able to answer the questions you wrote previously.

## Provided Materials

The attached zip file contains two folders:

* doc contains Javadoc information for some of the classes and interfaces the program requires. Code that you write must adhere to these specifications. This folder must be extracted (browsers don’t work well looking a files in an archive). Once extracted, open the index.html file with a browser. At the upper left of the window, “ALL CLASSES” will display links to all of the classes in the folder. Javadoc pages are provided for
  + DataSetGeneric , which is a class you must provide. There are enhancements to the DataSetGeneric class beyond your class from previous lessons. Note the package specification for DataSetGeneric. My DataSetGeneric solution uses inheritance, but you are free to choose a different solution. But your implementation must support the methods defined in the javadoc, plus *add* and *size* methods with the same behavior as *ArrayList.add* and *ArrayList.size*.
  + Payroll, the main method for the program.
  + Utils contains static methods for menu management.
  + Measurable, an interface exploited by DataSetGeneric and implemented by the Employee/Manager/Executive classes
  + Screener, an interface that must be created and implemented to exploit the DataSetGeneric enhancements.
  + Note the package specifications on these classes and interfaces.
* src contains the package structure and starter code. Starter code for the following interfaces and classes are provided
  + Measurable is defined in full
  + Payroll, which you must complete. **DO NOT MODIFY THE *main* METHOD**. You can complete the project by completing the method bodies such as addEmployee and displayEverybody, etc.
  + Utils is provided in full

## Package Structure and Classes

Use the following packages/classes structure for your program:

* edu.waketech.csc251.collection
  + DataSetGeneric
  + Measurable
* edu.waketech.csc251.hr
  + Payroll
* edu.waketech.csc251.hr.mgmt
  + Executive
  + Manager
  + The classes implementing Screeners to select Managers and Executives should be in this package
* edu.waketech.csc251.hr.person
  + Employee
  + The class implementing Screener for selecting Employees who are not also Manager or Executive should be in this package
* edu.waketech.csc251.hr.test
  + Test code for testing Employee/Manager/Executive should go here
* edu.waketech.csc251.test
  + Test code for testing DataSetGeneric should go here
* edu.waketech.csc251.tools
  + Screener
  + Utils

## Other Requirements

### JUnit

Provide JUnit tests for all completely new methods. For example, each *Screener* implementation should have its *test* method called with an *Executive*, a *Manager* and an *Employee* object, and the correct value verified.

*DataStoreGeneric’*s *getList* methods should be called and their return values verified. In particular, the *getList(Screener*….*)* signature should verify that various object types are correctly include or excluded from the output.

These tests don’t have to be large or elaborate. Create test dummy Employee, Manager, and Executive objects (hard coded), and call the various *Screener*’s with each of them. Verify the proper *true*/*false* return value of *test*. That’s nine very simple tests.

Similarly, create a *DataSetGeneric* object and feed your dummy *Employee*/*Manager*/*Executive* objects to it. Then test *getList()*, and *getList* with each *Screener*, and verify the output. That would be four tests, each with the same setup.

### Measurable

The measure of an Employee is the Employee’s salary.

## Hints and Suggestions

* Don’t use Scanner’s nextLine. Keep names (employee name, department names) limited to a single word; that’s how I’ll test. Doing this will let you limit Scanner to next, nextInt, and nextDouble.
* Plan the order of your development and testing. One strategy would be to comment out all but one method definition and the associated *if* statement in Payroll. Complete that method in DataSetGeneric. Test/debug that method. When that’s working, uncomment another user action, test it, etc. Program in small chunks, and test immediately.
* For *Screener* implementations, don’t forget the *instanceof* operator. Plus, the following incantation will return a *String* with the simple name of the concrete class for an object *obj*: *obj.getClass().getSimpleName()*. So if *obj* was created via *new Manager*, the simple name will be “Manager.” *obj instanceof Employee* is *true* in this case, because a *Manager* object is also an *Employee* object.

## Grading Elements

The program

* Contains all classes and interfaces as described in the Javadoc documentation
* Contains an inheritance hierarchy Employee/Manager/Executive as defined in the Inheritance lesson
* Enhances Employee/Manager/Executive to be Measurable
* Contains a generic DataSetGeneric class. Objects of this class are declared to contain only Employees or its subclasses
* Enhances DataSetGeneric to use Screener objects to return subsets of its elements
* Contains Screener objects to identify Employees, Managers and Executives
* Has all classes and interfaces in the appropriate packages
* Enhances the Payroll program to support the specified user interaction, especially the Screener-related and sorting features
* Payroll exploits DataSetGeneric fully, and does not duplicate or replace any of its capabilities
* Does not modify Payroll.main
* Provides at least thirteen different JUnit tests
  + each implementation of Screener is tested with an Employee, a Manager and an Executive object (nine tests)
  + DataSetGeneric.getList() that returns the entire store (one test)
  + DataSetGeneric.getList with a Screener parameter is tested that the elements of the store are returned, or not, appropriately (three tests)

## Sample Execution

Make a Selection:

[0] Exit

[1] Add Employee

[2] Add Manager

[3] Add Executive

[4] List All

[5] List (Regular) Employees

[6] List Managers

[7] List Executives

[8] Show Highest Salary Person

[9] Generate Payroll

[10] Display Sorted List by Name

[11] Display Sorted List By Salary

2

Name pointy-hair

Enter Salary 66000

Enter Department Managed development

Make a Selection:

[0] Exit

[1] Add Employee

[2] Add Manager

[3] Add Executive

[4] List All

[5] List (Regular) Employees

[6] List Managers

[7] List Executives

[8] Show Highest Salary Person

[9] Generate Payroll

[10] Display Sorted List by Name

[11] Display Sorted List By Salary

1

Name wally

Enter Salary 31000

Make a Selection:

[0] Exit

[1] Add Employee

[2] Add Manager

[3] Add Executive

[4] List All

[5] List (Regular) Employees

[6] List Managers

[7] List Executives

[8] Show Highest Salary Person

[9] Generate Payroll

[10] Display Sorted List by Name

[11] Display Sorted List By Salary

3

Name seagull

Enter Salary 88000

Enter Department Managed development

Enter bonus .3

Make a Selection:

[0] Exit

[1] Add Employee

[2] Add Manager

[3] Add Executive

[4] List All

[5] List (Regular) Employees

[6] List Managers

[7] List Executives

[8] Show Highest Salary Person

[9] Generate Payroll

[10] Display Sorted List by Name

[11] Display Sorted List By Salary

1

Name dilbert

Enter Salary 33000

Make a Selection:

[0] Exit

[1] Add Employee

[2] Add Manager

[3] Add Executive

[4] List All

[5] List (Regular) Employees

[6] List Managers

[7] List Executives

[8] Show Highest Salary Person

[9] Generate Payroll

[10] Display Sorted List by Name

[11] Display Sorted List By Salary

3

Name dogbert

Enter Salary 99999

Enter Department Managed hr

Enter bonus .6

Make a Selection:

[0] Exit

[1] Add Employee

[2] Add Manager

[3] Add Executive

[4] List All

[5] List (Regular) Employees

[6] List Managers

[7] List Executives

[8] Show Highest Salary Person

[9] Generate Payroll

[10] Display Sorted List by Name

[11] Display Sorted List By Salary

4

[0] Manager [department=development, toString()=Employee [name=pointy-hair, salary=66000.0]]

[1] Employee [name=wally, salary=31000.0]

[2] Executive [bonus=0.3, getSalary()=114400.0, getDepartment()=development, getName()=seagull]

[3] Employee [name=dilbert, salary=33000.0]

[4] Executive [bonus=0.6, getSalary()=159998.4, getDepartment()=hr, getName()=dogbert]

Press any key and enter to Continue:

x

Make a Selection:

[0] Exit

[1] Add Employee

[2] Add Manager

[3] Add Executive

[4] List All

[5] List (Regular) Employees

[6] List Managers

[7] List Executives

[8] Show Highest Salary Person

[9] Generate Payroll

[10] Display Sorted List by Name

[11] Display Sorted List By Salary

5

[0] Employee [name=wally, salary=31000.0]

[1] Employee [name=dilbert, salary=33000.0]

Press any key and enter to Continue:

x

Make a Selection:

[0] Exit

[1] Add Employee

[2] Add Manager

[3] Add Executive

[4] List All

[5] List (Regular) Employees

[6] List Managers

[7] List Executives

[8] Show Highest Salary Person

[9] Generate Payroll

[10] Display Sorted List by Name

[11] Display Sorted List By Salary

6

[0] Manager [department=development, toString()=Employee [name=pointy-hair, salary=66000.0]]

Press any key and enter to Continue:

x

Make a Selection:

[0] Exit

[1] Add Employee

[2] Add Manager

[3] Add Executive

[4] List All

[5] List (Regular) Employees

[6] List Managers

[7] List Executives

[8] Show Highest Salary Person

[9] Generate Payroll

[10] Display Sorted List by Name

[11] Display Sorted List By Salary

7

[0] Executive [bonus=0.3, getSalary()=114400.0, getDepartment()=development, getName()=seagull]

[1] Executive [bonus=0.6, getSalary()=159998.4, getDepartment()=hr, getName()=dogbert]

Press any key and enter to Continue:

x

Make a Selection:

[0] Exit

[1] Add Employee

[2] Add Manager

[3] Add Executive

[4] List All

[5] List (Regular) Employees

[6] List Managers

[7] List Executives

[8] Show Highest Salary Person

[9] Generate Payroll

[10] Display Sorted List by Name

[11] Display Sorted List By Salary

8

[0] Executive [bonus=0.6, getSalary()=159998.4, getDepartment()=hr, getName()=dogbert]

Press any key and enter to Continue:

x

Make a Selection:

[0] Exit

[1] Add Employee

[2] Add Manager

[3] Add Executive

[4] List All

[5] List (Regular) Employees

[6] List Managers

[7] List Executives

[8] Show Highest Salary Person

[9] Generate Payroll

[10] Display Sorted List by Name

[11] Display Sorted List By Salary

9

[0] Pay pointy-hair $66000.0

[1] Pay wally $31000.0

[2] Pay seagull $114400.0

[3] Pay dilbert $33000.0

[4] Pay dogbert $159998.4

Press any key and enter to Continue:

x

Make a Selection:

[0] Exit

[1] Add Employee

[2] Add Manager

[3] Add Executive

[4] List All

[5] List (Regular) Employees

[6] List Managers

[7] List Executives

[8] Show Highest Salary Person

[9] Generate Payroll

[10] Display Sorted List by Name

[11] Display Sorted List By Salary

10

[0] Employee [name=dilbert, salary=33000.0]

[1] Executive [bonus=0.6, getSalary()=159998.4, getDepartment()=hr, getName()=dogbert]

[2] Manager [department=development, toString()=Employee [name=pointy-hair, salary=66000.0]]

[3] Executive [bonus=0.3, getSalary()=114400.0, getDepartment()=development, getName()=seagull]

[4] Employee [name=wally, salary=31000.0]

Press any key and enter to Continue:

x

Make a Selection:

[0] Exit

[1] Add Employee

[2] Add Manager

[3] Add Executive

[4] List All

[5] List (Regular) Employees

[6] List Managers

[7] List Executives

[8] Show Highest Salary Person

[9] Generate Payroll

[10] Display Sorted List by Name

[11] Display Sorted List By Salary

11

[0] Employee [name=wally, salary=31000.0]

[1] Employee [name=dilbert, salary=33000.0]

[2] Manager [department=development, toString()=Employee [name=pointy-hair, salary=66000.0]]

[3] Executive [bonus=0.3, getSalary()=114400.0, getDepartment()=development, getName()=seagull]

[4] Executive [bonus=0.6, getSalary()=159998.4, getDepartment()=hr, getName()=dogbert]

Press any key and enter to Continue:

x

Make a Selection:

[0] Exit

[1] Add Employee

[2] Add Manager

[3] Add Executive

[4] List All

[5] List (Regular) Employees

[6] List Managers

[7] List Executives

[8] Show Highest Salary Person

[9] Generate Payroll

[10] Display Sorted List by Name

[11] Display Sorted List By Salary

0

Bye