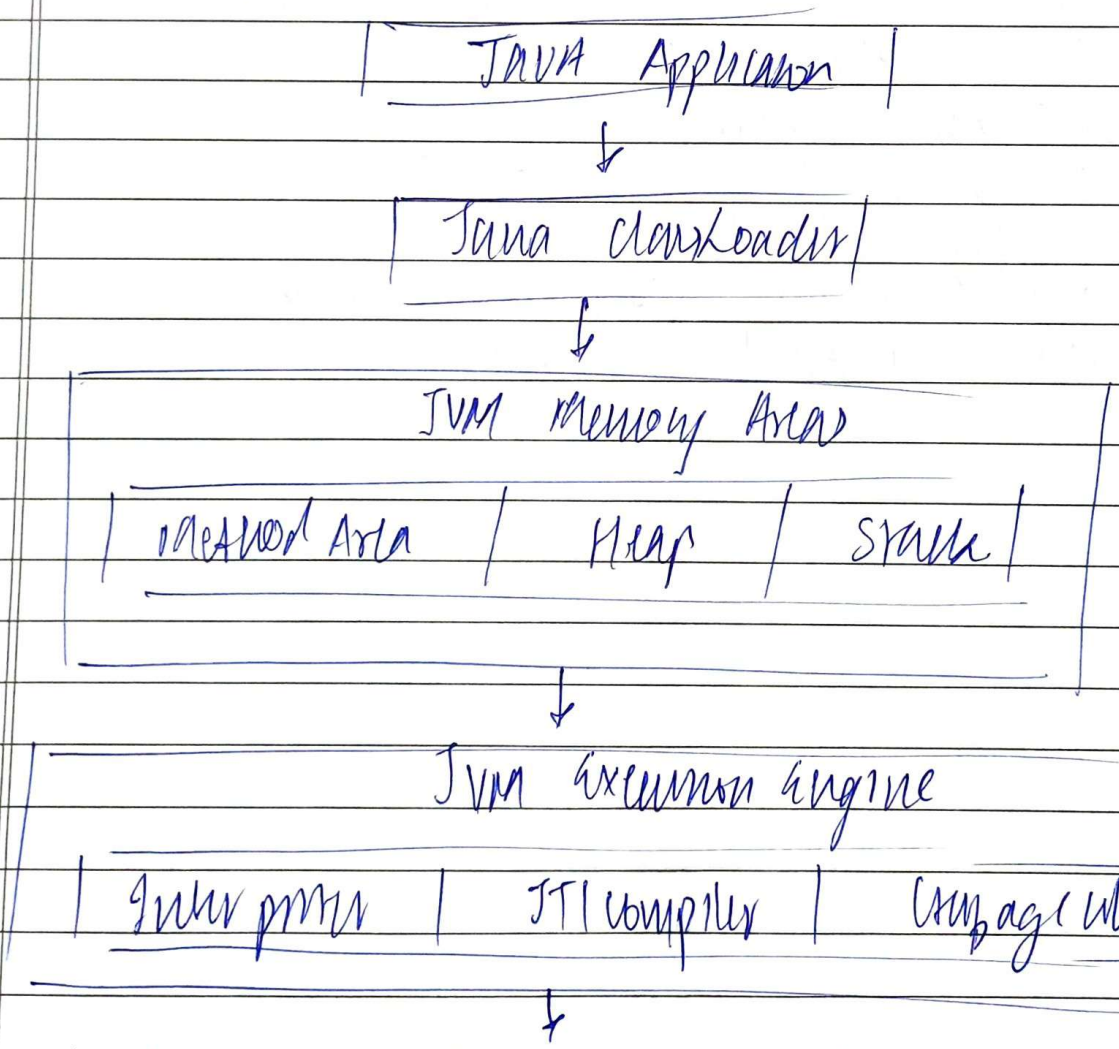
JAVA ASSIGNMENT

Aayan Srivastava, A073
2261240019

Q.1. Java Virtual Machine (JVM)

- It is a core component of the java platform that provides an execution env for java bytecode.
- It is responsible for converting the bytecode into machine-specific instructions, making java programs run on any platform without modification.



- JVM performs several critical functions:-
  - Loading the bytecode
  - Verifying the bytecode
  - Executing the bytecode.
  - Managing Memory via garbage
  - Providing runtime service

- Components:-
  1) Class Loader : loads class files into JVM.
  2) Memory Areas :
     - Method Area
     - Heap
     - Native method stack.
     - Stack
     - PC Register
  3) Execution engine
     - Interpreter
     - Garbage collector.
     - JIT compiler
  4) Native method ~~Libraries~~ Interface (JNI)
  5) Native method Libraries.

Q2.

```
class SumOfCubes {
    public static void main (String [] args) {
        if (args.length == 0) {
            SOP("enter number as argument");
            return;
        }

        String number = args[0];
        int sum = 0;

        for (int i = 0; i < number.length(); i++) {
            int digit = Character.getNumericValue (
                                number.charAt(i));

            sum += Math.pow(digit, 3);

        }

        SOP("sum of digit's cube" + sum);
    }
}
```

Q3.

```
import java.util.*
public class pattern1 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();
        int num = 1;

        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= i; j++) {
                System.out.print(num);
                num++;
            }
            System.out.println();
        }
    }
}
```

```
1
2 3
4 5 6
7 8 9 10 ...
```

```
public class pattern2 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        for (int i = n; i >= 1; i--) {
            for (int j = 1; j <= i; j++) {
                System.out.print("* ");
            }
            System.out.println();
        }
    }
}
```

```
* * * *
* * *
* *
*
...
```

```
class patterns3 {
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();

        for(int i = 1; i<=n; i++){

            for (int j = i; j<n ; j++) {
                System.out.print(" ");
            }

            char ch = 'A';
            for (int j = 1; j<= i, j++) {
                System.out.print(ch);
                ch++;
            }

            ch -= 2;
            for (int j = 1; j< i; j++ ) {
                System.out.print(ch);
                ch--;
            }

            System.out.println();
        }
    }
}
```

Q4.

```
import java.util.*;

class Reverse {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();

        int[] arr = new int[n];
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }

        for (int i = n-1; i >= 0; i--) {
            System.out.print(arr[i] + " ");
        }
    }
}
```

Q-5.

```
import java.util.*;

class Employee{
    int empid;
    String empname;
    String designation;
    double salary;

    public void get_employee(){
        Scanner sc = new Scanner(System.in);
        empid = sc.nextInt();
        empname = sc.nextLine();
        designation = sc.nextLine();
        salary = sc.nextDouble();
    }

    public void show_employee(){

        System.out.println("Employee_id : " + empid +
        "Employee_name" + empname + "designation" +
        designation + "Salary is : " + salary);

    }
}
```

```
class employeeDetails {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();

        Employee[] emp = new Employee[n];

        for (int i=0; i<n; i++) {
            emp[i] = new Employee();
            emp[i].get_employee();
        }

        for (int i=0; i<n; i++) {
            emp[i].show_employee();
        }
    }
}
```

Q.6 A constructor in Java is a special method used to initialize an object. It is called when an instance of a class is created. Constructors have the same name as the class they are in, and they do not have a return type.

Types of constructors :-

1) Default constructors

A default constructor is provided by the compiler if no constructor is defined. If a constructor is explicitly defined, the default constructor will not be provided by the compiler.

```
class Car {
    String model;
    Car() {
        model = "unknown";
    }
    void display() { }
}
public static void main(String[] args) {
    Car car1 = new Car();
    car1.display();
}
}
```

2) Parameterized constructor.
This constructor accepts arguments and initialized objects programs with specific values passed by the user.

```
class Car {
    String model;

    Car(String model) {
    this.model = model;
    }

    void display() {
      System.out.println("Model: " + this.model);
    }

    public static void main(String[] args) {

    Car car1 = new Car("sedan");
    Car car2 = new Car("SUV");

    car1.display();
    car2.display();
    }

}
```

3) Copy constructor.

A copy constructor initializes an object by copying another object of the same class. Java does not provide an explicit copy constructor, but you can define one manually by passing an object as an argument.

```
class Car {
    String model;

    Car(String model) {
        model = model;
    }

    Car(Car car) {
        this.model = car.model;
    }

    void display() { }

}

public class Main {
    public static void main(String[] args) {
        Car car1 = new Car("Tesla");
        Car car2 = new Car(car1);
        car2.display();
    }
}
```

Q 2. Class BankAccount {

```
String accountNumber;
String accountHolder;
double balance;

public BankAccount (String an, String ah, double b) {
    this.accountNumber = an;
    this.accountHolder = ah;
    this.balance = b;
}

String getAccountNumber() {
    return accountNumber;
}
String getAccountHolder() {
    return accountHolder;
}
double getBalance() {
    return balance;
}

void deposit (double amount) {
    balance += amount;
}
```

}

```
void withdraw (double amount){
    if (amount >0 && balance >= amount){
        balance -= amount;
    }
}
}
```

class SavingsAccount extends BankAccount{

```
double interRate;

SavingsAcount (String accountNumber, String
    accountHolder, double balance, double
                    interRate){
super(accountNumber, accountHolder, balance);
this.interRate = interestRate;
}

void applyInterest (){
    double interest = getBalance() * (interRate/
                                        100);
    deposit(interest);
}
```

```
void withdraw(double amount){
    if( getBalance() - amount >= 100){
        super.withdraw(amount);
    }
}
}

class LimitAccount extends BankAccount{
    double overdraftLimit;

    LimitAccount(String accountNumber,
        String accountHolder, double balance,
            double overdraftLimit){
        super(accountNumber, accountHolder,
            balance);
        this.overdraftLimit = overdraftLimit;
    }

    void withdraw(double amount){
        if(getBalance() - amount >=
                -overdraftLimit){
            super.withdraw(amount);
        }
    }
}
```

```
class AccountUtils {

    public static void transfer(BankAccount
            fromAccount, BankAccount toAccount,
                double amount) {

        if (fromAccount.isSufficient(amount)) {

            if (fromAccount.getBalance() - amount >=
                    -((CurrentAccount) fromAccount).
                        overdraftLimit) {
                fromAccount.withdraw(amount);
                toAccount.deposit(amount);
            }
            else {
                System.out.println("InsufficientBalance");
            }
        }
        else {

            if (fromAccount.getBalance() >= amount) {
                fromAccount.withdraw(amount);
                toAccount.deposit(amount);
            }
            else {
                System.out.println("Insufficient balance");
            }
        }
    }
}
```

```
class Main{
    public static void main(String[] args){
        SavingsAccount savings = new SavingsAccount(
                "A1000", "Aryan", 1000, 5);
        CurrentAccount current = new CurrentAccount(
                "B1000", "Sunita", 500, 200);

        savings.deposit(500);
        savings.withdraw(300);
        savings.applyInterest();
        System.out.println("Balance: " + savings.getBalance());

        current.withdraw(600);
        System.out.println("Current Balance" + current.
                getBalance());

        AccountUtils.transfer(savings, current, 400);
        System.out.println("Savings' balance" + savings.
                getBalance());
        System.out.println("Current balance" + current.getB-
                alance());
    }
}
```

Q.8.

```
class InvalidUsernameException extends Exception {
    public InvalidUsernameException(String message) {
        super(message);
    }
}

class InvalidEmailException extends Exception {
    public InvalidEmailException(String message) {
        super(message);
    }
}

class WeakPasswordException extends Exception {
    public WeakPasswordException(String message) {
        super(message);
    }
}

class User {
    String username;
    String email;
    String password;

    User(String username, String email,
         String password) {
        this.username = username;
        this.email = email;
        this.password = password;
    }
```

```
void validateUser() throws InvalidUsernameException, InvalidPasswordException,
                            InvalidEmailException {
    validateUsername();
    validateEmail();
    validatePassword();
}

void validateUsername() throws InvalidUsernameException {
    if (username.length() < 6) {
        throw new InvalidUsernameException(
                "username must be 6 long");
    }

    if (!username.matches("[a-zA-Z0-9]")) {
        throw new InvalidUsernameException(
                "only letters, numbers");
    }
}
```

```
void validateEmail() throws InvalidEmail-
                            Exception {
    String emailRegex = "^[A-Za-z0-9+_.-]+
                         @[A-Za-z0-9.-]+$";
    Pattern pattern = Pattern.compile(emailRegex);
    Matcher matcher = pattern.matcher(email);
    if (!matcher.matches()) {
        throw new InvalidEmailException("
                Invalid email format");
    }
}

void validatePassword() throws WeakPassword-
                            Exception {
    if (password.length() < 8) {
        throw new WeakPasswordException(
                "Atleast 8 characters");
    }
    if (!password.matches(".*[A-Z].*")) {
        throw new WeakPasswordException(
                "must contain at least one
                        uppercase char");
    }
}
```

```
if (!password.matches(".*[a-z].*")) {
    throw new WeakPasswordException(
        "password must contain one
            lower case char");
}

if (!password.matches(".*\\d.*")) {
    throw new WeakPasswordException(
        "must contain one
            digit");
}

if (!password.matches(".*[!@#$%^&*].*")) {
    throw new WeakPasswordException(
        "password must contain at least
            one special character");
}
}

class UserRegistration {

    void RegisterUser(User user) {

        try {
            user.validateUser();
            System.out.println("Registered successfully");
        }
```

```
catch (InvalidUsernameException e){
    System.out.println(e.getMessage());
}
catch (InvalidEmailException e){
    System.out.println(e.getMessage());
}
catch (InvalidPasswordException e){
    System.out.println(e.getMessage());
}
}
public static void main(String[] args){
    User user = new User("Arjun", "arjun@gmail.
                com", "Arjun@123");
    UserRegistration reg = new UserRegistration();
    reg.registerUser(user);
}
}
```