Technical Discussion for Xebia

Java, Python, JavaScript

Java & Python - Object Oriented Programming Languages JavaScript - Object Based Programming Language

- Even though Python is said to be Object Oriented, it is not purely Object Oriented. It is a multi-paradigm programming language.
- JavaScript is said to be Object Based because it does not support classes (ES5). It is prototype based.
- But with ES6, JavaScript supports classes. So, it is also Object Oriented Programming Language.
- Java is purely Object Oriented Programming Language.
- Java is a strongly typed language. Python and JavaScript are dynamically typed languages.

Object Oriented Programming Concepts

Context:

- Object Oriented Programming is a programming paradigm that is based on the concept of "objects", which can contain data, in the form of fields (often known as attributes or properties), and code, in the form of procedures (often known as methods).
- An Object is a building block of Object Oriented Programming. It relates to real-world entities. Each object can contain data in the form of fields and code in the form of procedures (methods).

Why Object Oriented Programming?

- Object Oriented Programming helps in organizing the code and makes it reusable.
- It helps in modeling real-world entities.
- It helps in maintaining the code and makes it easier to understand.

Concepts:

1. Class

- A class is a blueprint for creating objects.
- It is a template for objects.
- Collection of objects of similar type.
- It is a user-defined data type.
- It defines the properties and behavior of objects.
- It contains fields (attributes) and methods (procedures).

2. Object

- An object is an instance of a class.
- It is a real-world entity.
- It has state and behavior.

3. Data Encapsulation

- Data Encapsulation is the mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit.
- Example: Class

4. Inheritance

- Inheritance is a mechanism in which one class acquires the properties and behavior of another class.
- It helps in reusability.
- It helps in modeling real-world entities.
- Example: Parent Class and Child Class

Constructors:

- Constructors are special methods that are used to initialize objects.

```
// Creating a class Account

// Data Encapsulation: Wrapping of data and methods.
class Account {
    // Data members
    int accountNumber;
    double balance;

    // Constructors
    // Default Constructor are available by default implicitly
    // Constructor Overloading
    Account() {
        accountNumber = 123456;
        balance = 1000.00;
    }

    // Parameterized Constructor
    Account(int accountNumber) {
        this.accountNumber = accountNumber;
        balance = 1000.00;
    }

    Account(int accountNumber, double balance) {
        this.accountNumber = accountNumber;
        this.balance = balance;
    }

    // Methods

    // define a main method
    public static void main(String[] args) {
        // Create an object of Account class
        Account acc1 = new Account(); // instantiation of Account class
        System.out.println("Account Number: " + acc1.accountNumber);
        System.out.println("Balance: " + acc1.balance);
```

```
        Account acc2 = new Account(123457);
        System.out.println("Account Number: " + acc2.accountNumber);
        System.out.println("Balance: " + acc2.balance);

        Account acc3 = new Account(123458, 2000.00);
        System.out.println("Account Number: " + acc3.accountNumber);
        System.out.println("Balance: " + acc3.balance);
    }
}
```

```
class Account {
  // constructor overloading
  constructor(accountNumber = 111, balance = 0) {
    if (accountNumber === undefined && balance === undefined) {
      this.accountNumber = 111;
      this.balance = 0;
    } else if (balance === undefined) {
      this.accountNumber = accountNumber;
      this.balance = 0;
    } else {
      this.accountNumber = accountNumber;
      this.balance = balance;
    }
  }
}

let acc1 = new Account(123, 500);
console.log("Account Number: " + acc1.accountNumber);
console.log("Balance: " + acc1.balance);

let acc2 = new Account(123);
console.log("Account Number: " + acc2.accountNumber);
console.log("Balance: " + acc2.balance);

let acc3 = new Account();
console.log("Account Number: " + acc3.accountNumber);
console.log("Balance: " + acc3.balance);
```

```
class Account:
    # Constructor Overloading
    def __init__(self, accountNumber=None, balance=None):
        if accountNumber is None and balance is None:
            self.accountNumber = 0
            self.balance = 0
        elif accountNumber is not None and balance is None:
            self.accountNumber = accountNumber
            self.balance = 0
        else:
```

```
            self.accountNumber = accountNumber
            self.balance = balance


acc1 = Account(123, 1000)
print("Account Number:", acc1.accountNumber)
print("Balance:", acc1.balance)

acc2 = Account(456)
print("Account Number:", acc2.accountNumber)
print("Balance:", acc2.balance)

acc3 = Account()
print("Account Number:", acc3.accountNumber)
print("Balance:", acc3.balance)
```

Types of Constructors:

- Default Constructor: A constructor that does not have any parameters.
- Parameterized Constructor: A constructor that has parameters.

Constructor Overloading:

- Constructor Overloading is a concept in which a class can have more than one constructor.
- The constructors should have different parameters.
- They are supported in Java but not in Python and JavaScript.

Methods:

- Methods are functions defined in a class.
- They are used to define the behavior of objects.
- The difference between a function and a method is that a method is associated with an object.

Method Overloading:

- Method Overloading is a concept in which a class can have more than one method with the same name but different parameters.
- It is supported in Java but not in Python and JavaScript.

```
class Account {
  // constructor overloading
  constructor(accountNumber = 111, balance = 0) {
    if (accountNumber === undefined && balance === undefined) {
      this.accountNumber = 111;
      this.balance = 0;
    } else if (balance === undefined) {
      this.accountNumber = accountNumber;
      this.balance = 0;
    } else {
      this.accountNumber = accountNumber;
```

```
      this.balance = balance;
    }
  }

  // this -> current object
  deposit(amount) {
    this.balance += amount;
  }

  withdraw(amount) {
    if (this.balance >= amount) {
      this.balance -= amount;
    } else {
      console.log("Insufficient Balance");
    }
  }
}

let acc1 = new Account(123, 500);
acc1.deposit(500);
acc1.deposit(500);
console.log("Account Number: " + acc1.accountNumber);
console.log("Balance: " + acc1.balance);

let acc2 = new Account(124);
acc2.deposit(1000);
acc2.withdraw(500);
console.log("Account Number: " + acc2.accountNumber);
console.log("Balance: " + acc2.balance);
acc2.withdraw(1000);
```

```
class Account:
    # Constructor Overloading
    def __init__(self, accountNumber=None, balance=None):
        if accountNumber is None and balance is None:
            self.accountNumber = 0
            self.balance = 0
        elif accountNumber is not None and balance is None:
            self.accountNumber = accountNumber
            self.balance = 0
        else:
            self.accountNumber = accountNumber
            self.balance = balance

    def deposit(self, amount):
        self.balance += amount

    def withdraw(self, amount=25):
        if self.balance >= amount:
            self.balance -= amount
        else:
```

```
            print("Insufficient Balance")


acc1 = Account(123, 1000)
print("Account Number:", acc1.accountNumber)
print("Balance:", acc1.balance)

acc2 = Account(456)
print("Account Number:", acc2.accountNumber)
print("Balance:", acc2.balance)

acc2.deposit(500)
print("Balance:", acc2.balance)

acc2.withdraw(200)
print("Balance:", acc2.balance)

acc2.withdraw()  # method overloading
print("Balance:", acc2.balance)
```

```
// Creating a class Account

// Data Encapsulation: Wrapping of data and methods.
class Account {
    // Data members
    int accountNumber;
    double balance;

    // Constructors
    // Default Constructor are available by default implicitly
    // Constructor Overloading
    Account() {
        accountNumber = 123456;
        balance = 1000.00;
    }

    // Parameterized Constructor
    Account(int accountNumber) {
        this.accountNumber = accountNumber;
        balance = 1000.00;
    }

    Account(int accountNumber, double balance) {
        this.accountNumber = accountNumber;
        this.balance = balance;
    }

    // Methods
    void deposit(double amount) {
        this.balance += amount;
```

```
    }

    void withdraw(double amount) {
        if (balance >= amount) {
            balance -= amount;
        } else {
            System.out.println("Insufficient Balance");
        }
    }

    void withdraw() {
        balance -= 25;
    }

    // define a main method
    public static void main(String[] args) {
        // Create an object of Account class
        Account acc1 = new Account(); // instantiation of Account class
        System.out.println("Account Number: " + acc1.accountNumber);
        System.out.println("Balance: " + acc1.balance);

        Account acc2 = new Account(123457);
        System.out.println("Account Number: " + acc2.accountNumber);
        System.out.println("Balance: " + acc2.balance);
        acc2.deposit(500);
        System.out.println("Balance: " + acc2.balance);

        acc2.withdraw(1000);
        System.out.println("Balance: " + acc2.balance);

        // method overloading
        acc2.withdraw();
        System.out.println("Balance: " + acc2.balance);

    }
}
```

5. Polymorphism

- Polymorphism is a concept in which an object can take many forms.
- It is a combination of two words: poly + morphs.
- Poly means many and morphs means forms.
- Examples: Method Overloading, Method Overriding, Constructor Overloading

6. Inheritance

- Inheritance is a mechanism in which one class acquires the properties and behavior of another class.

Types of Inheritance:

- Single Inheritance: A class inherits from only one class.
- Multilevel Inheritance: A class inherits from another class, which in turn inherits from another class.

- Hierarchical Inheritance: Multiple classes inherit from a single class.
- Multiple Inheritance: A class inherits from multiple classes. [It is not supported in Java using classes. But we can achieve it using interfaces. It is supported in Python and JavaScript using classes.]
- Hybrid Inheritance: A combination of two or more types of inheritance.

super: It is a keyword in Java, Python, and JavaScript that is used to refer to the parent class.

```python
class A:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def sum(self):
        return self.a + self.b


class B(A):
    def __init__(self, a, b, c):
        # call the parent class constructor
        super().__init__(a, b)
        self.c = c

    # method overriding
    def sum(self):
        return self.a + self.b + self.c


obj = B(1, 2, 3)

print(obj.a, obj.b, obj.c)
print(obj.sum())
```

```javascript
class A {
  constructor(a, b) {
    this.a = a;
    this.b = b;
  }

  sum() {
    return this.a + this.b;
  }
}

class B extends A {
  constructor(a, b, c) {
    // calling the parent class constructor
    super(a, b);
    this.c = c;
  }
```

```
  sum() {
    // calling the parent class method
    return super.sum() + this.c;
  }
}

const obj = new B(1, 2, 3);

console.log(obj.sum()); // 3
```

```
class A {
    int a, b;

    A(int a, int b) {
        this.a = a;
        this.b = b;
    }

    int sum() {
        return a + b;
    }
}

class B extends A {
    int c;

    B(int a, int b, int c) {
        super(a, b);
        this.c = c;
    }

    // method overriding
    int sum() {
        return super.sum() + c;
    }

    public static void main(String[] args) {
        B obj = new B(1, 2, 3);
        System.out.println(obj.sum());
    }
}
```