(Naima IT-22001)

1. When to use Interface and when to abstract class. Develop a story and write codes to explain.

Ans: Let's dereloping an online payment system that handles multiple payment methods - such as Credit Card, PayPal, and UPI.

- All payment method must support pay(amount) - this is a common action, but how it's implemented varies.
- Some payment type (like credit card) require shared logic such as validating the card number, checking credit limit etc.
- Some payment methods can optionally support refunds, but not all do - that's an optional behavior.

Design approach:

(i) Use an interface 'Refundable' for payments that can be refunded - it's not application to all payments.
(ii) Use an abstract class 'CreditCardPayment' for credit cards to encapsulate common code.
(iii) Each payment method implements its own logic for pay()

P.T.O

Code :-

```
interface Refundable{
    void refund(double amount);
}
abstract class CreditCardPayment{
    String cardNumber;
    CreditCardPayment(String cardNumber){
        this.cardNumber = cardNumber;
    }
    void validateCard(){
        System.out.println("Validating card: " + cardNumber
    }
    abstract void pay(double amount);
}
class VisaCard payment extends CreditCardPayment
    implements Refundable{
    VisaCardPayment(String cardNumber){
        super(cardNumber);
    }
public void pay(double amount){
    validateCard();
    System.out.println("Paid $"+amount+" using Visa");
}
```

```
    public void refund(double amount){
        System.out.println("Refunded $"+ amount+ " to
                                    Visa Card");
    }
}
class PayPalPayment implements Refundable{
    public void pay(double amount){
    System.out.println("Paid $"+amount+ "using
                                    PayPal.");
    }
    public void refund(double amount){
    System.out.println("Refunded $"+ amount +
                                "via PayPal");
    }
}
```

21) Is it true that invoking method in an interface is slower than abstract class method?

Ans: No, it is no longer meaningfully true in modern Java (Post Java 7+). Interface method calls were slightly slower in early JVMs because of dynamic dispatch, but in modern JVMs this difference is negligible.

Example with benchmark:

```
interface InterfaceExample{
    void show();
}
abstract class AbstractExample{
    abstract void show();
class InterfaceImpl implements InterfaceExample{
    public void show(){
        System.out.println("Interface method");
    }
}
class AbstractImpl extends AbstractExample{
    public void show(){
        System.out.println("Abstract method");
    }
}
```

3# Make a table to summarize the difference between abstract class and interface?

=>

Interface	Abstract Class
① Define a contract for behavior	① Provid base class with shared code.
② Can implement multiple interface	② Only one abstract class can be extended.
③ Can not have constractors.	③ Can have constructors
④ Only public static final (constants)	④ Can have instance variabl.