

CSC 240: Computer Science III
Fall 2026 – Section 01
Homework 2 – Payroll (Chapter 10)

Name: _____

WCU ID: _____

Assigned: Thursday, September 24, 2026

Due: Tuesday, Oct 6, 2026, 11:59 PM

This homework contains 6 pages and 4 questions, worth a total of 100 points. It counts for **6%** of your course grade. You will build a small weekly payroll system that uses every Chapter 10 tool from Classes 6–9: **extends** and **super**, overriding, an **abstract class**, polymorphism and dynamic binding, an **interface** shared by unrelated classes, `equals()/toString()`, and `Comparable<T>`. The CSV loading in Q4 reuses the Chapter 9 tokenizing from HW1.

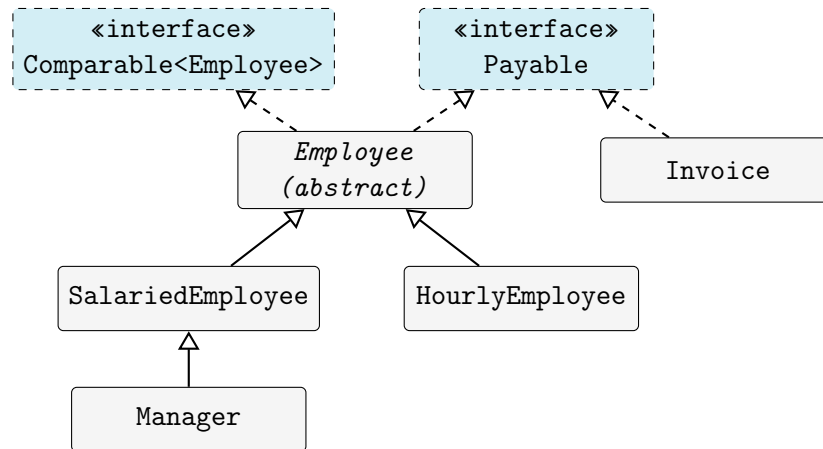
(DO NOT copy code from the book, slides, StackOverflow, YouTube, GitHub, ChatGPT, etc. See Section 6 of the syllabus.)

Instructions – read before you start

- **Individual work.** You may discuss ideas, but you may not share, exchange, or jointly write code. You must be able to explain and modify every line you submit.
- **Starter code.** Download `CSC240_HW2_starter.zip` from the course website (`wcu.ninja/csc240`) or D2L. It contains `Payable.java`, `Employee.java`, `SalariedEmployee.java`, `HourlyEmployee.java`, `Manager.java`, `Invoice.java`, `PayrollApp.java`, and `employees.csv`. The starter compiles as-is. **Do not modify** `Payable.java` or `employees.csv`, and keep every given field, constructor, and method signature; add your code where the `TODO` comments are. You may add private helper methods.
- **Submission.** Put your name in the comment header of each file. Submit the **seven** `.java` files in one zip named `LastName_FirstName_HW2.zip` to the D2L dropbox before the deadline. The D2L timestamp is authoritative.
- **Code must compile** with the JDK version listed in D2L. A file that does not compile receives at most half of that question's points. Partially working programs receive partial credit when you clearly comment what works and what does not.
- **Answers without code earn 0 points.** Every number your program prints must be computed by your Java code.
- **Style** (up to -5 points overall): meaningful names, consistent indentation, `@Override` on every overriding method, and a short comment for each part.
- **Late work:** no credit unless a valid excuse is communicated before the deadline (syllabus, Section 7).

Question:	1	2	3	4	Total
Points:	35	15	25	25	100
Score:					

The class design



solid arrow = extends (is-a) dashed arrow = implements (can-do)

1. The Employee hierarchy (35 points) – abstract classes, extends, overriding

(a) (15 points) **Employee (abstract)**. In `Employee.java`:

- declare `public abstract double weeklyPay();` – no body;
- add `public String getKind()` returning `"Employee"`;
- override `toString()` to return exactly
`String.format("%-5s %-20s %-9s $%9.2f",`
`id, name, getKind(), weeklyPay());`

Notice that `toString()` lives in `Employee` but calls `weeklyPay()` and `getKind()`. At run time Java calls the *subclass* versions (dynamic binding), so you write this one `toString()` once, and it works for every kind of employee.

- (b) (8 points) **SalariedEmployee**. Its weekly pay is `annualSalary / 52`, and its kind is `"Salaried"`.
- (c) (12 points) **HourlyEmployee**. The first 40 hours are paid at `hourlyRate`; each hour over 40 is paid at `1.5 * hourlyRate` (use the constants `REGULAR_HOURS` and `OVERTIME_RATE`). Kind is `"Hourly"`. *Self-check: 22.50/h for 45 hours is \$1068.75; 31.00/h for 38 hours is \$1178.00.*

2. (15 points) **Manager** – multi-level inheritance, `super`

A **Manager** is a **SalariedEmployee** who also receives a fixed `weeklyBonus`. Finish `Manager.java`: store the bonus, add `getWeeklyBonus()`, override `getKind()` to return

"Manager", and override `weeklyPay()` to return the salaried weekly pay **plus** the bonus. You **must** call `super.weeklyPay()`; do not repeat the `/ 52` calculation. (`annualSalary` is private in `SalariedEmployee`, so you couldn't use it directly anyway.) *Self-check: salary 104000 with bonus 250 gives \$2250.00 per week.*

3. Interfaces: Payable, equals() and Comparable (25 points)

`Payable` (given, do not modify) describes *anything the company pays this week*:

```
public interface Payable {
    double getPaymentAmount();
    String getPayee();
}
```

- (2 points) Change the header of `Employee` to `public abstract class Employee implements Payable, Comparable<Employee>`. (A class can extend one class but implement any number of interfaces.)
- (4 points) In `Employee`, implement `getPaymentAmount()` (returns `weeklyPay()`) and `getPayee()` (returns the name). The subclasses inherit both, so none of them change.
- (7 points) Override `equals(Object other)` in `Employee`: two employees are equal when their **ids** are equal. The parameter type must be `Object`. Check `instanceof Employee` before casting, and compare the ids with `equals`, not `==`. Also override `hashCode()` to return `id.hashCode()`.
- (7 points) Implement `compareTo(Employee other)`: an employee with the smaller `weeklyPay()` comes first (use `Double.compare`). If the pay is exactly equal, compare the ids.
- (5 points) `Invoice`. An invoice is *not* an employee and extends nothing, but the company still has to pay it. Make `Invoice` implements `Payable`: the payment is `quantity * unitPrice` and the payee is the vendor. Also override `toString()`, e.g. `Invoice: 3 x monitor from Dell`.

4. PayrollApp (25 points) – polymorphism in action

`employees.csv` has a header row, then one employee per line.

Each line has the format `type,id,name,pay,extra`:

S,E101,Ada Lovelace,78000,	<i>Salaried: pay = annual salary, no extra</i>
H,E102,Alan Turing,22.50,45	<i>Hourly: pay = hourly rate, extra = hours</i>
M,E103,Grace Hopper,104000,250	<i>Manager: pay = annual salary, extra = bonus</i>

- (8 points) `parseEmployee(String line)`. Split the line (use `split(",", -1)` so the empty last field of an S row is kept) and return a new `SalariedEmployee`, `HourlyEmployee`, or `Manager` depending on the type letter. The method returns `Employee`; the object you return is a subclass (a *Manager is an Employee*).

- (b) (4 points) **Duplicates.** The file contains one data-entry mistake: the same id appears twice. While loading, if `staff.contains(e)` is `true`, print `Skipping duplicate ID <id>: <name>` and do not add it. `ArrayList.contains` calls *your* `equals()`, so this works only if Q3(c) is correct.
- (c) (5 points) **Payroll report.** Print every employee in file order with `println(e)`, then the total weekly payroll. Do **not** use `instanceof` or casts here; `toString()` and `weeklyPay()` already do the right thing for each object.
- (d) (4 points) **Sorted report.** Copy the list, sort it with `Collections.sort` (which calls *your* `compareTo`), reverse it, and print the employees from highest to lowest pay.
- (e) (4 points) **All payments.** Build an `ArrayList<Payable>` containing every employee **and** the two invoices listed in the starter. Print each payee and amount, then the grand total. Use only `Payable` methods in this loop, because that is the only thing an employee and an invoice have in common.

Expected output (complete)

Your output must match this exactly, including spacing.

```
Skipping duplicate ID E104: K. Johnson
Loaded 10 employees.

=== Weekly payroll ===
E101 Ada Lovelace           Salaried $ 1500.00
E102 Alan Turing            Hourly    $ 1068.75
E103 Grace Hopper           Manager   $ 2250.00
E104 Katherine Johnson      Hourly    $ 1178.00
E105 Edsger Dijkstra        Salaried $ 1750.00
E106 Margaret Hamilton      Hourly    $ 1581.25
E107 Barbara Liskov         Manager   $ 2675.00
E108 Linus Torvalds         Hourly    $ 380.00
E109 Donald Knuth           Salaried $ 1250.00
E110 Frances Allen          Salaried $ 1600.00
Total weekly payroll: $15233.00

=== Sorted by weekly pay (highest first) ===
E107 Barbara Liskov         Manager   $ 2675.00
E103 Grace Hopper           Manager   $ 2250.00
E105 Edsger Dijkstra        Salaried $ 1750.00
E110 Frances Allen          Salaried $ 1600.00
E106 Margaret Hamilton      Hourly    $ 1581.25
E101 Ada Lovelace           Salaried $ 1500.00
E109 Donald Knuth           Salaried $ 1250.00
E104 Katherine Johnson      Hourly    $ 1178.00
E102 Alan Turing            Hourly    $ 1068.75
E108 Linus Torvalds         Hourly    $ 380.00

=== All payments this week ===
Ada Lovelace                $ 1500.00
Alan Turing                  $ 1068.75
Grace Hopper                 $ 2250.00
Katherine Johnson           $ 1178.00
Edsger Dijkstra              $ 1750.00
Margaret Hamilton           $ 1581.25
Barbara Liskov               $ 2675.00
Linus Torvalds               $ 380.00
Donald Knuth                 $ 1250.00
Frances Allen                $ 1600.00
Staples                      $ 515.88
Dell                         $ 658.50
Total to pay out: $16407.38
```

(There is a blank line before each === heading. We will also test your classes with **other** employees and CSV rows, so compute everything in code and don't print these numbers directly.)

Before you submit – checklist

- ☐ All seven .java files compile with no errors, and each has my name in its header comment.
- ☐ Employee is abstract, weeklyPay() has no body there, and toString() is written only once (in Employee).

- ☐ `Manager.weeklyPay()` calls `super.weeklyPay()`.
- ☐ `equals` takes an `Object` parameter; the duplicate row is skipped.
- ☐ No `instanceof` and no casts in `PayrollApp`'s report loops.
- ☐ My output matches the expected output exactly.
- ☐ Zip named `LastName_FirstName_HW2.zip` (seven `.java` files, no `.class` files) uploaded to D2L before the deadline.