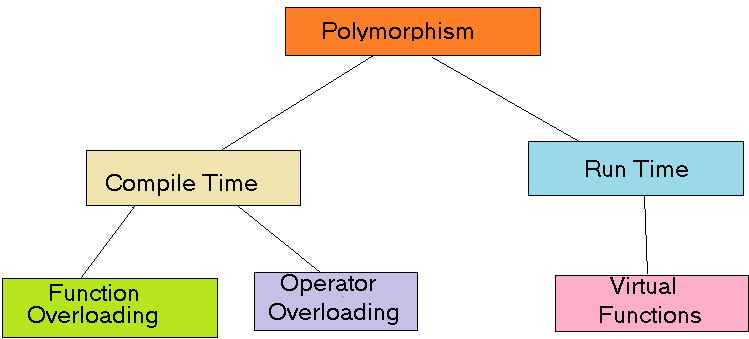
**# Polymorphism**

* + In Polymorphism we can perform a single task/action in different ways.
  + Polymorphism is derived from the Greek words ‘poly’ and ‘morph,’

meaning ‘many’ and ‘forms,’

**There are two types of polymorphism in Java:**

* compile-time polymorphism
* runtime polymorphism.



1. **compile-time polymorphism** :-
2. Compile time binding is re-solved by the complier
3. It is also known as static polymorphism or method overloading.
4. It is achieved by Method overloading and operator overloading.
5. When more then one methods have the same name but different parameters in single class .
6. **runtime polymorphism.**
7. **Runtime polymorphism** or **Dynamic Method Dispatch** is a process in which a call to an overridden method is resolved at runtime

**class A**

**{**

void show()

{

System.out.println("Default Parameter");

}

void show(int x)

{

System.out.println("One Parameter");

}

void show(int x, int y)

{

System.out.println("Two Parameter");

}

**}**

**class Q01\_Polymorphism\_Overloading**

**{**

public static void main(String args[])

{

A a = new A();

a.show();

a.show(10);

a.show(10, 20);

}

**}**

Output :-

Default Parameter

One Parameter

Two Parameter

**class A**

**{**

void show()

{

System.out.println("Default Parameter");

}

void show()

{

System.out.println("Default Parameter");

}

**}**

**class Q02\_Polymorphism\_Overloading**

**{**

public static void main(String args[])

{

A a = new A();

a.show();

}

**}**

Output :-

**error: method show() is already defined in class A**

**void show()**

**class A**

**{**

void show()

{

System.out.println("Default Parameter");

}

int show()

{

System.out.println("Default Parameter");

return 10;

}

**}**

**class Q03\_Polymorphism\_Overloading**

**{**

public static void main(String args[])

{

A a = new A();

a.show();

}

**}**

Output :-

**error: method show() is already defined in class A**

**int show()**

**class A**

**{**

void show(byte x, byte y)

{

System.out.println("Byte.");

}

void show(short x, short y)

{

System.out.println("Short.");

}

void show(int x, int y)

{

System.out.println("Int.");

}

void show(long x, long y)

{

System.out.println("Long.");

}

**}**

**class Q04\_Polymorphism\_Overloading**

**{**

public static void main(String args[])

{

A a = new A();

byte b1 = 10, b2 = 20;

short s1 = 100, s2 = 200;

int i1 = 1000, i2 = 2000;

long l1 = 10000, l2 = 20000;

a.show(b1, b2);

a.show(s1, s2);

a.show(i1, i2);

a.show(l1, l2);

System.out.println("");

a.show(1, 2);

a.show(10, 20);

a.show(100, 200);

a.show(1000, 2000);

a.show(10000, 20000);

}

**}**

Output :-

Byte.

Short.

Int.

Long.

Int.

Int.

Int.

Int.

**class A**

**{**

void show(byte x, byte y)

{

System.out.println("Byte.");

}

void show(short x, short y)

{

System.out.println("Short.");

}

void show(long x, long y)

{

System.out.println("Long.");

}

**}**

**class Q05\_Polymorphism\_Overloading**

**{**

public static void main(String args[])

{

A a = new A();

byte b1 = 10, b2 = 20;

short s1 = 100, s2 = 200;

int i1 = 1000, i2 = 2000;

long l1 = 10000, l2 = 20000;

a.show(b1, b2);

a.show(s1, s2);

a.show(i1, i2);

a.show(l1, l2);

a.show(1, 2);

a.show(10, 20);

a.show(100, 200);

a.show(1000, 2000);

a.show(10000, 20000);

}

**}**

Output :-

Byte.

Short.

Long.

Long.

Long.

Long.

Long.

Long.

Long.

**class A**

**{**

void show(byte x, byte y)

{

System.out.println("Byte.");

}

void show(short x, short y)

{

System.out.println("Short.");

}

**}**

**class Q06\_Polymorphism\_Overloading**

**{**

public static void main(String args[])

{

A a = new A();

byte b1 = 10, b2 = 20;

short s1 = 100, s2 = 200;

int i1 = 1000, i2 = 2000;

long l1 = 10000, l2 = 20000;

a.show(b1, b2);

a.show(s1, s2);

a.show(i1, i2);

a.show(l1, l2);

System.out.println("");

}

}

Output :-

**no suitable method found for show(int,int)**

**a.show(i1, i2);**

**^**

**error: no suitable method found for show(long,long)**

**a.show(l1, l2);**

^

**class A**

**{**

void show(byte x, byte y)

{

System.out.println("Byte.");

}

void show(short x, short y)

{

System.out.println("Short.");

}

**}**

**class Q07\_Polymorphism\_Overloading**

**{**

public static void main(String args[])

{

A a = new A();

a.show(10, 20);

}

**}**

Output :-

**no suitable method found for show(int,int)**

**a.show(10, 20);**

^

**class A**

**{**

void show(int x, double y)

{

System.out.println("Int : Double");

}

void show(double x, int y)

{

System.out.println("Double: Int");

}

**}**

**class Q08\_Polymorphism\_Overloading**

**{**

public static void main(String args[])

{

A a = new A();

a.show(10, 20.20);

a.show(10.10, 20);

a.show(10, 20);

a.show(10.10, 20.20);

}

**}**

Output :-

Int : Double

Double: Int

**error: reference to show is ambiguous**

**a.show(10, 20);**

**^**

**error: no suitable method found for show(double,double)**

**a.show(10.10, 20.20);**

**^**

**class A**

**{**

void show(int x, int y)

{

System.out.println("Class A");

}

**}**

**class B extends A**

**{**

void show(int x, int y)

{

System.out.println("Class B");

}

**}**

**class Q09\_Polymorphism\_\_OverRiding**

**{**

public static void main(String args[])

{

B b = new B();

b.show(10, 20);

}

**}**

Output :-

**Class B**

**class A**

**{}**

**class B extends A**

**{**

void show(int x, int y)

{

System.out.println("Class B");

}

**}**

**class Q10\_Polymorphism\_\_OverRiding**

**{**

public static void main(String args[])

{

B b = new B();

b.show(10, 20);

}

**}**

Output :- **Class B**

**class A**

**{**

**}**

**class B extends A**

**{**

**}**

**class Q11\_Polymorphism\_OverRiding**

**{**

public static void main(String args[])

{

B b = new B();

b.show(10, 20);

}

}

Output :-

**Class A**

**class A**

**{}**

**class B extends A**

**{}**

**class Q12\_Polymorphism\_\_OverRiding**

**{**

public static void main(String args[])

{

B b = new B();

b.show(10, 20);

}

}

Output :-

error: cannot find symbol

b.show(10, 20);

^

---------------------------------------------------------------------------------------------