IT23039

(1) Demonstrate how a child class can access protected member of its parents class within the same package. Explain with example what happens when the child class is in a different package.

In java a protected member can be accessed

1. ~~wt~~ within the same package (like default access)

2. In a subclass, even if the subclass is in a different package.

Case-1: Child class in the same package

parent class (same package)

```
Package Pack1;
public class parent {
    protected int value = 10;
```

I123039

```
    protected void show() {
      System.out.printlh ("value: " + value);
    }
}
```

Child class: (Same package)

```
Package Pack1;
public class child extends parent {
    public void display () {
    System.out.println (value);
    Show () ;
}
public static void main (String[] args) {
        Child c = new Child();
        c.display ();
}
}
```

- Child class can directly access proteeted variable and method.
- work jsut like package-level acess.
- No inheritance restriction issues.

1123039

Child Class (Different package)

```
Package pack2;
import pack1.parent;
Public class child extends parent {
  public void display() {
     System.out.println (value);
  }
  Public static Void main (String [] args) {
       Child c = new child();
       c.display ();
  }
}
```

- protected member is accessible through inheritan
- Access via this.value or directly is allowed.

2. Compare abstract classes and interfaces in terms of multiple inheritance. when would you pefer to use an abstract class and when an interface.

Ans:

Abstract class:

- A class can extend only one abstract class.
- Does not support multiple inheritance of classes.

```
abstract class A {
        abstract void show();
    }
abstract class B {
          abstract void display();
    }
class c extends A, B { }
```

Interface

- A class can implement multiple interfaces.
- supports multiple inheritance.

```
interface A {
        Void Show();
}
interface B {
        void display();
}
class c implements A, B {
        public void Show() {
        system.out.println("Show");
        }
public void display() {
        system.out.println("Display");
    }
}
```

IT23039

Difference Between Abstract class and interface.

Feature	Abstract class	Interface
Multiple inheritance	Not supported	Supported
Methods	Abstract + Concrete	Abstract methods
Variables	Instance variable allowed	only public static final constants
Constructors	Yes	NO
Access modifiers	Any	Methods are public by default
State	can have state	No instance state

Use an abstract class when:

(i) we want share common code.

(ii) we need to maintain state.

(iii) Classes are closely related.

(iv) We want protected / private members.

(v) We don't ~~me~~ need multiple inheritance.

Example:

```
abstract class vehicle{
        int speed;
   abstract void move();
   void setspeed (int s) {
        speed = s;
   }
}
```

Use an interface when:

(i) we need multiple inheritance

(ii) we want to define a contract / capability

(iii) classes are unrelated but share behavior.

(iv) we want loose coupling

(v) we want to support future extensions.

```
interface Flyable{
        void fly();
   }
interface Swimmable{
        void swim();
   }
```

IT123039

```
class Duck implements Flyable, Swimmable {
        public void fly() {
            system.out.println ('Duck flies');
    }
    public void swim() {
            system.out.println(" Duck swim");
    }
  }
```

How

IT23039

(3) How does encapsulation ensure data security and integrity? Show with a BankAccount class using private variables and validated methods ~~n~~ such as setAccountNumber (string) setInitialBalance (double) that rejects null, negative or empty values.

Ans:

Encapsulation means wrapping data (variables) and method together and restricting direct access to data using access modifiers like private.

How it ensures data security

- Data members are declared private.
- Outside classes cannot modify data directly.
- Access is controlled only through methods.

How it ensures data integrity:

- Setter methods validate input.
- Invalid data (null, empty, negative values) is rejected.
- Object always stay in a valid state.

BankAccount Example :

```
class BankAccount {
    private String accountNumber;
    private double balance;

public void setAccountNumber (String accountNumber
  {
    if (accountNumber==NUll ||accountNumber.trim().isEmpty
                                                    ()) {
      system.out.println("Invalid accountnumber!");
     return ;
  }
     this.accountNumber = accountNumber ;
}
public void SetInitialBalance (double balance) {
     if ( balance < 0 ) {
       system.out.println ("Initial balance cannot be
                                    negative! ") ;
     return;
  }
   this . balance = balance.
}
```

I.T 23039

```
public string getAccountNumber() {
        return accountNumber;
   }
    public double getBalance () {
            return blo balance;

   }
 }
```

Test class

```
public class main {
    public static void main (String[] args) {
        BankAccount account = new BankAccount();
   account.SetAccountNumber (" "); //rejected
   account. setAccountNumber (null); //rejected
   account.setAccountNumber ("Acct 2345");

   account. SetInitialBalance (-500); //rejected
   account. SetInitialBalance (1000);
   System.out.println ("Account Number: " + account.
                              getAccountNumber() );
   System.out.println(" Balance : " + account.getBalance
                                    ());
    }
}
```

Output :

```
Invalid account number!
Invalid account number!
Initial balance cannot be negative!
Account Number : ACC12345
Balance : 1000.0
```