

DAY-40

11 August 2023 15:49

- A CLASS CAN ACT AS BOTH IMPLEMENTATION CLASS AS WELL AS SUBCLASS , i.e, A CLASS CAN HAVE EXTENDS RELATIONSHIP AND IMPLMENTS RELATIONSHIP AT THE SAME TIME

```
package interfaceDemo;
```

```
public interface Int1
{
    void hungry();
}
```

```
package interfaceDemo;
```

```
public class Test1
{
    public void tea()
    {
        System.out.println("i want a tea");
    }
}
```

```
package interfaceDemo;
```

```
public class Test2 extends Test1 implements Int1
{
    public void hungry()
    {
        System.out.println("i am hungry");
    }
    public static void main(String[] args)
    {
        Test2 a1 = new Test2();
        a1.hungry();
        a1.tea();
    }
}
```

WHEN TO GO FOR CLASS AND WHEN TO GO FOR INTERFACE ?

- WHENEVER WE HAVE PARTIAL IMPLEMENTATION OF AN APPLICATION, WE WILL MAKE USE OF ABSTRACT CLASS.
- BUT, WHEN WE DON'T HAVE ANY IMPLEMENTATION OF THE APPLICATION THEN, WE WILL MAKE USE OF INTERFACE.

***** DIFFERENCE BETWEEN INTERFACE AND ABSTRACT CLASS

INTERFACE	ABSTRACT CLASS
IT ALLOWS ONLY ABSTRACT METHODS	IT ALLOWS BOTH ABSTRACT AS WELL AS CONCRETE METHODS
ALL DATA MEMBERS ARE BY DEFAULT STATIC AND FINAL	DATA MEMBERS CAN NON STATIC/STATIC
WE CAN ACHIVE MULTIPLE INHERTIANCE	MULTIPLE INTERITANCE IS NOT ALLOWED
IT DOESN'T SUPPORT CONSTRUCTORS	IT SUPPORTSS CONSTRCUTORS
IMPLEMENTATION IS PROVIDED IN IMPLEMENTATION CLASS USING IMPLEMENTS KEYWORD	IMPLEMENTATION WILL PROVIDED IN SUB CLASS

COUPLING

COUPLING REFERS TO THE DEGREE TO WHICH ONE CLASS KNOWS ABOUT ANOTHER CLASS

TYPES OF COUPLING

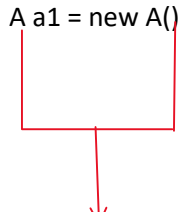
1. TIGHT COUPLING
2. LOOSE COUPLING

TIGHT COUPLING

- A CLASS TYPE REFERENCE CAN REFER ONLY THA TPARTICULAR CLASS OBJECT i.e, THE REFERENCE VARIABLE AND THE OBJECT SHOULD ALWAYS BE OF THE SAME CLASS TYPE.
- USING ON CLASS REFERENCE , WE CAN ACCESS THE PROPERTIES OF THAT PARTICULAR CLASS ONLY
- IF AT ALL WE TRY TO ACCESS ANOTHER CLASS PROPERTY WITH DIFFERENT CLASS TYPE REFERENCE, THE COMPILER WILL THROW AN ERROR BECAUSE , THE REFERENCE AND OBEJECT OF A CLASS TIGHTLY COUPLED WITH EACH OTHER.

```
CLASS A
{
    DATA MEMBERS
    MEMBER FUNCTIONS
}
```

```
CLASS B
{
    A a1 = new A()
}
```



TIGHTLY COUPLED

package coupling;

```
public abstract class TightCoupling
{
```

```

        public abstract void m1();
    }

package coupling;

public class Tight1 extends TightCoupling
{
    public void m1()
    {
        System.out.println("in m1 of first child");
    }
}

package coupling;

public class Tight2 extends TightCoupling
{
    public void m1()
    {
        System.out.println("in m1 of child2");
    }
}

package coupling;

public class Tight3 extends TightCoupling
{
    public void m1()
    {
        System.out.println("in m1 of child3");
    }
}

package coupling;

public class TightDemo
{
    public static void main(String[] args)
    {
        Tight1 t1 = new Tight1();

        Tight2 t2 = new Tight2();

        Tight3 t3 = new Tight3();

        t1.m1();
        t2.m1();
        t3.m1();
    }
}

```

LOOSE COUPLING

- USING INTERFACE, WE CAN ACHIEVE LOOSE COUPLING.
- WHENEVER WE CREATE REFERENCE OF AN INTERFACE, USING THIS INTERFACE TYPE REFERENCE WE CAN REFER TO ANY IMPLEMENTATION CLASS OBJECTS
- USING INTERFACE TYPE REFERENCE , WE CAN REFER TO ONLY THOSE CLASS OBJECTS WHICH ARE PROVIDING IMPLEMENTATION FOR THE INTERFACE
- THIS PROPERTY OF INTERFACE TYPE REFERENCE TO REFER DIFFERENT IMPLEMENTATION CLASS OBJECT ACCORDING TO THE USER REQUIREMENT IS CALLED LOOSE COUPLING.

package coupling;

```
public interface LooseCoupling
{
    void subject();
}
```

package coupling;

```
public class A implements LooseCoupling
{
    public void subject()
    {
        System.out.println("HTML");
    }
}
```

package coupling;

```
public class B implements LooseCoupling
{
    public void subject()
    {
        System.out.println("sql");
    }
}
```

package coupling;

```
public class C implements LooseCoupling
{
    public void subject()
    {
        System.out.println("core java");
    }
}
```

package coupling;

```
public class D
{
}
```

```
public static void main(String[] args)
{
    int choice =2;

    switch (choice)
    {

        case 1:

            LooseCoupling l1 = new A();
            l1.subject();

            break;

        case 2:

            LooseCoupling l2 = new B();
            l2.subject();

            break;

        case 3:

            LooseCoupling l3 = new C();
            l3.subject();

            break;

    }

}
}
```

