DAY-41

14 August 2023 15:37

ABSTRACTION

- HIDING THE IMPLEMENTATION AND GIVING FUNCTIONALITY TO THE USER IS CALLED AS ABSTRACTION
- WE CAN ACHIEVE ABSTRACTION IN 2 WAYS
    - USING ABSTRACT CLASS
    - USING INTERFACE
- IT IS RECOMMENDED TO USER INTERFACE TO PERFORM ABSTRACTION BECAUSE USING INTERFACE, WE CAN ACHIEVE LOOSE COUPLING.
- TO ACHIEVE ABSTRACTION, THE FOLLOWING STEPS MUST BE FOLLOWED :
    1. IDENTIFY THE COMMON PROPERTIES AND DECLARE THEM INSIDE INTERFACE.
    2. PROVIDE THE DIFFERENT IMPEMENTATION IN DIFFERENT IMPLMENTATION CLASSES
    3. CREATE AN ITERFACE TYPE REFERENCE AND REFER TO DIFFERENT IMPLEMENATION CLASS OBJECTS ACCORDING TO THE USER REQUIREMENT.

TYPE CASTING

WHAT IS TYPE CASTING

- CONVERTING ONE TYPE OF INFORMATION INTO ANOTHER TYPE OF INFORMATION IS KNOWN AS TYPECASTING.
- TYPECASTING IS CLASSFIED INTO 2 TYPES
    - DATA TYPE CASTING
    - CLASS TYPE CASTING

DATA TYPE CASTING

- CONVERTING ONE TYPE OF DATA INTO ANOTHER TYPE DATA IS KNOWN AS DATA TYPE CASTING
- DATA TYPE CASTING IS CLASSFIED INTO 2 TYPES :
    1. WIDENING
    2. NARROWING

WIDENING

- CONVERTING SMALLER TYPE OF DATA INTO LARGET TYPE OF DATA IS KNOWN AS WIDENING.
- WIDENING CAN HAPPEN BOTH IMPLICITLY BY JVM AND EXPLICITLY BY USER

Example :

```
package casting;

public class Widening {
```

```
    public static void main(String[] args)
    {
        double b = 10;//implicit widening by jvm
        System.out.println(b);

         double c = (double)100;//explicit widening
         System.out.println(c);
    }

}
```

NARROWING

- COVERTING LARGER TYPE OF DATA INTO SMALLER TYPE OF DATA IS KNOWN AS NARROWING
- SINCE, NARROWING INVOLVES DATA LOSS, THAT IS WHY IT HAS TO BE DONE EXPLICITLY BY USER ONLY.

```
package casting;

public class Narrowing
{

    public static void main(String[] args)
    {
        //int a = 2.1; //loss of data

         int a = (int)2.1; //explicit narrowing
         System.out.println(a);

         byte b = (byte)4.5; //explicit narrowing
         System.out.println(b);

         char d = (char)100000000;//explicit narrowing
         System.out.println(d);

    }

}
```

CLASS TYPE CASTING

- CONVERTING ONE CLASS INFORMATION TO ANOTHER CLASS IS KNOWN AS CLASS TYPE CASTING.
- CONVERTING SUBCLASS TYPE INFORMATION INTO SUPER CLASS TYPE INFORMATION IS CALLED AS CLASS TYPE CASTING
- FOR CLASS TYPE CASTING TO HAPPEN, 2 STEPS ARE TO BE FOLLOWED :
    1. THERE MUST BE **IS-A** RELATIONSHIP AMONG THE CLASSES
    2. THE CLASS WHICH WE ARE TRYING TO CONVERT SHOULD CONTAIN THE PROPERTY OF THE CLASS TO WHICH WE ARE TRYING TO CONVERT.
- CLASS TYPE CASTING CAN BE DONE BY 2 TYPES :
    - UPCASTING
    - DOWNCASTING

UPCASTING

- CONVERTING SUB CLASS TYPE INFORMATION INTO SUPER CLASS TYPE INFORMATION IS KNOWN AS UPCASTING.
- SINCE, IT FOLLOWS ABOVE 2 RULES, UPCASTING CAN BE DINE IMPLICITLY BY JVM AND ALSO EXPLICITLY BY THE USER.

DOWNCASTING

- CONVERTING SUPER CLASS TYPE INFORMATION INTO SUBCLASS TYPE INFORMATION IS KNOWN AS DOWNCASTING.
- SINCE, THE SUPER CLASS WILL NOT CONTAIN THE PROPERTY OF SUBCLASS, THAT IS WHY DOWNCASTING HAS TO BE DONE EXPLICITLY BY USER ONLY
- WE CAN PERFORM DOWNCASTING ONLY AFTER DOING UPCASTING, IF WE TRY TO DOWNCAST WITHOUT PERFORMING UPCASTING , JVM WILL THORW AN EXCEPTION (ClassCasteException).

```
package casting;

public class Upcast
{
	int x = 10;
	public void m1()
	{
		System.out.println("in m1 of parent");
	}

}
```

```
package casting;

public class UpcastChild extends Upcast
{
	int y = 20;

	public void m2()
	{
		System.out.println("in m2 of child");
	}
	public static void main(String[] args)
	{
		Upcast a = new UpcastChild();//implicit upcasting
		// a.m2(); //hiddent due to upcasting
		a.m1();

		 Upcast a1 = (Upcast)new UpcastChild(); //explicit upcasting

		  UpcastChild b = (UpcastChild)a;//explicit upcasting

		//  UpcastChild c = new Upcast(); //since upcasting is not done

	}
```

}

POLYMORPHISM

WHAT IS POLYMORPHISM

- AN OBEJCT SHOWING DIFFERENT BEHAVIOUR IN ITS DIFFERENT STAGES IS KNOWN AS POLYMORPHISM
- THERE 2 TYPES OF POLYMORPHISM JAVA
    1. STATIC POLYMORPHISM - COMPILE TIME - EARLY BINDING
    2. DYNAMIC POLYMORPHISM - RUN TIME - LATE BINDING

STATIC POLMORPHISM

- IT IS ALSO KNOWN AS COMPILE TIME POLYMORPHISM
- IN THIS POLYMORPHISM , THE BINDING/MAPPING OF METHODS DECLARATION WITH METHOD DEFENITION IS DONE BY COMPILER DURING COMPILE TIME.
- THE COMPILER WILL PERFORM THE BINDING OPERATION BASED ON THE :
    - LENGTH OF ARGUMENT
    - TYPE OF ARGUMENT
    - ORDER OF OCCURRENCE OF ARGUMENT
- SINCE THE BINDING TAKES PLACE DURING COMPILE TIME, IT IS ALSO KNOWN AS EARLY BINDING POLYMORPHISM

EXAMPLE :

- CONSTRCUTOR OVERLOADING
- METHOD OVERLOADING

```
package polymorphism;

public class Test1
{
	static String s = "BitsQ";

	public void Class(int tid, String sub)
	{
		System.out.println("Welcome to "+s);
		System.out.println("Your ID is "+tid);
		System.out.println("you are taking "+sub+ " subject");
	}

	public void Class(int sid, int cid)
	{
		System.out.println("Welcome to "+s);
		System.out.println("Your Student ID is "+sid);
		System.out.println("you are attending class code "+ cid);
	}
```

```
	public static void main(String[] args)
	{
		int tid = 10;
		String sub = "java";
		int sid = 25;
		int cid = 15;

		Test1 a1 = new Test1();
		a1.Class(tid, sub);
		a1.Class(sid, cid);
	}

}
```

DYNAMIC POLYMORPHISM

- IT IS KNOWN AS RUNTIME POLYMORPHISM
- IN THIS POLYMORPHISM , THE BINDING OF METHOD DECLARATION WITH METHOD DEFENITION IS DONE DURING RUNTIME BASED ON THE TYPE OF OBJECT.
- HENCE, IT IS CALLED AS RUNTIME POLYMORPHISM/LATE BINDING.
    - EXAMPLE : METHOD OVERRIDING

```
package polymorphism;

public class General
{
	public void appointment(int pid, int did)
	{
		System.out.println("patient id is "+pid);
		System.out.println("doctor id is "+did);
		System.out.println("appointment for general checkup confirmed");
	}
}

package polymorphism;

public class Scan extends General
{

	public void appointment(int pid, int did)
	{
		System.out.println("patient id is "+pid);
		System.out.println("doctor id is "+did);
		System.out.println("appointment for scan confirmed");
	}
	public static void main(String[] args)
	{
		int pid = 1234;
		int did = 7894;

		Scan a1 = new Scan();
```

```
        a1.appointment(pid, did);

        General a2 = new General();
        a2.appointment(pid, did);

    }

}
```

INSTANCE OF

- IT IS A KEYWORD USED TO CHECK IF GIVEN OBJECT CONTAINS THE PROERTIES OF THE SPECIFIED CLASS OR NOT
- IF THE GIVEN OBJECT CONTAINS PROPERTIES OF SPECIFIED CLASS THEN , IT WILL RETURN TRUE ELSE FALSE.
- USUALLY, WE MAKE USE OF ISNTANCE OF KEYWORD WHENEVER WE ARE DEALIING WITH MULTIPLE OBJECTS

```
package polymorphism;

public class Test2 {

    public static void main(String[] args)
    {
        Test2 a1 = new Test2();

        Test2 a2 = new Test2();

        Test2 a3 = new Test2();


        System.out.println(a1 instanceof Test2);
        System.out.println(a2 instanceof Test2);
        System.out.println(a3 instanceof Test2);



    }

}
```