POLYMORPHISM

- Polymorphism is derived from two different Greek words ‘Poly’means Numerous ‘Morphs’ means form Which means Numerous form. Polymorphism is the ability of an object to exhibit more than one form with the same name.
- **For Understanding :**
- One name -----> Multiple forms
- One variable name ------> Different values
- One method name -----> Different behaviour

TYPES OF POLYMORPHISM

In java, we have two types of polymorphism,

1. Compile-time polymorphism

2. Runtime Polymorphism

COMPILE-TIME POLYMORPHISM :

- If the binding is achieved at the compile-time and the same behaviour is executed it is known as compile-time polymorphism.
- It is also said to be static polymorphism.

NOTE :

Binding means an association of method call to the method definition.

Compile time Polymorphism is achieved by :

1. Method overloading

2. constructor overloading

3. Variable shadowing

4. Method shadowing

5. Operator overloading (does not support in java)

METHOD OVERLOADING

If more than one method is created with the same name but different formal arguments in the same class are known as method overloading.

EXAMPLE : java.lang.Math;

abs(double d)

abs(float f)

abs(int i)

abs(long l)

- These are some of the overloaded methods (methods with the same name but different formal arguments) implemented in java.lang.Math class.

CONSTRUCTOR OVERLOADING :

- A class having more than one constructor with different formal arguments is known as constructor overloading.

RUNTIME POLYMORPHISM :

RUNTIME POLYMORPHISM :

- If the binding is achieved at the compile time and different behaviour is executed then it is known as runtime polymorphism.
- It is also known as dynamic binding.
- It is achieved by method overriding.

METHOD OVERRIDING :

- If the subclass and superclass have non static methods with the same signature, it is known as method overriding.

Rule to achieve method overriding :

- **Is-A relationship is mandatory.**
- **It is applicable only for nonstatic methods.**
- **The signature of the subclass method and superclass method should be the same.**

EXAMPLE :

```
class Parent
{
public void test()
{
System.out.println(“From parent”);
}
}
class Child extends Parent
{
public void test()
{
System.out.println(“From child”);
}
class Driver
{
public static void main(String[] args)
{
Parent p = new Child();
p.test();
}
}
```

EXAMPLE :

Child c = new Child();

c.test(); // from child

Parent p = c;

p.test(); // from child

Internal runtime object is a child so child test() will get executed, it does not depend on the reference type

NOTE : Variable overriding is not applicable.

METHOD SHADOWING :

If a subclass and superclass have the static method with the same signature, it is known as method shadowing.

Which method implementation gets executed, depending on what?

In method shadowing binding is done at compile-time, hence it is compile-time polymorphism. The execution of the method depends on the reference type and does not depend on the type of object created

NOTE :

- **Access modifier should be same or higher visibility than superclass method.**
- **Method shadowing is applicable only for the static method.**
- **It is compiled time polymorphism**
- **Execution of implemented method depends on the reference type of an object**

Example:

```
class Parent
{
public static void m1()
{
System.out.println("From m1() of Parent");
}
}
class Child extends Parent
{
public static void m1()
{
System.out.println("From m1() of Child");
}
}
class Shadowing
{
public static void main(String[] args)
{
Parent p = new Parent();
p.m1();//From Parent
Child c = new Child();
c.m1();//From Child
Parent p1 = c;
p1.m1();//From Parent
}
}
```

VARIABLE SHADOWING :

VARIABLE SHADOWING :

If the superclass and subclass have variables with the same name then it is known as variable shadowing.

Which variable is used, depending on what?

In variable shadowing binding is done at compile-time, hence it is a compile-time polymorphism. The Variable used depends on the reference type and does not depend on the type of object created.

NOTE :

- **It is applicable for both static and non-static variables.**
- **It is a compile-time polymorphism.**
- **Variable usage depends on the type of reference and does not depend on the type of object created.**

Example 1: Static Variable

```
class A
{
static int i = 20;
}
class B extends A
{
static int i = 40;
}
class VariableShadowing
{
public static void main(String[] args)
{
A a = new A();
System.out.println(a.i);//20
B b = new B();
System.out.println(b.i);//40
a=b;
System.out.println(a.i);//20
}
}
```

Example 2: Non-Static Variable

```
class A
{
 int i = 20;
}
class B extends A
{
 int i = 40;
}
class VariableShadowing
{
public static void main(String[] args)
{
A a = new A();
System.out.println(a.i);//20
B b = new B();
System.out.println(b.i);//40
a=b;
//A a=new B();
System.out.println(a.i);//20
}
}
```