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: Hermione's Potion

Problem Statement:

In the magical world of Harry Potter, the corridors of Hogwarts School of Witchcraft and Wizardry are filled with the whispers of students eager to learn the ancient art of potion brewing. Among them is Hermione Granger, the brightest witch of her age, known for her diligence and exceptional talent. One day, while perusing the library's extensive collection of potion texts, Hermione stumbles upon an old, dusty book that describes a rare and powerful potion called the Invisibility Draught.

This potion is renowned for its ability to grant the drinker temporary invisibility, making it invaluable for stealthy adventures and secretive endeavors. However, brewing it requires precise amounts of specific magical ingredients: Unicorn Tears and Dragon Blood.

Determined to brew the potion perfectly, Hermione sets out to gather these ingredients from the Hogwarts pantry.

Your task is to design a program to manage the ingredients in a `PotionContainer`. The program should be able to **add ingredients, use them, and check whether enough ingredients are available** to brew the Invisibility Draught.

You need to create a class `Potion` with attributes for `ingredient(String)` and `volume(double)` and a method to **calculate the consume of ingredient** . A class, `PotionContainer`, will carry out various actions and calculations such as using **potion ingredients, checking remaining volume, ensuring there are enough ingredients** to brew a potion and **print out the volume of remain ingredients**.

Invisibility Draught Recipe:

- **Unicorn Tears**: 200 ml
- **Dragon Blood**: 150 ml

Test Program :

```
public class PotionTest {
    public static void main(String[] args) {
        PotionContainer container = new PotionContainer();

        System.out.println("Adding potions to the container...");
        container.addPotion("Unicorn Tears", 200.0);
        container.addPotion("Dragon Blood", 150.0);
        System.out.println("Potion container successfully initialized.\n");

        System.out.println("=== Using Potions ===");
        container.usePotion("Unicorn Tears", 50.0);
        System.out.println("Remaining volume of Unicorn Tears: " +
                         container.getRemainingVolume("Unicorn Tears") + " ml");

        container.usePotion("Dragon Blood", 30.0);
       System.out.println("Remaining volume of Dragon Blood: " +
        container.getRemainingVolume("Dragon Blood") + " ml");

       System.out.println("\nAttempting to use more Dragon Blood than
       available...");
       container.usePotion("Dragon Blood", 200.0);
       System.out.println("Remaining volume of  Dragon Blood: " +
       container.getRemainingVolume("Dragon Blood") + " ml");

       System.out.println("\n=== Checking Potion Availability for Invisibility
       Draught ===");
        double requiredUnicornTears = 200.0;
        double requiredDragonBlood = 150.0;
       boolean readyForInvisibilityDraught =
        container.isEnoughForPotion(requiredUnicornTears, requiredDragonBlood);

        System.out.println("\nCan prepare Invisibility Draught?");
        if (readyForInvisibilityDraught) {
            System.out.println("Yes, we have enough Unicorn Tears and Dragon
            Blood!");
        } else {
            System.out.println("No, we do not have enough ingredients to prepare
            the Invisibility Draught.");
        }

        System.out.println("\nFinal state of the potion container:");
        container.printPotions();
    }
}
```

Sample Output :

```
Adding potions to the container...
200.0 ml of Unicorn Tears added to the container.
150.0 ml of Dragon Blood added to the container.
Potion container successfully initialized.

=== Using Potions ===
50.0 ml of Unicorn Tears used.
Remaining volume of Unicorn Tears: 150.0 ml
30.0 ml of Dragon Blood used.
Remaining volume of Dragon Blood: 120.0 ml

Attempting to use more Dragon Blood than available...
Not enough Dragon Blood available.
Remaining volume of Dragon Blood: 120.0 ml

=== Checking Potion Availability for Invisibility Draught ===

Can prepare Invisibility Draught?
No, we do not have enough ingredients to prepare the Invisibility Draught.

Final state of the potion container:

--- Potion Inventory ---
Unicorn Tears: 150.00 ml
Dragon Blood: 120.00 ml
```

Question 2 : Zoo Creatures

Problem Statement

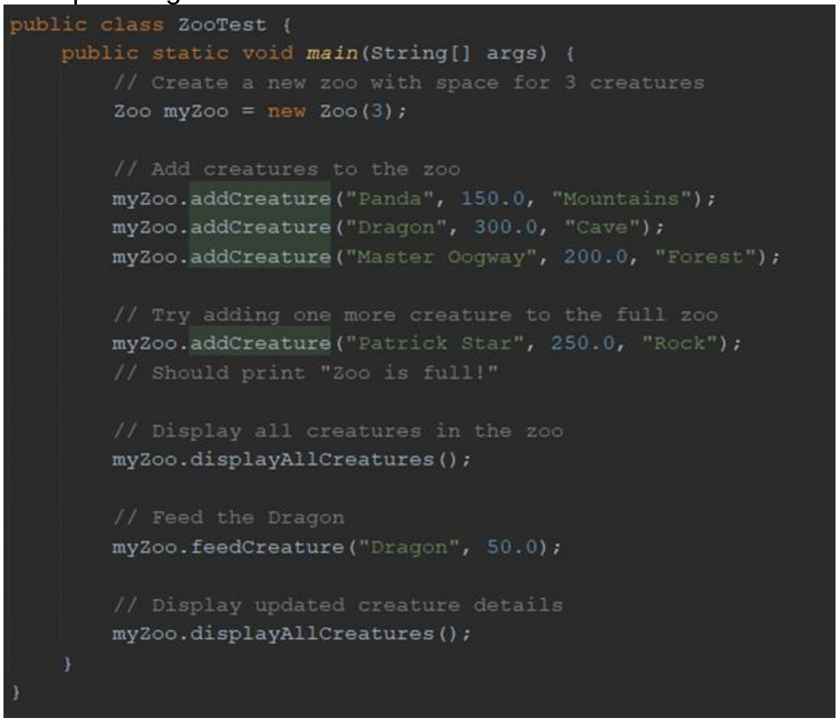
Define a class `Creature` with attributes `species (String), magicPower (double), and habitat (String)`. Implement methods to:

a) `feed(double foodAmount)` : Increases the creature's magic power.
b) `displayInfo()` : Displays the creature's details.

Create a class `Zoo` that manages a collection of creatures. Allow users to :

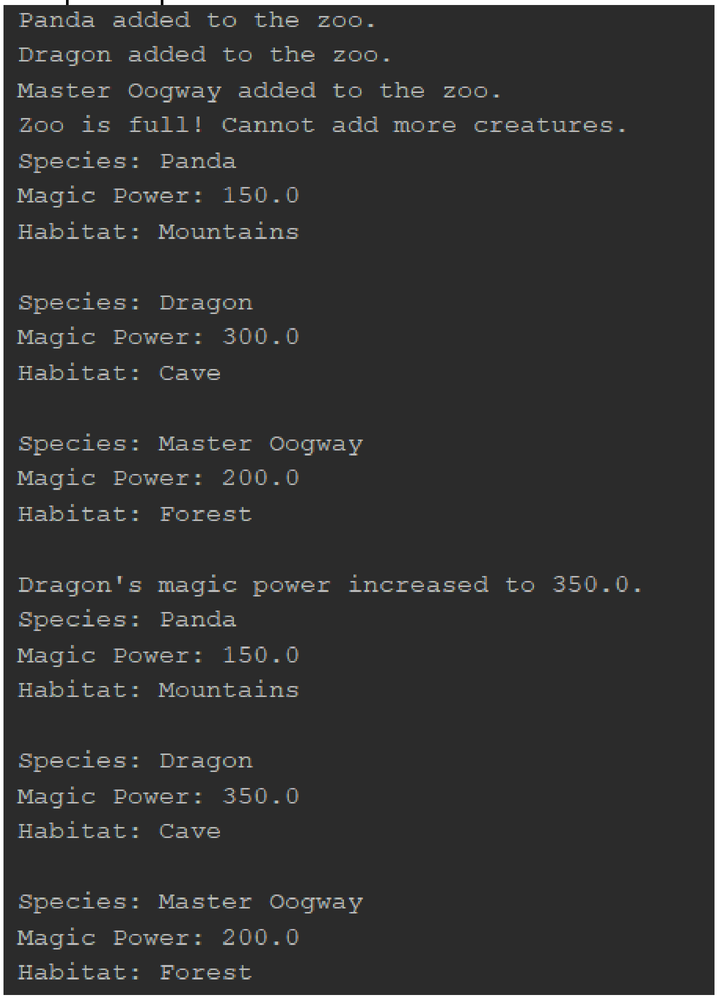
a) add new creatures
b) feed them
c) display information about all the creatures in the zoo.

Sample Program



```java
public class ZooTest {
    public static void main(String[] args) {
        // Create a new zoo with space for 3 creatures
        Zoo myZoo = new Zoo(3);

        // Add creatures to the zoo
        myZoo.addCreature("Panda", 150.0, "Mountains");
        myZoo.addCreature("Dragon", 300.0, "Cave");
        myZoo.addCreature("Master Oogway", 200.0, "Forest");

        // Try adding one more creature to the full zoo
        myZoo.addCreature("Patrick Star", 250.0, "Rock");
        // Should print "Zoo is full!"

        // Display all creatures in the zoo
        myZoo.displayAllCreatures();

        // Feed the Dragon
        myZoo.feedCreature("Dragon", 50.0);

        // Display updated creature details
        myZoo.displayAllCreatures();
    }
}
```

Sample Output



```
Panda added to the zoo.
Dragon added to the zoo.
Master Oogway added to the zoo.
Zoo is full! Cannot add more creatures.
Species: Panda
Magic Power: 150.0
Habitat: Mountains

Species: Dragon
Magic Power: 300.0
Habitat: Cave

Species: Master Oogway
Magic Power: 200.0
Habitat: Forest

Dragon's magic power increased to 350.0.
Species: Panda
Magic Power: 150.0
Habitat: Mountains

Species: Dragon
Magic Power: 350.0
Habitat: Cave

Species: Master Oogway
Magic Power: 200.0
Habitat: Forest
```

Question 3: Enhanced Pet Adoption Centre System

Problem Statement:

The **Happy Tails Pet Adoption Centre** manages a wide variety of pets, including **dogs**, **cats**, **rabbits**, and even **exotic animals** like **parrots** and **reptiles**. The centre needs a comprehensive system to keep track of each pet's details and adoption history.

Each pet has attributes such as **`name, species, breed, age, adoption status`**, and a **`health record`** (e.g., vaccinations, medical history). The centre also wants to match pets with potential adopters based on their preferences, such as **`pet type`** and **`age range`**.

Your task is to design a program that handles the following:

1. **`Pet` Class**:
   - Attributes: `name, species, breed, age, healthRecord, isAdopted,adopterName`.
   - Methods: `adopt(String adopterName), getDetails().`

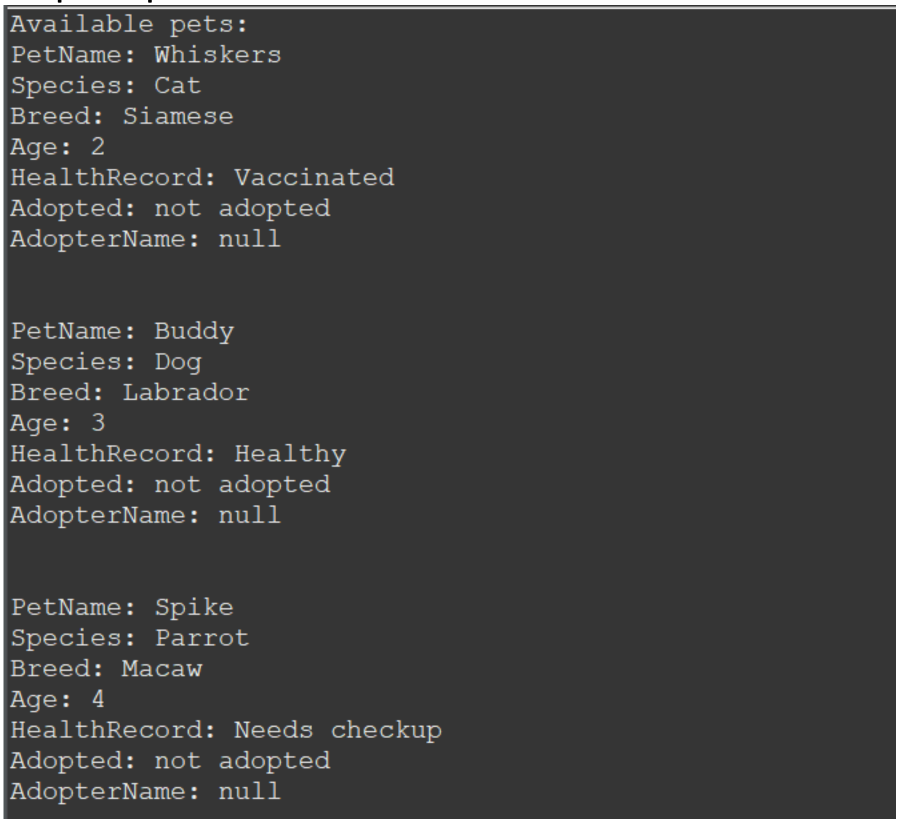
2. **`Adopter` Class**:
   - Attributes: `name, preferredSpecies, preferredAgeRange`.
   - Methods: `viewMatchingPets(PetAdoptionCentre centre).`

3. **`PetAdoptionCentre` Class**:
   - Manages all pets and adoptation.
   - Methods:
     - `addPet(Pet pet)`: Adds a new pet to the system.
     - `adoptPet(Pet pet)`: Handles the adoption process.
     - `viewAvailablePets()`: Displays pets available for adoption in ascending order of age.
     - `getPetByName(String petName)`: returns a pet using the name

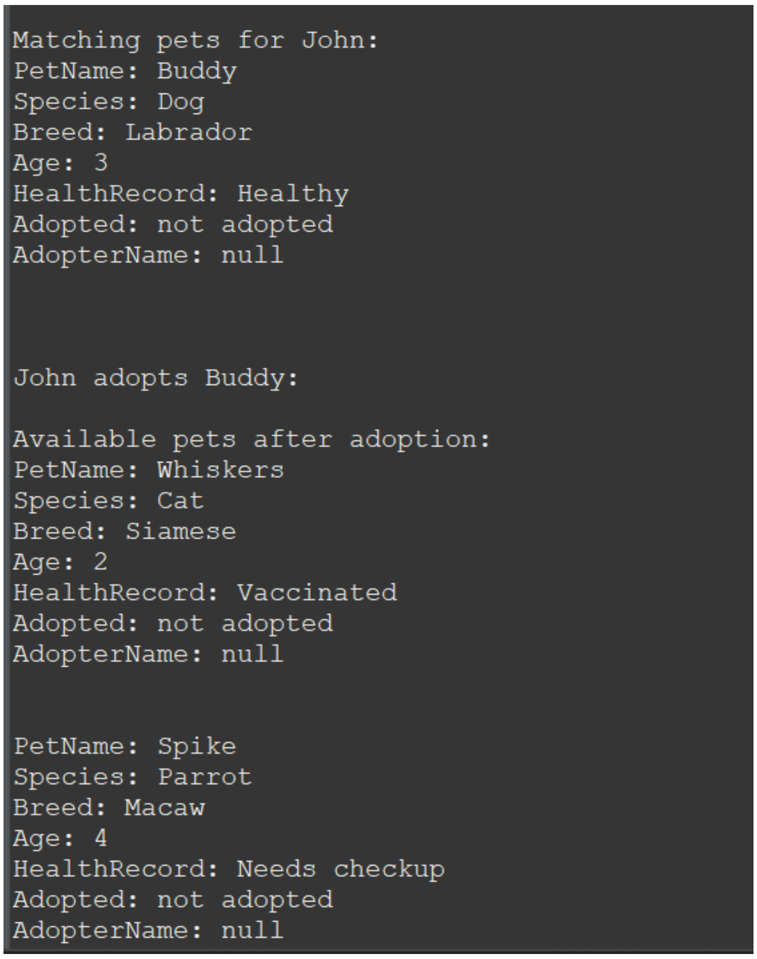
Test Program

```
public class PetAdoptionTest {
    public static void main(String[] args) {
        public static void main(String[] args) {
        PetAdoptionCentre centre = new PetAdoptionCentre();

        // Adding pets to the centre
        centre.addPet(new Pet("Buddy", "Dog", "Labrador", 3, "Healthy"));
        centre.addPet(new Pet("Whiskers", "Cat", "Siamese", 2, "Vaccinated"));

        // Viewing available pets
        System.out.println("Available pets:");
        centre.viewAvailablePets();

        // Adopter matching and adoption process
        Adopter adopter1 = new Adopter("John", "Dog", "1-4");
        System.out.println("\nMatching pets for John:");
        adopter1.viewMatchingPets(centre);

        System.out.println("\nJohn adopts Buddy:");
        centre.adoptPet(centre.getPetByName("Buddy"),adopter1);
        System.out.println("\nAvailable pets after adoption:");
        centre.viewAvailablePets();
    }
    }
}
```

Sample Output:



```
Available pets:
PetName: Whiskers
Species: Cat
Breed: Siamese
Age: 2
HealthRecord: Vaccinated
Adopted: not adopted
AdopterName: null


PetName: Buddy
Species: Dog
Breed: Labrador
Age: 3
HealthRecord: Healthy
Adopted: not adopted
AdopterName: null


PetName: Spike
Species: Parrot
Breed: Macaw
Age: 4
HealthRecord: Needs checkup
Adopted: not adopted
AdopterName: null
```



```
Matching pets for John:
PetName: Buddy
Species: Dog
Breed: Labrador
Age: 3
HealthRecord: Healthy
Adopted: not adopted
AdopterName: null


John adopts Buddy:

Available pets after adoption:
PetName: Whiskers
Species: Cat
Breed: Siamese
Age: 2
HealthRecord: Vaccinated
Adopted: not adopted
AdopterName: null


PetName: Spike
Species: Parrot
Breed: Macaw
Age: 4
HealthRecord: Needs checkup
Adopted: not adopted
AdopterName: null
```

Question 4: Museum Exhibit Management System

Problem Statement:

The **Metropolis Museum of Art** has a vast collection of exhibits, including **paintings**, **sculptures**, and **ancient artifacts**. The museum needs a system to manage these exhibits, track their details, and allow visitors to search and explore the collection. Each exhibit has attributes such as **`title`**, **`artist`**, **`year`**, **`type`** (e.g., painting, sculpture), and **`description`**.

To simplify data management, the museum stores information about its exhibits in a **text file** called *`exhibits.txt`*. The system must be able to **read** this file and create exhibit objects based on the data. Visitors can then **search** for exhibits by artist, type, or year, and view detailed information about each exhibit.

Requirements

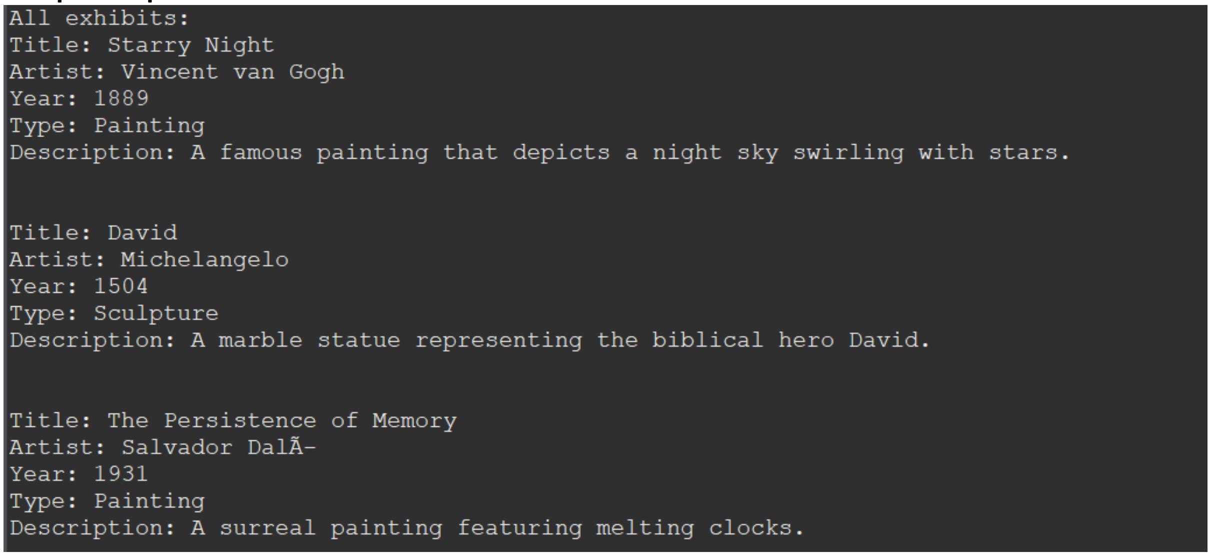
1. **`Exhibit`** **Class**:
   - Attributes: `title, artist, year, type, and description.`
   - Methods: `getDetails().`
2. **`Museum`** **Class**:
   - Reads data from the *`exhibits.txt`* file and creates Exhibit objects.
   - Stores all exhibits in a collection (e.g., ArrayList).
   - Methods:

- `loadExhibits(String filename)`: Reads exhibit details from the file and populates the collection.
- `searchExhibitsByArtist(String artist)`: Searches and displays exhibits by a specific artist.
- `searchExhibitsByType(String type)`: Searches and displays exhibits of a specific type (e.g., paintings).
- `searchExhibitsByYear(int year)`: Searches and displays exhibits from a specific year.
- `viewAllExhibits()`: Displays all exhibits.

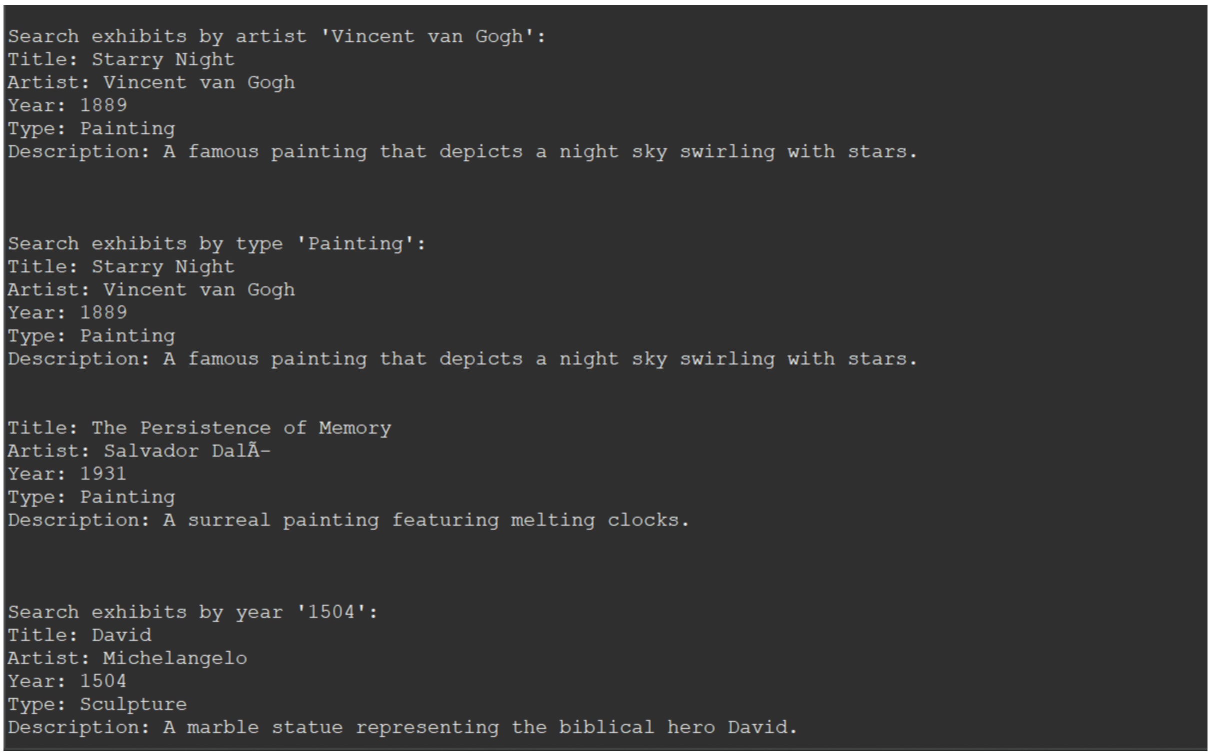
Test Program:

```java
public class Testing {

       public static void main(String[] args) {
          Museum museum = new Museum();
          museum.loadExhibits("Exhibits.txt");

          System.out.println("All exhibits:");
          museum.viewAllExhibits();

          System.out.println("\nSearch exhibits by artist 'Vincent van Gogh':");
          museum.searchExhibitsByArtist("Vincent van Gogh");

          System.out.println("\nSearch exhibits by type 'Painting':");
          museum.searchExhibitsByType("Painting");

          System.out.println("\nSearch exhibits by year '1504':");
          museum.searchExhibitsByYear(1504);

       }
}
```

Sample Output:



```
All exhibits:
Title: Starry Night
Artist: Vincent van Gogh
Year: 1889
Type: Painting
Description: A famous painting that depicts a night sky swirling with stars.


Title: David
Artist: Michelangelo
Year: 1504
Type: Sculpture
Description: A marble statue representing the biblical hero David.


Title: The Persistence of Memory
Artist: Salvador DalÃ-
Year: 1931
Type: Painting
Description: A surreal painting featuring melting clocks.
```



```
Search exhibits by artist 'Vincent van Gogh':
Title: Starry Night
Artist: Vincent van Gogh
Year: 1889
Type: Painting
Description: A famous painting that depicts a night sky swirling with stars.



Search exhibits by type 'Painting':
Title: Starry Night
Artist: Vincent van Gogh
Year: 1889
Type: Painting
Description: A famous painting that depicts a night sky swirling with stars.


Title: The Persistence of Memory
Artist: Salvador DalÃ-
Year: 1931
Type: Painting
Description: A surreal painting featuring melting clocks.



Search exhibits by year '1504':
Title: David
Artist: Michelangelo
Year: 1504
Type: Sculpture
Description: A marble statue representing the biblical hero David.
```

Question 5: Banking System

Problem Statement:

You have been hired to create a simple banking system. Implement the following classes to manage customers and their bank accounts.

1. `BankAccount` Class:

- **Attributes**:
  - `String accountNumber`: A unique account number for the customer.
  - `String accountHolderName`: The name of the account holder.
  - `double balance`: The current balance of the bank account (default is 0.0).

- **Methods**:
  - Constructor that initializes the account with an `account number, account holder's name,` and `an initial deposit amount.`
  - `void deposit(double amount)`: Adds the specified amount to the account balance.
  - boolean withdraw(double amount): Deducts the specified amount from the balance if there are sufficient funds; returns true if successful and false if there are insufficient funds.
  - Accessors (`getAccountNumber(), getAccountHolderName(), getBalance()`) for all attributes.
  - Mutator for `balance` (should be private, and only updated through deposit/withdraw methods).

2. `Customer` Class:

- **Attributes**:
  - `String name`: The name of the customer.
  - `String customerId`: A unique identifier for the customer.
  - `ArrayList<BankAccount> accounts`: A list to store the customer's bank accounts.
- **Methods**:
  - Constructor that initializes the `customer's name and customer ID.`
  - `void addAccount(BankAccount account)`: Adds a new BankAccount to the customer's list of accounts.
  - `BankAccount getAccount(String accountNumber)`: Finds and returns the account with the specified account number, or null if the account does not exist.
  - Accessors for `name` and `customerId`.
  - `void displayAccounts()`: Prints out details of all the customer's accounts (account number and balance).

3. `Bank` Class:

- **Attributes**:
  - `String bankName`: The name of the bank.
  - `ArrayList<Customer> customers`: A list to store the bank's customers.
- **Methods**:
  - Constructor that initializes the bank with a name.
  - `void addCustomer(Customer customer)`: Adds a new customer to the bank's list of customers.
  - `Customer getCustomer(String customerId)`: Finds and returns the customer with the specified ID, or null if the customer does not exist.
  - `void displayAllCustomers()`: Prints out the details of all customers (name and customer ID).

Functional Requirements:

1. **Create a `Bank` System**:
   - The bank should have a name and be able to add customers and bank accounts for each customer.
2. **Create `Customer` and `BankAccount`**:
   - A customer can have multiple bank accounts (e.g., a savings account and a checking account).
3. **Basic Transactions**:
   - Allow customers to deposit and withdraw money from their accounts. Ensure that withdrawals do not exceed the available balance.
4. **Display Information**:
   - The system should be able to display the list of all customers and their associated accounts.

Sample Output:

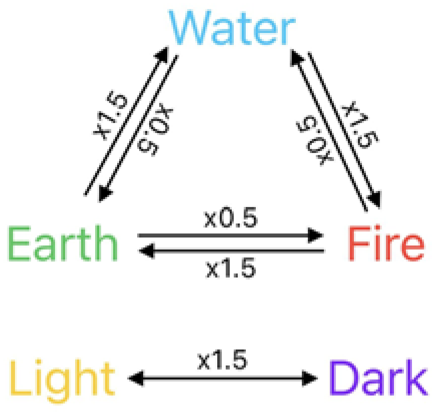
```
Welcome to SimpleBank!
Creating a new customer: John Doe (ID: C001)
Adding a savings account for John Doe with account number
A1001 and an initial deposit of $500.
Depositing $200 into account A1001...
New balance: $700.0
Withdrawing $100 from account A1001...
New balance: $600.0
Displaying all accounts for customer John Doe:
Account Number: A1001, Balance: $600.0
Displaying all customers of SimpleBank:
Customer: John Doe, ID: C001
```

Question 6: Tower of Saviors

Problem Statement:

Tower of Saviors is a combination of a match 3 game and an RPG and features characters from various mythologies and cultures. It takes place in a fantasy world where players assume the role of "Summoners" and collect various monster cards with various mythological backgrounds and rarities. These heroes utilize one of 5 elements, which are Water, Fire, Earth, Light and Dark, to help the "Summoners" defeat the enemies. However, each of the enemies have their own strengths and weaknesses against elements, which means some level of strategy is necessary.

When a hero is attacking an enemy, its attack will get buffed or debuffed based on its element and the enemy's element. The dominance relationships of elements are as below:



A class `Hero` is created to store information about each hero, including their `name`, `element`, `health points (HP)` and `attack`.

- Method `calculateDamage(Villain enemy, int rsMultiplier)` is created to calculate and return the damage dealt to the enemy.

- Override `toString()` method to return information of the hero.

Another class `Villain` is created to store information about the enemies, including their `name`, `element`, `maximum health points (maxHp), hp, attack, defense, initial cool down(initialCd) and currentCd.`

- Create a method `getDamaged(double damage)` to calculate remaining hp of the enemy after being attacked. If the remaining hp < 0, set the remaining hp to 0.
- Create a method `resetHp`() to reset the villain's hp.
- Create a method `decreaseCd`() to decrease the villain's currentCd at the end of each round.
- Create a method `resetCd`() to reset the villain's currentCd after attacking.
- Override `toString`() method to return information of the villain.

A class `Team` is created to manage the heroes we have such that we may better organize and evaluate our heroes based on their combat abilities against enemies. It should have attributes `deck`, `heroList` and `hp`.

- Create a method `formTeam()` which will clear the `heroList` and randomly choose **4** heroes from the deck and add them into the `heroList`. Then, calculate the team's hp by summing hp of the heroes chosen.
- Create a method `getDamaged(double damage)` to calculate remaining hp of the team after being attacked. If the remaining hp < 0, set the remaining hp to 0.
- Create a method `resetTeamHp()` to reset the team's hp.
- Override `toString`() method to return the team's hp, and attributes of each hero in heroList.

A class `Game` is created to control the flow of the game.

- Create a method `battle(Team team, Villain enemy).`
  Before the battle starts, **reset** hp of the team and the enemy, and CD of the enemy.
  Then, in each round,
  - **3** runestones, which may be Water, Fire, Earth, Light or Dark, are randomly chosen to be dissolved.
  - Hero(es) with corresponding element to the runestones dissolved will deal damage to the enemy. The damage formula is given as:

*Damage = Attack * Dominance multiplier * Runestone multiplier - Defense*

    Note:
    1. *Runestone multiplier* refers to number of runestones of corresponding element dissolved
    2. If *Damage* < 1, deal **1** damage to the enemy
  - If the `currentCd` of the enemy = **1** and remaining hp > 0, it will deal damage to the team and reset its current CD. Otherwise, CD of the enemy is decreased by **1** at the end of the round.
  - Show the flow of attack of each hero and enemy.
  - At the end of the round, display the remaining hp for each team and enemy, and the CD of the enemy.

If the remaining hp of the enemy <= 0, the team wins the battle. If the remaining hp of the team <= 0, the team loses the battle. Print the result of the battle.

Sample Output 1

```

Team's HP: 206.0

Hero 1
Name: Nathaniel
Element: Light
HP: 37.0
Attack: 24.0

Hero 2
Name: Skuld
Element: Fire
HP: 66.0
Attack: 16.0

Hero 3
Name: Molly
Element: Water
HP: 45.0
Attack: 20.0

Hero 4
Name: Poseidon
Element: Water
HP: 58.0
Attack: 19.0

Round 1
Enemy's current CD: 1
Runestones dissolved:
 - Light
 - Light
 - Fire

Skuld dealt 1.0 damage to Giemsa
Nathaniel dealt 33.0 damage to Giemsa

Team's remaining HP: 206.0
Enemy's remaining HP: 66.0

Round 2
Enemy's current CD: 2
Runestones dissolved:
 - Dark
 - Dark
 - Earth

No hero attacked in this round
Giemsa dealt 150.0 damage to the team

Team's remaining HP: 56.0
Enemy's remaining HP: 66.0

Round 3
```

```
Enemy's current CD: 1
Runestones dissolved:
 - Light
 - Water
 - Light

Molly dealt 5.0 damage to Giemsa
Poseidon dealt 4.0 damage to Giemsa
Nathaniel dealt 33.0 damage to Giemsa

Team's remaining HP: 56.0
Enemy's remaining HP: 24.0

Round 4
Enemy's current CD: 2
Runestones dissolved:
 - Dark
 - Earth
 - Water

Molly dealt 5.0 damage to Giemsa
Poseidon dealt 4.0 damage to Giemsa
Giemsa dealt 150.0 damage to the team

Team's remaining HP: 0.0
Enemy's remaining HP: 15.0

The team lose.
```

Sample Output 2

```


Team's HP: 207.0

Hero 1
Name: Endor
Element: Dark
HP: 43.0
Attack: 21.0

Hero 2
Name: Nathaniel
Element: Light
HP: 37.0
Attack: 24.0
Hero 3
Name: Duncan
Element: Earth
HP: 53.0
Attack: 16.0

Hero 4
Name: Verthandi
```

```
Element: Earth
HP: 74.0
Attack: 13.0

Round 1
Enemy's current CD: 1
Runestones dissolved:
- Earth
- Light
- Water

Duncan dealt 9.0 damage to Giemsa
Verthandi dealt 4.5 damage to Giemsa
Nathaniel dealt 9.0 damage to Giemsa

Team's remaining HP: 207.0
Enemy's remaining HP: 77.5

Round 2
Enemy's current CD: 2
Runestones dissolved:
- Light
- Earth
- Light

Duncan dealt 9.0 damage to Giemsa
Verthandi dealt 4.5 damage to Giemsa
Nathaniel dealt 33.0 damage to Giemsa
Giemsa dealt 150.0 damage to the team

Team's remaining HP: 57.0
Enemy's remaining HP: 31.0

Round 3
Enemy's current CD: 1
Runestones dissolved:
- Earth
- Earth
- Earth

Duncan dealt 57.0 damage to Giemsa
Verthandi dealt 43.5 damage to Giemsa

Team's remaining HP: 57.0
Enemy's remaining HP: 0.0

The team won!
```

Test Program

```
public class TOSTest {
    public static void main(String[] args) {
        Hero molly = new Hero("Molly", "Water", 45, 20);
        Hero sean = new Hero("Sean", "Fire", 36, 24);
        Hero duncan = new Hero("Duncan", "Earth", 53, 16);
        Hero nathaniel = new Hero("Nathaniel", "Light", 37, 24);
        Hero endor = new Hero("Endor", "Dark", 43, 21);
        Hero urd = new Hero("Urd", "Water", 65, 17);
        Hero skuld = new Hero("Skuld", "Fire", 66, 16);
        Hero verthandi = new Hero("Verthandi", "Earth", 74, 13);
        Hero idun = new Hero("Idun", "Light", 59, 19);
        Hero valkyrie = new Hero("Valkyrie", "Dark", 61, 18);
        Hero poseidon = new Hero("Poseidon", "Water", 58, 19);
        Hero hephaestus = new Hero("Hephaestus", "Fire", 51, 22);
        Hero athena = new Hero("Athena", "Earth", 61, 18);
        Hero apollo = new Hero("Apollo", "Light", 55, 16);
        Hero artemis = new Hero("Artemis", "Dark", 50, 23);

        Hero[] heroes = {molly, sean, duncan, nathaniel, endor, urd, skuld,
verthandi, idun, valkyrie, poseidon, hephaestus, athena, apollo, artemis};

        Villain giemsa = new Villain("Giemsa", "Water", 100, 150, 15, 2);
        Villain diablo = new Villain("Diablo", "Fire", 120, 163, 13, 3);
        Villain nidhogg = new Villain("Nidhogg", "Earth", 130, 189, 11, 4);
        Villain lucifer = new Villain("Lucifer", "Light", 110, 207, 17, 3);
        Villain odin = new Villain("Odin", "Dark", 135, 196, 14, 5);

        Team team = new Team(heroes);
        Game game = new Game();

        team.formTeam();
        System.out.println(team);

        game.battle(team, giemsa);
    }
}
```