Assignment 3

Refer to Canvas for assignment due dates for your section.

Objectives:

- Implement solutions conforming to the object-oriented design principles of:
  - Encapsulation
  - Inheritance
  - Information hiding
  - Abstraction
- Write code that is modular and easy to extend
- Use inheritance to minimize code duplication
- Continue to meet the objectives of previous assignments

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.
- One pdf or image file for each UML Class Diagram that you create. 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.

Suggestions for approaching this assignment

You only have one problem to work on this week because this assignment requires you to think very carefully about the design of your solution. An object-oriented solution will make ***extensive***

use of inheritance and have little to no code duplication between classes—this is what we will look for when grading your work.

The specification is long and complex. Read the **whole** problem then take time to plan your design before you start coding. Start by identifying commonalities across the various services (see below) the system needs to account for. For example, look for services that could share fields or calculations. Only once you have a good sense of the relationships between the various services should you start coding. If you find yourself writing the same code in multiple places, consider how you can refactor your design to make better use of inheritance and other OOD principles.

Problem 1

You are developing an invoicing system for a company that provides property services (e.g. cleaning, gardening, painting to homeowners. You are expected to write the part of the system that will automatically calculate a price for a service based on the work to be done and the home's characteristics.

There are three main types of service that the company provides:

- Interior - work done on the inside of a home.
- Exterior - work done on the outside of a home.
- Specialist - work that requires particular qualifications or licenses.

Interior services can be one of the following:

- Cleaning
- Painting

Exterior services can be one of the following:

- Gardening
- Window cleaning

Specialist services can be one of the following:

- Electrical
- Plumbing

The system keeps track of the following information for **all services**:

- Property address, represented as a `String`
- Property size, represented as an `enum`. The size can be small, medium, or large.
- A `boolean` that indicates whether or not the service is carried out monthly.
- The number of services previously carried out at the address, represented as an `integer`. Retrieving and updating this value is not the responsibility of the invoicing system—you can assume that another part of the system will get the number of services when a service is created and update it when a service is completed.

Additionally, some service types need to track additional information;

- **All interior services**: the number of pets living at the address, an `integer`.
- **Window cleaning services**: the number of floors the property has, represented as an `integer`. The maximum number of floors that the company can work with is 3. It should not be possible to create a window cleaning service for a property with more than 3 floors.
- **All specialist services**: the number of licensed employees required to complete the work, represented as an `integer`, and a `boolean` that indicates if the work to be done is complex.
    - **All specialist services** require at least one licensed employee. If this requirement is not met, set the number of licensed employees to 1 unless the following point also applies.
    - **Complex work** requires at least 2 employees if the house is small or medium sized, and at least 3 if the house is large. If a new complex specialist service is created that doesn't meet these minimum requirements, increase the number of licensed employees to meet the requirement.
- **Electrical services** can require a maximum number of 4 licensed employees. It should not be possible to create an electrical service if more than 4 licensed employees are required.

Design and implement classes to represent the above services. Additionally, every service should have a method with the following signature:

```
public double calculatePrice()
```

When this method is called on a service, it will return the total price of the service based on the following rules:

- For **all services except specialist services**, the company charges a base rate of $80 per hour.
- For **all specialist services**, the base rate is $200 per licensed employee.
- For **exterior services**, the number of hours needed to complete the job is estimated as 1 hour for small properties, 2 hours for medium properties, and 4 hours for large properties.
- For **interior cleaning**, the number of hours needed to complete the service is estimated as 1 hour for small properties, 2 hours for medium properties, and 4 hours for large properties.
- For **painting**, the time needed to complete the service is 16 hours for small and medium-sized properties, and 24 hours for large properties.
- For **window cleaning**, a 5% fee is added to the price if the property has more than one floor.
- For **interior services**, a 5% fee is added to the price if the property has 1 or 2 pets. A 7% fee is added for 3 or more pets.

- For **gardening services**, the company charges a $20 waste removal fee.
- After all of the above charges have been calculated, the company will discount **every 10th service** by 50%. For example, if the number of past services is 9 or 19, a 50% discount should be applied. This discount cannot be applied to specialist services.
- After all of the above charges have been calculated, the company offers a 10% discount for **monthly** services. This discount should not be combined with the 10th service discount. If a 10th service discount is applied and the service is monthly, the monthly discount should not be applied. This discount cannot be applied to specialist services.
- For **electrical work**, a permitting fee of $50 is added to the total.
- For **plumbing work**, a permitting fee of $20 is added to the total.

Don’t forget to write tests and generate a UML diagram!