INSTRUCTIONS :

1. Complete all questions in your designated project group.
2. All members must contribute to writing the codes. (i.e. 1 question = 1 person, and share the workload if there's an additional question relative to the actual number of members in your team (i.e. 5)). Ensure that all members must understand and explain codes from any of the questions.
3. During viva, all students in each team will be randomly asked to describe, answer and edit any of the answers provided. Marks will be given to your ability to present the answers.

Lab Report

Prepare a report to solve the above problems. The report should contain all the sections as below for each question:

Section	**Description**
1. Problem	Description on the problem
2. Solution	Explanation on how to solve the problems below
3. Sample Input & Output	A few sets of input and output (snapshot)
4. Source code	Java source code

Requirements
1. Group Assignment (Grouping is the same as your project group)
2. Cover page that includes all student matric number and full name.
3. Font: Times New Roman 12, Line Spacing: 1 ½ Spacing
4. Submit to Spectrum according to your OCC. **Deadline** : Before your viva session (W13).

Question 1: Vehicle

Problem Statement:

In a bustling city where people rely on rented vehicles for their daily commutes, vacations, and business trips, a vehicle rental company offers a range of vehicles for customers to rent based on their needs. The company manages various vehicles such as cars, vans, and motorcycles with different rental rates.

You are tasked with designing an advanced **`Vehicle`** class for a rental system that handles various **types of vehicles, pricing models, and additional rental rules**. The system must support **fuel charges, and tiered discounting** based on both vehicle type and rental duration.

The **`Vehicle`** class should include:

a) Constructor:

- An empty constructor that initializes:
  I. `vehicleID` as "0000"
  II. `type` as "Car"
  III. `brand` as "Myvi"
  IV. `rentalRate` as 100/day
  V. `fuelLevel` as 100%

- A constructor that accepts parameters for `vehicleID`, `type`, `brand`, `rentalRate`, and `fuelLevel`.

b) Accessor and mutator methods for all fields.

c) Methods:

- `calculateRentalRate` :The rental rate should also be adjusted based on vehicle type:
  I. Vans: 20% more expensive than the base rate.
  II. Motorcycles: 25% less expensive than the base rate.

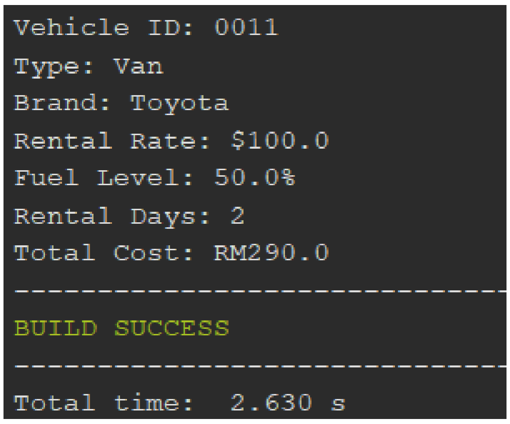
- `calculateRentalCost` (int input) : A method to calculate the total cost of renting the vehicle based on:

Discount tiers:
1. 5% discount for rentals between 7-14 days.
2. 10% discount for rentals longer than 14 days.

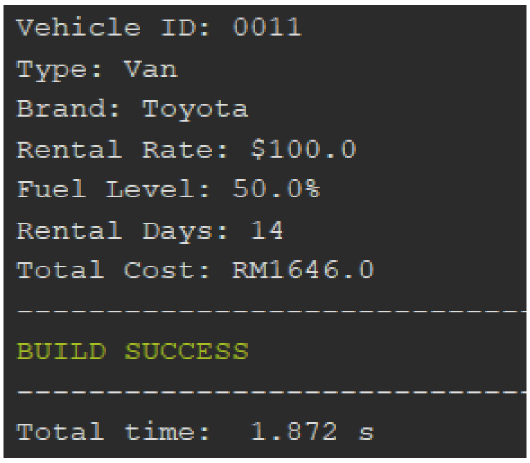
- `calculateFuelCharge`(): Calculates fuel charges based on the fuel level:
  - If fuel is below 50%, a RM50 penalty applies.

- `toString`(): Override the method to return a detailed summary of the vehicle's rental information, including fuel level and the final total cost.

Sample output:



```
Vehicle ID: 0011
Type: Van
Brand: Toyota
Rental Rate: $100.0
Fuel Level: 50.0%
Rental Days: 2
Total Cost: RM290.0
------------------------------
BUILD SUCCESS
------------------------------
Total time:  2.630 s
```



```
Vehicle ID: 0011
Type: Van
Brand: Toyota
Rental Rate: $100.0
Fuel Level: 50.0%
Rental Days: 14
Total Cost: RM1646.0
----------------------------
BUILD SUCCESS
----------------------------
Total time:  1.872 s
```

Question 2 : Hero Party

Problem Statement

Genshin Impact is an exciting open-world game featuring the beautiful world of Teyvat, where selected individuals in the world are granted elemental powers in the form of accessories known as Visions. These special heroes utilize one of 4 elements, which are Pyro, Cryo, Hydro and Electro, to defeat monsters and enemies that threaten the peace of their world. However, each of these monsters have their own strengths and weaknesses against elements, which means some level of strategy is necessary.

You are required to create a class `Hero` to store information about each hero, including their name, element and power. The element can only be one of the four aforementioned elements.

You should also create a class `Monster` to store information about the monster, including their name, pyro resistance, hydro resistance, electro resistance, cryo resistance and their health points (HP).

A class, `HeroParty`, is created to manage the heroes we have such that we may better organize and evaluate our heroes based on their combat abilities against monsters. First, sort and print the list of heroes based on their power. Then, we need to decide which heroes in the party can defeat a particular monster. Create a method `battleWinners(Monster enemy)` which will evaluate each hero's ability against the particular monster and print the name of heroes that will win. The hero is considered to win if their damage is equal to or exceeds the HP of the monster. The damage formula is given as:

$$\text{Damage}=\text{Power} \times (100\%-\text{ElementalRes}\%)$$

Note: ElementalRes refers to the corresponding elemental res based on the character's element.

In addition to ordinary monsters, there may sometimes be boss monsters which are much stronger than their ordinary counterparts. Thankfully, heroes do not have to work alone and can get the support of 1 other hero to help them. The elements of both hero will interact to provide the main hero with a special buff. The buffs are as follows:

Element Combination	**Buffs**
Pyro x Hydro	Damage ×1.5 for first hero
Pyro x Cryo	Damage ×2.0 for first hero
Pyro x Electro	Extra damage=Random number between 50 to 100
Hydro x Electro	Extra damage=(Random number between 1-20)×5
Hydro x Cryo	-
Electro x Cryo	Reduce resistance for second hero by 10%

Note: Damage multiplier is calculated at the end, after deducting elemental resistance.

Create a method called `battleBoss(Monster boss)`, which calculates the pair of heroes that will deal the highest damage to the enemy and the value of the damage. Note that the order of heroes in the pair is important. Both heroes will deal damage to the boss based on the original equation, plus whatever relevant buffs added. The order of element combination does not matter. Print the pair of heroes with highest damage and the value of the damage.

Test program

```
public class GenshinTest {
    public static void main(String[] args) {
        Hero amber = new Hero("Amber", "Pyro", 80.0);
        Hero kaeya = new Hero("Kaeya", "Cryo", 60.0);
        Hero lisa = new Hero("Lisa", "Electro", 100.0);
        Hero barbara = new Hero("Barbara", "Hydro", 40.0);

        Hero[] heroList = {amber, kaeya, lisa, barbara};
        HeroParty party = new HeroParty(heroList);
        party.sortList();

        System.out.println("List of heroes based on power: ");
        for (Hero hero : heroList){
            System.out.println(hero);
        }

        Monster mitachurl = new Monster("Mitachurl", 10, 10, 10, 10,
                                        80.0);

        System.out.println("\nHeroes that will win against " +
                           mitachurl.getName());
        party.battleWinners(mitachurl);

        System.out.println();
        Monster cryoRegisvine = new Monster("Cryo Regisvine", 20, 20, 20,
                                            40, 200.0);
        party.battleBoss(cryoRegisvine);
    }
}
```

Sample Output

```
List of heroes based on power:
Barbara
Kaeya
Amber
Lisa

Heroes that will win against Mitachurl
Lisa

The pair with the highest damage: Amber and Lisa
Total damage dealt: 223.0
```

Question 3: Room Rental Management System

Problem Statement:

Write a Java program where you are required to implement a class called **`Room`** with the following specifications:

Instance Variables:
- private String `roomNumber`: Unique room identifier (e.g., "101").
- private String `roomType`: Room type ("Single", "Double", or "Suite").
- private double `rentalPrice`: Rental price per night.

Static Variable:
- private static int `totalRooms`: Counts the total number of rooms created.

Constructor:
- Accepts `roomNumber`, `roomType`, and `rentalPrice` as parameters, initializes instance variables, and increments `totalRooms`.

Encapsulation:
- Provide public getter and setter methods for each instance variable.

Method Overloading:
- `calculateRent(int days)`**:** Calculates total rent for the specified number of days.
- `calculateRent(int days, double discount)`**:** Calculates total rent after applying a discount.

Wrapper Class:

- `getTotalRooms`(): Returns `totalRooms` as an Integer object.

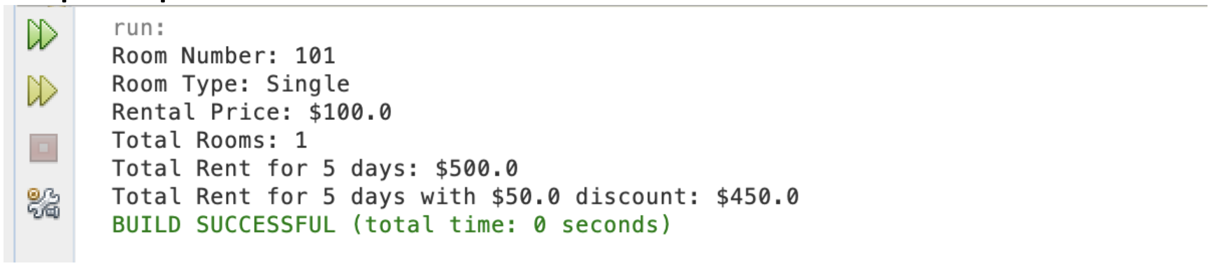
Static Method:

- `displayRoomDetails(Room room)`: Prints room details, including room number, type, rental price, and total rooms.

In the main method:

- Instantiate at least **Two (2)** Room objects.
- Use the methods to display room details and calculate rent with and without a discount.

Sample Output :



```
run:
Room Number: 101
Room Type: Single
Rental Price: $100.0
Total Rooms: 1
Total Rent for 5 days: $500.0
Total Rent for 5 days with $50.0 discount: $450.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Question 4: Player and Enemy

Problem Statement:

You are tasked with creating a simplified text-based adventure game in Java. The game has four primary entities: **`Player`, `Enemy`, `Item`** and **`Black Magic`**. Implement the following classes to manage these entities and create basic game interactions.

1. `Player` Class:

- **Attributes**:
  - String `name`: The name of the player.
  - int `health`: The player's health points (default is 100).
  - int `attackPower`: The power of the player's attacks (default is 10).

- **Methods**:
  - Constructor that takes the player's name and initializes `health` to 100 and `attackPower` to 10.
  - int `attack(Enemy enemy)`: Reduces the enemy's health based on the player's `attackPower`.
  - Accessor and mutator for `attackPower`.
  - `boolean isAlive()`: Returns true if the player's health is greater than 0.
  - Include a `toString` method that returns the combat log

2. `Enemy` Class:

- **Attributes**:
  - String `type`: The type of enemy (e.g., "Goblin", "Orc").
  - int `health`: The enemy's health points (default is 50).
  - int `attackPower`: The power of the enemy's attacks (default is 5).

- **Methods**:
  - Constructor that takes the type of the enemy and initializes `health` to 50 and `attackPower` to 5.
  - `void attack(Player player)`: Reduces the player's health based on the enemy's attackPower.
  - `boolean isAlive()`: Returns true if the enemy's health is greater than 0.
  - Include a `toString` method that returns the combat log.

3. `Item` Class:

- **Attributes**:
  - String `name`: The name of the item (but not limited to)
    - `Health Potion`
    - `Sword`
  - int `effect`: The effect the item has
    - `Health Potion`: Increases the Player's health by 20
    - `Sword`: Increases the Player's next attack power by 15.

- **Methods**:
  - Constructor that initializes the item's `name` and `effect`.
  - int `use(Player player)`: Uses the item's effect
  - Include a `toString` method that returns the combat log.

4. `Black Magic` Class: (used by Enemy)

- **Attributes**:
  - String `name`: The name of the magic (but not limited to:)
    - `Poison`
    - `Magic Orb`
  - int `effect`: The effect the item has
    - `Poison`: Damages the Player by 10, reducing the Player's next attack power by 10%
    - `Magic Orb`: Increases the Enemy's next attack power by 200% (300% of current attack power)

- **Methods**:
  - Constructor that initializes the item's `name` and `effect`.

- int `use(Player player, Enemy enemy)`: Uses the BlackMagic to enhance attack power or damage the player
- Include a `toString` method that returns the combat log.

Game Flow:

1. Create a Player object with a name of your choice.
2. Create an Enemy object with a type of your choice.
3. Create a few Item objects like a "Health Potion" that restores health and a "Sword" that increases attack power.
4. The player and enemy take turns attacking each other using the attack methods. (written in combat log that will be generated by toString method)
5. The player can use items to regain health or increase attack power during their turn.
6. The game ends when either the player or the enemy is defeated.

Question 5: Movie Collection System

Problem Statement:

In a world where stories come alive through cinema, you are tasked with creating a movie collection system that allows users to manage their movie collection seamlessly. This system should enable users to easily add, remove, search, and display their movies based on various criteria.

Your mission is to create a Java application that manages a collection of movies. The application should include the following components:

1. **`Movie` Class:**
   - **Attributes:**
     - `title`
     - `director`
     - `year`
     - `type`
   - **Constructor:** A constructor that initializes all attributes.
   - **Accessors:** Methods to retrieve the `title, director, year`, and `type of the movie.`
   - **`toString()` Method:** Override the **`toString`** method to return a formatted string representation of the movie details.

2. **`MovieCollection` Class:**
   - **Attributes:**
     - `movieCollection`
     - `numMovie`
   - **Constructor:** Initializes the movie collection with a maximum capacity (e.g., 100).
   - **Methods:**
     - **`isEmpty():`** Returns true if the collection is empty, otherwise false.

- **`addMovie(Movie movie)`:** Adds a movie to the collection. If the collection is full, it should print a message.
- **`removeMovie(String title)`:** Removes a movie by its title. If the movie is not found, it should print an appropriate message.
- **`findmovieCollectionByDirector(String director)`:** Prints all movies directed by the specified director. If none are found, it should print a message.
- **`findMovieByTitle(String title)`:** Searches for a movie by its title and prints its details if found.
- **`sortmovieCollectionByReleaseYear()`:** Sorts the movie collection by release year in ascending order.
- **`findmovieCollectionByReleaseYearRange(int startYear, int endYear)`:** Prints all movies released within the specified year range.
- **`displayCollection()`:** Displays all movies in the collection.

Test Program

```
public class TestProgram {

    public static void main(String[] args) {
        MovieCollection movieCollection = new MovieCollection();

        Movie movie1 = new Movie("Inception", "Christopher Nolan", 2010,
                                "Science Fiction");
        Movie movie2 = new Movie("The Shawshank Redemption", "Frank
                                 Darabont", 1994, "Drama");
        Movie movie3 = new Movie("Pulp Fiction", "Quentin Tarantino",
                                 1994, "Crime");
        Movie movie4 = new Movie("The Dark Knight", "Christopher Nolan",
                                 2008, "Action");
        Movie movie5 = new Movie("The Godfather", "Francis Ford Coppola",
                                 1972, "Crime");

        movieCollection.addMovie(movie1);
        movieCollection.addMovie(movie2);
        movieCollection.addMovie(movie3);
        movieCollection.addMovie(movie4);
        movieCollection.addMovie(movie5);

        System.out.println("Searching for movie(s) by director:");
     movieCollection.findmovieCollectionByDirector("Christopher Nolan");

        System.out.println("\nSearching for a movie by title:");
        movieCollection.findMovieByTitle("Inception");

        System.out.println("\nMovies released between 1990 and 2010:");
        movieCollection.findmovieCollectionByReleaseYearRange(1990,
        2010);


        System.out.println("\nSorted movieCollection by release year:");
        movieCollection.sortmovieCollectionByReleaseYear();
        movieCollection.displayCollection();

        movieCollection.removeMovie(movie3.getTitle());
        System.out.println("\nUpdated Movie Collection:");
        movieCollection.displayCollection();
    }
}
```

Sample Output

```

Searching for movie(s) by director:
Movies directed by 'Christopher Nolan':
- Title: Inception
  Director: Christopher Nolan
  Type: Science Fiction
  Year: 2010

- Title: The Dark Knight
  Director: Christopher Nolan
  Type: Action
  Year: 2008


Searching for a movie by title:
Movie 'Inception' found:
- Title: Inception
  Director: Christopher Nolan
  Type: Science Fiction
  Year: 2010


Movies released between 1990 and 2010:
- Title: Inception
  Director: Christopher Nolan
  Type: Science Fiction
  Year: 2010

- Title: The Shawshank Redemption
  Director: Frank Darabont
  Type: Drama
  Year: 1994

- Title: Pulp Fiction
  Director: Quentin Tarantino
  Type: Crime
  Year: 1994

- Title: The Dark Knight
  Director: Christopher Nolan
  Type: Action
  Year: 2008


Sorted movieCollection by release year:
- Title: The Godfather
  Director: Francis Ford Coppola
  Type: Crime
  Year: 1972
```

```
- Title: Pulp Fiction
  Director: Quentin Tarantino
  Type: Crime
  Year: 1994

- Title: The Shawshank Redemption
  Director: Frank Darabont
  Type: Drama
  Year: 1994

- Title: The Dark Knight
  Director: Christopher Nolan
  Type: Action
  Year: 2008

- Title: Inception
  Director: Christopher Nolan
  Type: Science Fiction
  Year: 2010

Removed movie: Pulp Fiction

Updated Movie Collection:
- Title: The Godfather
  Director: Francis Ford Coppola
  Type: Crime
  Year: 1972

- Title: The Shawshank Redemption
  Director: Frank Darabont
  Type: Drama
  Year: 1994

- Title: The Dark Knight
  Director: Christopher Nolan
  Type: Action
  Year: 2008

- Title: Inception
  Director: Christopher Nolan
  Type: Science Fiction
  Year: 2010
```

Question 6: Order Fulfilment System

Problem Statement:
You are tasked to create an order fulfilment system where customers place orders, and the system tracks inventory, processes orders, and generates invoices.
The system should handle the following tasks:

1. Add product(s) into inventory
2. Remove product(s) from inventory
3. Calculate total payment of the orders
4. Track the customer's inventory
5. Generate invoices for orders

The Java application should contain the following components:

1. `Product` **Class**
   - **Attributes:**
     - `name`
     - `price`
     - `stock`
   - **Constructor:** A constructor that initializes all attributes.
   - **Accessors:** Methods to retrieve the name and price of the product.
   - **Methods:**
     - **`reduceStock()`** : Decreases the stock when a customer orders the product.
     - **`toString()`** : Override the toString() method to return a formatted string that shows name and price of the product.

2. `Order` **Class**
   - **Attributes:**
     - `products`
   - **Constructor:** A constructor that initializes all attributes.
   - **Accessors:** Methods to retrieve the productList of the product.
   - **Methods:**
     - **`addProduct(Product product)`** : Adds a product to the order.
     - **`removeProduct(Product product)` :** Removes a product from the order.
     - **`calculateTotal()`** : Calculate and return the total of the order based on prices and quantities of the products added.

3. `Customer` **Class**
   - **Attributes:**
     - `name`
     - `email`
     - `shippingAddress`
   - **Constructor:** A constructor that initializes all attributes.
   - **Accessors:** Methods to retrieve the `email`, `name` and `shippingAddress` of the customer.
   - **Methods:**
     - **`toString()`** : Override the toString() method to return a formatted string representation of the customer's details.

4. `Inventory` **Class**
   - **Attributes:**
     - `productStock` : Stores products and their available quantities
   - **Constructor:** A constructor that initializes all attributes.
   - **Methods:**
     - **`addProduct(Product product, int qty)`** : Adds products to the inventory with their stock count.

- **`isAvailable(Product product, int qty)`** : Checks if a specific product and quantity are available in the stock.
- **`updateStock(Product product, int qty)`** : Updates the stock after a successful order.

5. `Invoice` **Class**
   - **Attributes:**
     - `order`
     - `customer`
   - **Constructor:** A constructor that initializes all attributes.
   - **Methods:**
     - **`toString()`** : Override the toString() method to return a formatted string that shows customer info, products purchased and the total price.

Test Program

```
public class TestProgram {
    public static void main(String[] args) {
        Product laptop = new Product("Laptop", 1200.00, 10);
        Product phone = new Product("Smartphone", 800.00, 5);
        Product powerBank = new Product("Power Bank", 100.00, 0);

        Customer customer = new Customer("John Doe", "john@example .com",
"1234 Main St");

        Inventory inventory = new Inventory();
        inventory.addProduct(laptop, 10);
        inventory.addProduct(phone, 5);
        inventory.addProduct(powerBank, 0);

        Order order = new Order();
        if (inventory.isAvailable(laptop, 1)) {
            order.addProduct(laptop);
            inventory.updateStock(laptop, 1);
        } else {
            System.out.println(laptop.getName() + " is out of stock.");
        }

        if (inventory.isAvailable(phone, 2)) {
            order.addProduct(phone);
            inventory.updateStock(phone, 2);
        } else {
            System.out.println(phone.getName() + " is out of stock.");
        }

        if (inventory.isAvailable(powerBank, 1)) {
            order.addProduct(powerBank);
            inventory.updateStock(powerBank, 1);
        } else {
            System.out.println(powerBank.getName() + " is out of stock.");
        }


        Invoice invoice = new Invoice(order, customer);
        System.out.println(invoice);
    }
}
```

Sample Output

```
Power Bank is out of stock.
Invoice for John Doe
Shipping Address: 1234 Main St

Products:
- Laptop: RM1200.00
- Smartphone: RM800.00

Total: RM2000.00
```