Assignment 6

Refer to Canvas for assignment due dates for your section.

Objectives:
- Write recursive implementations of ADTs.
- Test ADTs.

General Requirements

Create a new Gradle project for this assignment in your course GitHub repo. Make sure to follow the instructions provided in "Using Gradle with IntelliJ" on Canvas.

Create a separate package for each problem in the assignment. Create all your files in the appropriate package.

To submit your work, push it to GitHub and create a release. Refer to the instructions on Canvas.

Your repository should contain:
- One .java file per Java class.
- One .java file per Java test class.
- UML diagrams for each problem. UML diagrams can be generated using IntelliJ or hand-drawn.
- All non-test classes and non-test methods must have valid Javadoc.

Your repository should **not** contain:
- Any .class files.
- Any .html files.
- Any IntelliJ specific files.

For both problems in this assignment, your underlying data structure must be recursive.
You may not use any of Java's built-in collections (e.g., LinkedList) or maps (e.g., HashMap).

Problem 1

Your task is to implement an immutable Priority Queue (PQ). A priority queue is a data structure, where every element of a PQ contains two properties:
1. A priority - an Integer
2. A value associated with the priority - in our case the value will be a String.

Your PQ implementation must support the following ADT operations:

- `PriorityQueue createEmpty()`: Creates and returns an empty PQ.
- `Boolean isEmpty()`: Checks if PQ is empty. Returns true if the PQ contains no items, false otherwise.
- `PriorityQueue add(Integer priority, String value)`: Adds the given element (the priority and its associated value) to the PQ.
- `String peek()`: Returns the *value* in the PQ that has the *highest* priority.
  - For two positive integers, `i` and `j`. If `i` < `j` then `i` has a lower priority than `j`. The PQ remains unchanged. Calling `peek()` on an empty PQ should throw an exception.
- `PriorityQueue pop()`: Returns a copy of the PQ without the element with the *highest* priority. Calling `pop()` on an empty PQ should throw an exception.

Multiple elements in the PQ may have the same priority, which will impact `peek()` and `pop()`. You may choose how to handle this situation but be sure to describe how you handle it in the documentation for the affected methods.

Problem 2

You are a part of a team developing a new **payroll system** for some company. Your team is tasked with specifying an ADT `PayrollSystem`, which will be used to store and manage individual employee information.
A class `Employee`, which stores contact and demographic information of every employee has already been written, and it is provided below for your convenience.

Your ADT `PayrollSystem` will need to support the following functionality:

- Check if the `PayrollSystem` is empty.
- Count the number of `Employees` in the `PayrollSystem`.
- Add an `Employee` to the `PayrollSystem`. Please note that `PayrollSystem` **does not allow** duplicate `Employees`, but it allows `Employee` roles to change. So, if some `Employee` already exists in the `PayrollSystem`, you want to update that `Employee’s` **current role information**.
- Remove a specified `Employee` from the `PayrollSystem`. If the `Employee` does not exist in the `PayrollSystem`, the system should throw `EmployeeNotFoundException`, which you will have to implement yourself.
- Check for a specified `Employee` in the `PayrollSystem`.
- Find and return all `Employees` from the `PayrollSystem` whose annual earnings are over $150 000.
- Find and return all `Employees` from the `PayrollSystem` who are employed in the same role, provided as an input argument `String currentRole`.
- Find and return all `Employees` from the `PayrollSystem` who joined the company in the same year, provided as input argument of the data type `Integer`.

As always, your implementation should include the `equals()` method. To determine if two `PayrollSystems` are equal, remember that the order of `Employees` stored in

`PayrollSystems` does not matter. If the exact same elements are present in both `PayrollSystems`, they should be equal.