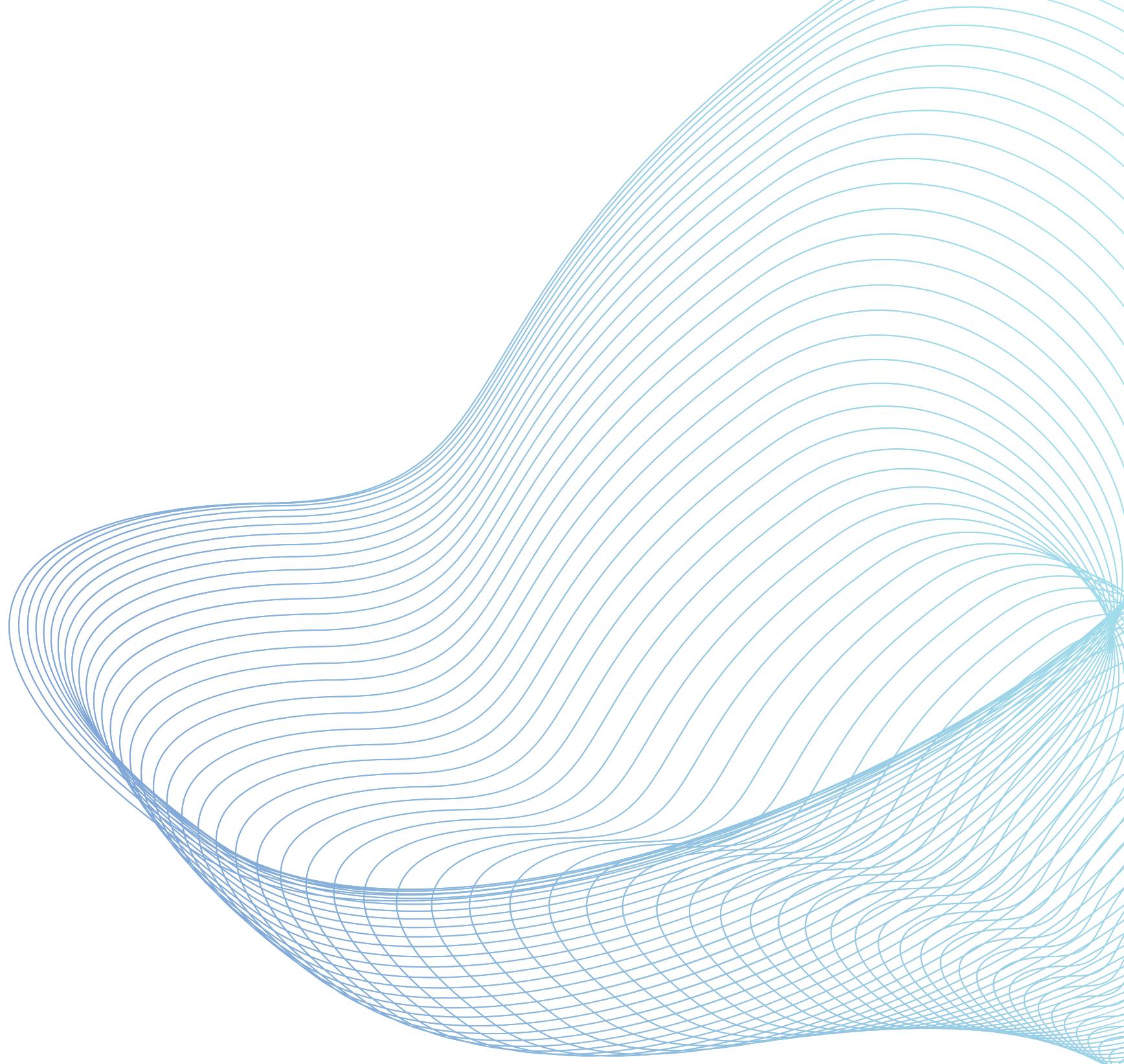


# **Dynamic Method Dispatch**



**-By Kulvir Singh Saggu  
(GU-2021-4246)**

# AGENDA

- Dynamic Method dispatch
- Syntax and code
- How the code works
- Upcasting
- Points To Remember

The screenshot shows an IDE interface with a file tree on the left and a code editor on the right.

**File Tree:**

- .idea
- out
- SRC
  - ARRAY
  - ghexit
  - JAVA\_PROGRAM
  - kulvir.java
  - Main
  - METHOD\_DISPATCH.java
  - mETHOD\_GELLO.java
  - run\_time\_polymorphism.java
  - ubuntu
- GNA University.iml
- External Libraries
- Scratches and Consoles

**Code Editor:**

```
2  public void hell(){  
3      System.out.println("Your welcome");  
4  }  
5  public void check(){  
6      System.out.println("Your welcome sir");  
7  }  
8  }  
9  class kulvir extends run{  
10     public void check(){  
11         System.out.println("hello how are you");  
12     }  
13     public void hell(){  
14         System.out.println("Hello have a great day");  
15     }  
16  }  
17  }  
18  public class run_time_polymorphism {  
19      public static void main(String[] args) {  
20          new run_time_polymorphism().hell();  
21          new run_time_polymorphism().check();  
22          new kulvir().check();  
23          new kulvir().hell();  
24      }  
25  }  
26  
```

**Run Tab:**

Run: run\_time\_polymorphism x  
"C:\Program Files\Java\jdk-19\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2.1\lib\javassist-agent.jar"

Output:  
Your welcome sir.  
Your welcome  
Process finished with exit code 0

# DYNAMIC METHOD DISPATCH

The Basic Points we have to know before we go further because dynamic method dispatch follow these points as you can see.

1

**Using Method Overriding we can achieve runtime polymorphism.**

2

**Dynamic Metod Dispatch is a Mechanism that decides which overridden method will be called at runtime.**

3

**Here we use the reference of Superclass and object(instance) of Subclass.**

4

**Based on the object being referred, it will decided which method will be called.**

# SYNTAX

- Superclass obj = new Subclass\_obj()
- **Note:- We can use reference variable of superclass and obj of subclass.**
- **It is worked on run time polymorphism,we can say it is dynamic binding or late binding.**

# Code:-

We know that run time polymorphism work on method overriding, having same fuctions and same parameters.

```
class Animal {  
    void breath() {  
        System.out.println("Breathing...");  
    }  
}  
  
class Dog extends Animal {  
    void breath() {  
        System.out.println("Breathing as Dog...");  
    }  
}  
  
class Fish extends Animal {  
    void breath() {  
        System.out.println("Breathing as Fish...");  
    }  
}
```

# In the main class we write:-

We know that run time polymorphism work on method overriding, having same fuctions and same parameters.

```
class methodOverrideDemo {  
    public static void main(String args[]) {  
        Animal a;  
        a = new Dog();  
        a.breath();  
        a = new Fish();  
        a.breath();  
    }  
}
```

**OutPut:**

- 1 | Breathing as Dog...
- 2 | Breathing as Fish...

# Run-time polymorphism cannot be achieved by data members

```
// Java program to illustrate the fact that
// runtime polymorphism cannot be achieved
// by data members

// class A
class A
{
    int x = 10;
}

// class B
class B extends A
{
    int x = 20;
}

// Driver class
public class Test
{
    public static void main(String args[])
    {
        A a = new B(); // object of type B

        // Data member of class A will be accessed
        System.out.println(a.x);
    }
}
```

Output:

10

## Explanation of flow of execution of the above program:

```
public class X {  
    X0 {  
        System.out.println("Parent class constructor");  
        m1();  
    }  
    void m1(){  
        System.out.println("Parent method");  
    }  
}  
  
public class Y extends X {  
    Y0 {  
        System.out.println("Child class constructor");  
    }  
    void m10 {  
        System.out.println("Child class method");  
    }  
}  
  
public class Demo extends Y {  
    public static void main(String[] args) {  
        Y y=new Y0;  
        y.m10; } }
```

Fig: Flow of execution of program

# How the superclass and subclass will assigned?

```
class Bank {  
    int maximumInterestRate = 15;  
}
```

```
class Sbi extends Bank {  
    int maximumInterestRate = 9;  
}
```

```
public class MethodOverridingDemo {  
  
    public static void main(String[] args) {  
        Bank b = new Sbi();  
        System.out.println("Maximum Interest Rate : " + b.maximumInterestRate);  
    }  
}
```

Reference  
variable of  
parent class



Object of  
Child class

# Upcasting

**It means that we can access overidden methods of a child class or subclass. We cannot acces their attributes and variables.**

- Here we will doing upcasting:-
- For example:-  
Parent a = new child();  
[Here parent reerence variable will access the child method of a subclass.]

# Points to remember

## **Dynamic Method Dispatch in Java**

- Dynamic method dispatch is also known as run time polymorphism.
- It is the process through which a call to an overridden method is resolved at runtime.
- This technique is used to resolve a call to an overridden method at runtime rather than compile time.
- To properly understand Dynamic method dispatch in Java, it is important to understand the concept of upcasting because dynamic method dispatch is based on upcasting.



A large, bold, black sans-serif font text "THANKYOU" is centered within a white rectangular frame. The frame is set against a background of abstract, translucent blue and teal watercolor washes. The composition is surrounded by a light gray border and is framed by stylized, thin blue line art resembling petals or leaves in the corners.

**THANKYOU**