CPSC 312 Mobile App Development (Android)

Gonzaga University

Gina Sprint

PA3 Inheritance and Interfaces (100 points)

Individual, non-collaborative assignment

Learner Objectives

At the conclusion of this programming assignment, participants should be able to:

- Implement a class hierarchy that utilizes inheritance
- Understand the purpose of the `super`, `extends`, and `implements` keywords
- Define an interface
- Implement a class that implements a programmer-defined interface and a standard Java interface
- CPSC 224 only: Utilize the javadoc tool to document your code and produce html documentation files in a doc/ folder. Details on how to do this are in the CPSC 224 coding standard document.

Prerequisites

Before starting this programming assignment, participants should be able to:

- Define classes
- Declare objects to instantiate programmer-defined classes
- Implement/invoke class methods
- Understand and implement basic object-oriented programming concepts

Acknowledgments

Content used in this assignment is based upon information in the following sources:

- Chris Peter's Inheritance Lab

Github Classroom Setup

For this assignment, you will use GitHub Classroom to create a private code repository to track code changes and submit your assignment. Open this PA3 link to accept the assignment and create a private repository for your assignment in Github classroom:

- CPSC 312: https://classroom.github.com/a/M8h-wsvz

- CPSC 224: https://classroom.github.com/a/ZSC_wafa

Your repo, for example, will be named GonzagaCPSC312/pa3-yourusername (where yourusername is your GitHub username). I highly recommend committing/pushing regularly so your work is always backed up. We will grade your most recent commit, even if that commit is after the due date (your work will be marked late if this is the case).

Overview and Requirements

This programming assignment is divided into two tasks, Task1 and Task2. Both tasks should be in the SAME IntelliJ IDEA project (this is not best practice but it'll be easier to submit and grade this way).

Task1: Inheritance (60 pts)

You are a programmer in the IT department of an important law firm. Your job is to create a program that will report gross salary amounts and other compensation for employees at your company.

Classes to Define

There are three types of employees in your firm:

- Programmers
- Lawyers
- Accountants

Your computer-based solution will use *inheritance* to reflect the "general-to-specific" nature of your employee hierarchy.

Common attributes (i.e. fields) for all employees are:

- `name` (a `String`)
- `salary` (a `double`)

Specific employees get extra compensation in the form of various "perks". Attributes for specific employee perks are:

- Programmers: bus pass provided (a `boolean`)
- Lawyers: stock options earned (an integer)
- Accountants: parking allowance amount (a `double`)

The three specific classes of employees should **extend the abstract `Employee` class** (nobody is a generic employee) and implement their own `reportSalary()` method.

The pay schedule is:

- Employees earn the base salary of $40K per year
- Accountants earn the base employee salary per year
- Programmers earn the base employee salary plus $20K per year

- Lawyers earn the base employee salary plus $30K a year

Additional Requirements

- Each class has constructors, a `toString()`, and relevant getters and setters
    - Hint: implement a `toString()` method in the `Employee` class. Override `toString()` in each subclass while still making use of the `toString()` you defined in `Employee` (think `super`!).
- **Each subclass of `Employee` must implement its own `reportSalary()` method** (hmmm….). The specific implementations of `reportSalary()` includes a `println()`
    - For example: `I'm a programmer. I make $60000.00 and I do not get a bus pass.`
- Salaries for specific types of employees are *in addition to* the base employee salary. Raising the base employee salary should automatically raise salaries for all types of employees (hmmm….).
- An object of the `Employee` class *cannot be instantiated* because it would be too general.
- Attribute(s) that belong to a super or sub class should be initialized in the corresponding class constructor.
- Format all monetary output to 2 decimal places using Java's `DecimalFormat` class.

Create a driver program that creates an array of 8 `Employee` objects that represents your company's employee base. Display information about your employees and report their salaries. Fill your array with the following employees:

Programmers	<Your name here>	No bus pass
	Ima Nerd	Bus pass
Lawyers	Kenny Dewitt	10 shares signing bonus
	Dan D. Lyon	0 shares signing bonus
	Willie Makit	100 shares signing bonus
Accountants	Hal E. Luya	$17.00 parking allowance
	Midas Well	$45.50 parking allowance
	Doll R. Bill	$2.50 parking allowance

Task2: Interfaces (40 pts)

In this task, we are going to define functionality for a car that implements two interfaces: one interface to "drive" a car and one interface to compare cars for sorting purposes.

Classes/Interfaces to Define

Define an interface called `Driveable`. `Driveable` contains the single abstract method:
`void drive(int milesDriven)`
Which represents driving a certain number of miles.

Define a class called `Car` that implements the `Driveable` interface and the Java `Comparable` interface. A `Car` has the following fields:

1. `make` (a `String`, e.g. "Toyota")
2. `model` (a `String`, e.g. "Prius")
3. `year` (an integer, e.g. 2005)
4. `odometerReading` (an integer, e.g. 30000)

`Car` should have constructors, a `toString()`, and relevant getters and setters. Here is an example string representation of a `Car` object:
`1985 DeLorean DMC-12 with 100000 miles`

Additional Requirements

- When "driving" the car, be sure to update the odometer reading on the car.
- Two cars are compared first by year, then by make, then by model, then by mileage (e.g. `odometerReading`)

Create a driver program that tests your `Car` class. At a minimum, your program should:

- Create an array of 10 `Car` objects (initialize the `Car`s in your array so enough "driving" can make the `Car`s out of order)
- Display the `Car`s in the array
- Sort the array
    - Hint: check out the `java.util.Arrays` class for a helper method to do this
- "Drive" a few cars in the array
- Re-sort and re-display the `Car`s in the array

Bonus (7 pts)

- (4 pts) In Task 2, write your own sort method instead of using the `java.util.Arrays` helper method.
    - Note: this is great interview practice!
- (3 pts) In both Task 1 and Task 2, use a Java collection type instead of an array type.
    - Hint: see the Java Tutorials "Collections" Trail Lesson "Lesson: Interfaces"

Submitting Assignments

1. Use Github classroom to submit your assignment via a Github repo. See the "Github Classroom Setup" section at the beginning of this document for details on how to do this. You must commit your solution by the due date and time.

2. Your repo should contain your entire IntelliJ IDEA project. Double check that this is the case by cloning (or downloading a zip) your submission repo and opening your project in IntelliJ IDEA and running your code.

Grading Guidelines

This assignment is worth 100 points + 7 points bonus. Your assignment will be evaluated based on a successful compilation and adherence to the program requirements. We will grade according to the following criteria:

- Task 1 (60 pts):
  - 25 pts for implementing the requirements of the `Employee` class
    - 5 pts for not being instantiable
    - 5 pts for correct fields and constructors(s)
    - 5 pts for correct `reportSalary()`
    - 5 pts for correct `toString()`
    - 5 pts for correct getters/setters
  - 15 pts for implementing the requirements of the 3 `Employee` subclasses
    - 5 pts for each correct subclass
  - 5 pts for implementing a program driver class
  - 5 pts for formatting all decimal output to two decimal places
  - 10 pts for adherence to proper programming style and comments established for the class
    - Note: if you are in 224, this includes javadoc'ing your code and including a doc/ folder with your generated html files!!
- Task 2 (40 pts)
  - 5 pts for correctly defining the `Driveable` interface
  - 25 pts for implementing the requirements of the `Car` class
    - 5 pts for correct fields and constructors(s)
    - 5 pts for correctly implementing `Driveable`
    - 7 pts for correctly implementing `Comparable`
    - 5 pts for correct `toString()`
    - 3 pts for correct getters/setters
  - 5 pts for implementing a program driver class
  - 5 pts for adherence to proper programming style and comments established for the class
    - Note: if you are in 224, this includes javadoc'ing your code and including a doc/ folder with your generated html files!!