OOP
Lab 10.

🎯 Objectives

- Sorting:
  - Natural ordering: **`Comparable<T>`**
  - Arbitrary ordering: **`Comparator<T>`**
- Anonymous classes and anonymous functions (lambdas)

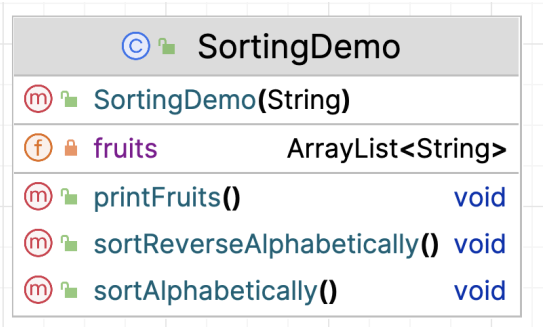
Create a Project/Module with 3 packages: `lab10_1, lab10_2, lab10_3`

Exercise 1.

Ask an LLM to generate a list of fruits, at least 20. You may use ChatGPT or Gemini. Copy the generated content into a text file.

Implement the `SortingDemo` class as shown in the diagram.

- The constructor reads the fruits from the text file.
- The `sort` methods sort the fruits alphabetically, and reverse alphabetically, respectively.
- The `printFruits` methods print the fruits separated by spaces.

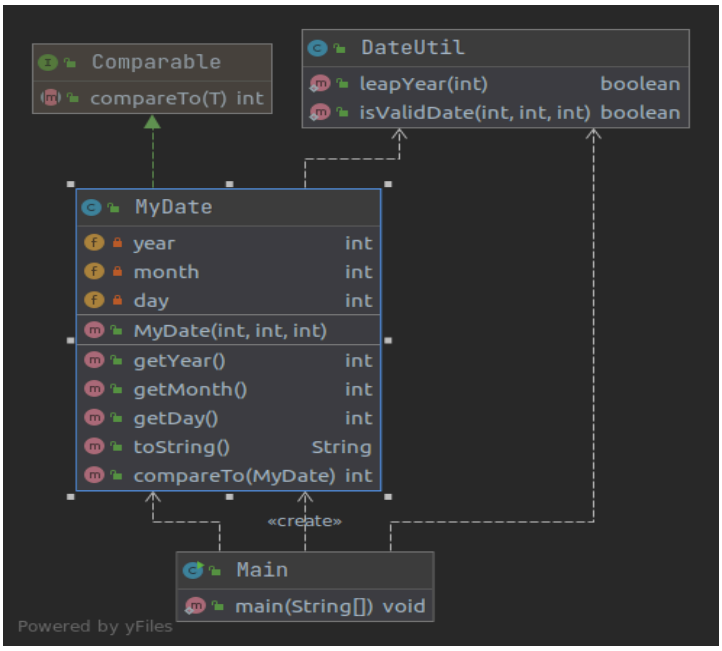


Test your class: create an instance of the class and call its methods.
The output should be transparent.

Exercise 2.

In this exercise you have to modify exercise 3 from Lab 2 (lab2_3 package).

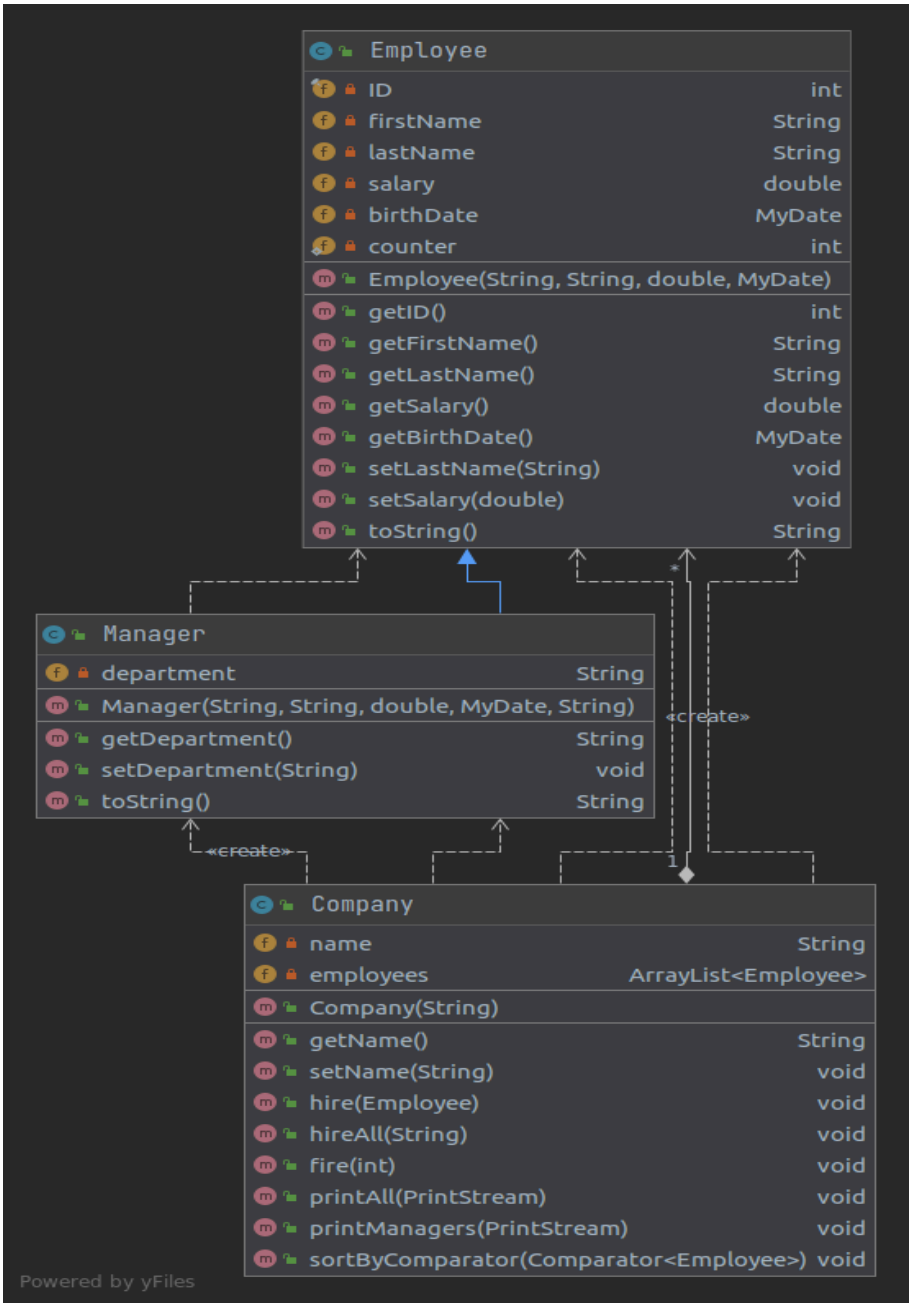
- Create a copy of the `DateUtil` and the `MyDate` classes.
- Add a **natural ordering** to the `MyDate` class in order to be able to compare two dates (implement the **Comparable<MyDate>** interface!).
- Main class - main method:
  - Generate 10 valid dates from the current year and store them in an `ArrayList`
  - Print the `ArrayList` to the standard output
  - Sort the `ArrayList`
  - Print the `ArrayList` to the standard output



OOP
Lab 10.

Exercise 3.

Implement the classes on the following class diagram:



Explanation:

- `hireAll (String csvFile)`: hires all the employees read from
- `sortByComparator(`**`Comparator<Employee>`** `comp)`: sorts the employees using the comp object
- `Input file example: employees.csv`

```
Incze, Zsolt-Tamas, 3100, 2000, 9, 25, WebDev
Kacso, Robert, 2900, 1998, 11, 15
Dumbravean-Katai, David, 3200, 2000, 8, 25
Balint, Zsolt, 1500, 1998, 2, 1
Fuzi, Zalan, 2100, 1999, 7, 9
Horvath, Janos, 2500, 1999, 12, 1
Bagoly, Norbert, 3000, 1999, 10, 1
Gabos, Alpar, 2900, 1997, 6, 1
Burszan, Hunor, 3500, 2000, 5, 3, MobileDev
```

- `sortByComparator` using **anonymous class**:

```
System.out.println("Alphabetically: ");
comp.sortByComparator(new Comparator<Employee>() {
   @Override
   public int compare(Employee e1, Employee e2) {
       // compare e1 to e2
   }
});
comp.printAll(System.out);
```

- `sortByComparator` using **lambda (anonymous function)**:

```
System.out.println("Alphabetically: ");
comp.sortByComparator(
    (Employee e1, Employee e2) -> {
       // // compare e1 to e2
   }
});
comp.printAll(System.out);
```

- Sort the employees by the following criteria (one by one):
  - Alphabetically
  - Decreasing salary order

- Managers followed by employees, both categories sorted alphabetically. Expected output for this criterion:

```
Manager{ID=9 ,firstName='Burszan', lastName='Hunor', ...}
Manager{ID=1 ,firstName='Incze', lastName='Zsolt-Tamas', ...}
Employee{ID=7, firstName='Bagoly', lastName='Norbert', ...}
Employee{ID=4, firstName='Balint', lastName='Zsolt', ...}
Employee{ID=3, firstName='Dumbravean-Katai', lastName='David', ...}
Employee{ID=5, firstName='Fuzi', lastName='Zalan', ...}
Employee{ID=8, firstName='Gabos', lastName='Alpar', ...}
Employee{ID=6, firstName='Horvath', lastName='Janos', ...}
Employee{ID=2, firstName='Kacso', lastName='Robert', ...}
```