JAVA LABORATORY WEEK-6

PROGRAM ON ABSTRACT CLASS AND INTERFACE

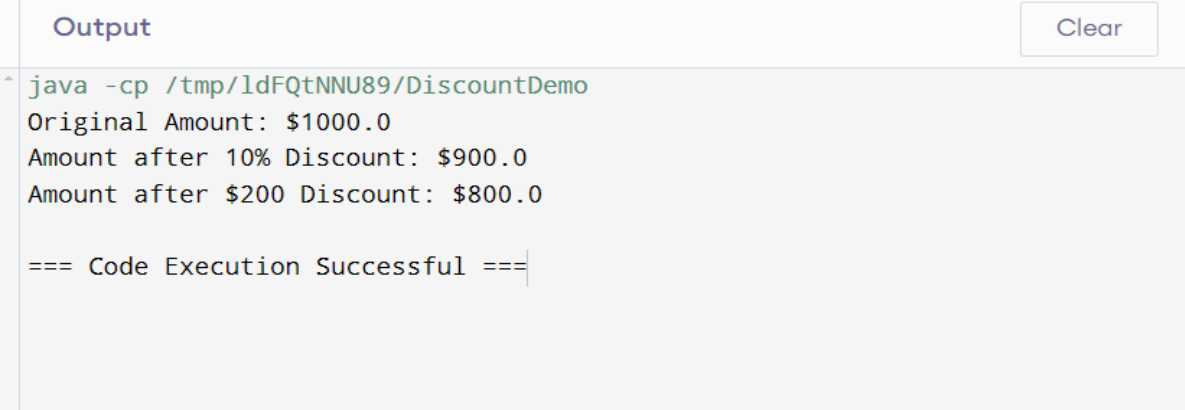
1.Create an abstract class Discount with an abstract method applyDiscount(double amount). Implement two concrete subclasses: PercentageDiscount and FixedAmountDiscount. PercentageDiscount should apply a percentage discount to a given amount, while FixedAmountDiscount should apply a fixed amount discount. Demonstrate applying discounts and showing the final amounts.

PROGRAM:

```
abstract class Discount {
    public abstract double applyDiscount(double amount);
}

class PercentageDiscount extends Discount {
    private double percentage;

    public PercentageDiscount(double percentage) {
        this.percentage = percentage;
    }

    @Override
    public double applyDiscount(double amount) {
        return amount - (amount * (percentage / 100));
    }
}

class FixedAmountDiscount extends Discount {
    private double discountAmount;

    public FixedAmountDiscount(double discountAmount) {
        this.discountAmount = discountAmount;
    }

    @Override
    public double applyDiscount(double amount) {
        return amount - discountAmount;
    }
}

public class DiscountDemo {
    public static void main(String[] args) {
```

```
        double originalAmount = 1000.00;

        Discount percentageDiscount = new PercentageDiscount(10);
        double finalAmountWithPercentageDiscount =
percentageDiscount.applyDiscount(originalAmount);
        System.out.println("Original Amount: $" + originalAmount);
        System.out.println("Amount after 10% Discount: $" +
finalAmountWithPercentageDiscount);

        Discount fixedAmountDiscount = new FixedAmountDiscount(200);
        double finalAmountWithFixedAmountDiscount =
fixedAmountDiscount.applyDiscount(originalAmount);
        System.out.println("Amount after $200 Discount: $" +
finalAmountWithFixedAmountDiscount);
    }
}
```

OUTPUT:



2.Create a banking system simulation, the BankTransaction abstract class provides a base for handling transactions with methods to get and set the balance. The DepositTransaction class increases the balance when executed and ensures the deposit amount is valid. The WithdrawalTransaction class decreases the balance and includes logic to handle insufficient funds and invalid amounts. The Main class demonstrates how these transactions are executed and how the balance changes accordingly, handling edge cases such as invalid amounts and insufficient funds.

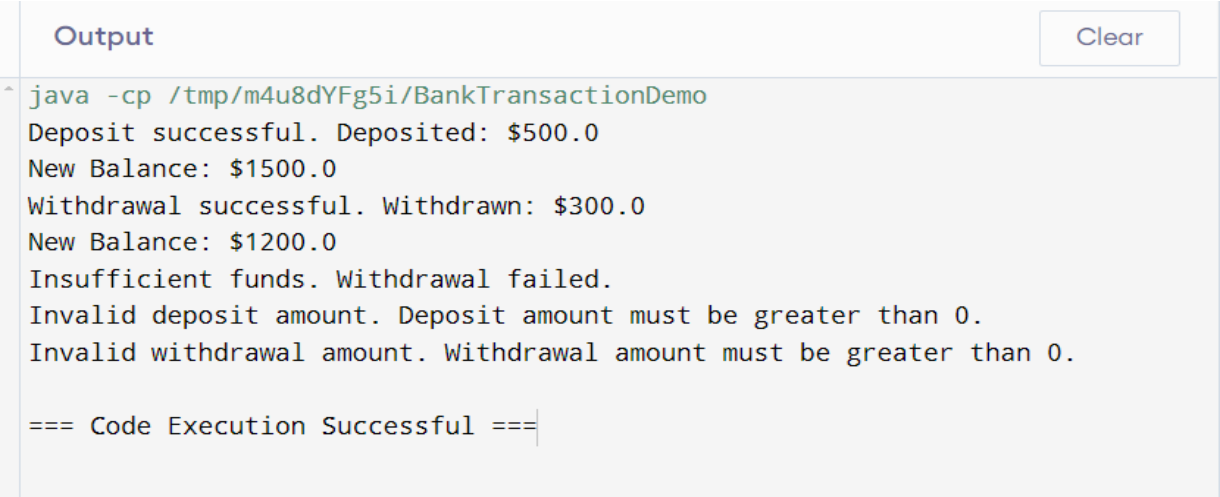
PROGRAM:

```
abstract class BankTransaction {
    protected double balance;

    public BankTransaction(double balance) {
        this.balance = balance;
    }

    public double getBalance() {
        return balance;
    }

    public void setBalance(double balance) {
        this.balance = balance;
    }

    public abstract void execute();
}

class DepositTransaction extends BankTransaction {
    private double amount;

    public DepositTransaction(double balance, double amount) {
        super(balance);
        this.amount = amount;
    }

    @Override
    public void execute() {
        if (amount <= 0) {
            System.out.println("Invalid deposit amount. Deposit amount must be greater than 0.");
        } else {
            balance += amount;
```

```
            System.out.println("Deposit successful. Deposited: $" + amount);
            System.out.println("New Balance: $" + balance);
        }
    }
}

class WithdrawalTransaction extends BankTransaction {
    private double amount;

    public WithdrawalTransaction(double balance, double amount) {
        super(balance);
        this.amount = amount;
    }

    @Override
    public void execute() {
        if (amount <= 0) {
            System.out.println("Invalid withdrawal amount. Withdrawal amount must be greater than
0.");
        } else if (amount > balance) {
            System.out.println("Insufficient funds. Withdrawal failed.");
        } else {
            balance -= amount;
            System.out.println("Withdrawal successful. Withdrawn: $" + amount);
            System.out.println("New Balance: $" + balance);
        }
    }
}

public class BankTransactionDemo {
    public static void main(String[] args) {
        double initialBalance = 1000.00;

        BankTransaction deposit = new DepositTransaction(initialBalance, 500);
        deposit.execute();
        initialBalance = deposit.getBalance();

        BankTransaction withdrawal = new WithdrawalTransaction(initialBalance, 300);
        withdrawal.execute();
        initialBalance = withdrawal.getBalance();

        BankTransaction insufficientFundsWithdrawal = new
WithdrawalTransaction(initialBalance, 1500);
        insufficientFundsWithdrawal.execute();

        BankTransaction invalidDeposit = new DepositTransaction(initialBalance, -100);
```

```
        invalidDeposit.execute();

        BankTransaction invalidWithdrawal = new WithdrawalTransaction(initialBalance, -50);
        invalidWithdrawal.execute();
    }
}
```

OUTPUT:



3.Define an interface named Payment with methods void pay(double amount), double getBalance(), and void addFunds(double amount). Create classes CreditCardPayment and PaypalPayment that implement the Payment interface, each maintaining its own balance. Finally, create a main class to demonstrate adding funds and making payment using these payment classes.

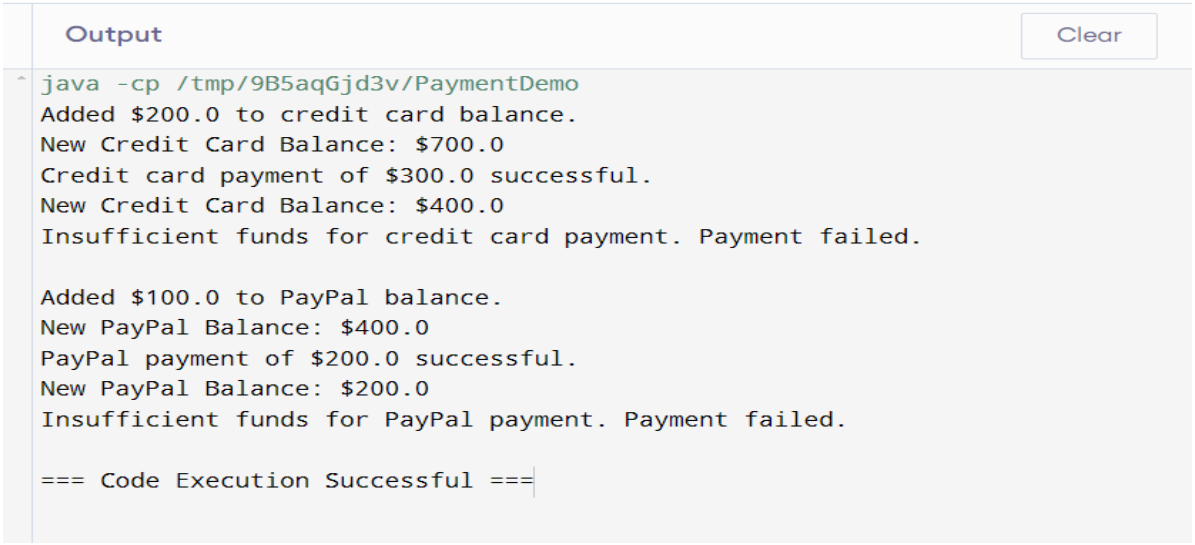
PROGRAM:

```
interface Payment {
    void pay(double amount);

    double getBalance();

    void addFunds(double amount);
}

class CreditCardPayment implements Payment {
    private double balance;

    public CreditCardPayment(double initialBalance) {
        this.balance = initialBalance;
    }

    @Override
    public void pay(double amount) {
        if (amount <= 0) {
            System.out.println("Invalid payment amount. Payment amount must be greater than 0.");
        } else if (amount > balance) {
            System.out.println("Insufficient funds for credit card payment. Payment failed.");
        } else {
            balance -= amount;
            System.out.println("Credit card payment of $" + amount + " successful.");
            System.out.println("New Credit Card Balance: $" + balance);
        }
    }

    @Override
    public double getBalance() {
        return balance;
    }

    @Override
    public void addFunds(double amount) {
        if (amount <= 0) {
            System.out.println("Invalid fund amount. Amount to add must be greater than 0.");
        } else {
```

```
            balance += amount;
            System.out.println("Added $" + amount + " to credit card balance.");
            System.out.println("New Credit Card Balance: $" + balance);
        }
    }
}

class PaypalPayment implements Payment {
    private double balance;

    public PaypalPayment(double initialBalance) {
        this.balance = initialBalance;
    }

    @Override
    public void pay(double amount) {
        if (amount <= 0) {
            System.out.println("Invalid payment amount. Payment amount must be greater than 0.");
        } else if (amount > balance) {
            System.out.println("Insufficient funds for PayPal payment. Payment failed.");
        } else {
            balance -= amount;
            System.out.println("PayPal payment of $" + amount + " successful.");
            System.out.println("New PayPal Balance: $" + balance);
        }
    }

    @Override
    public double getBalance() {
        return balance;
    }

    @Override
    public void addFunds(double amount) {
        if (amount <= 0) {
            System.out.println("Invalid fund amount. Amount to add must be greater than 0.");
        } else {
            balance += amount;
            System.out.println("Added $" + amount + " to PayPal balance.");
            System.out.println("New PayPal Balance: $" + balance);
        }
    }
}

public class PaymentDemo {
    public static void main(String[] args) {
```

```
        Payment creditCard = new CreditCardPayment(500);
        creditCard.addFunds(200);
        creditCard.pay(300);
        creditCard.pay(500);

        System.out.println();

        Payment paypal = new PaypalPayment(300);
        paypal.addFunds(100);
        paypal.pay(200);
        paypal.pay(300);
    }
}
```

OUTPUT:



4.Define an interface named Authentication with methods boolean authenticate(String userId, String credentials) and a default method void logout(String userId), which prints a default logout message. Create classes PasswordAuthentication and BiometricAuthentication that implement the Authentication interface, each with its own logic for authentication and an optional override of the default logout method if needed. Demonstrate the use of these classes in a Main class by performing authentication and logout operations.

PROGRAM:

```
interface Authentication {
    boolean authenticate(String userId, String credentials);

    default void logout(String userId) {
        System.out.println(userId + " has been logged out successfully.");
    }
}

class PasswordAuthentication implements Authentication {
    private String storedPassword = "securePass123";

    @Override
    public boolean authenticate(String userId, String password) {
        if (password.equals(storedPassword)) {
            System.out.println("Password authentication successful for user: " + userId);
            return true;
        } else {
            System.out.println("Password authentication failed for user: " + userId);
            return false;
        }
    }

    @Override
    public void logout(String userId) {
        System.out.println("User " + userId + " has been securely logged out from Password
Authentication.");
    }
}

class BiometricAuthentication implements Authentication {
    private String storedBiometricData = "fingerprintHash123";

    @Override
    public boolean authenticate(String userId, String biometricData) {
        if (biometricData.equals(storedBiometricData)) {
            System.out.println("Biometric authentication successful for user: " + userId);
```

```
            return true;
        } else {
            System.out.println("Biometric authentication failed for user: " + userId);
            return false;
        }
    }
}

public class AuthenticationDemo {
    public static void main(String[] args) {
        // Create a PasswordAuthentication object
        Authentication passwordAuth = new PasswordAuthentication();

        // Perform password authentication
        boolean isAuthenticated = passwordAuth.authenticate("User1", "securePass123");
        if (isAuthenticated) {
            passwordAuth.logout("User1");
        }

        System.out.println();

        Authentication biometricAuth = new BiometricAuthentication();

        isAuthenticated = biometricAuth.authenticate("User2", "fingerprintHash123");
        if (isAuthenticated) {
            biometricAuth.logout("User2");
        }
    }
}
```

OUTPUT: