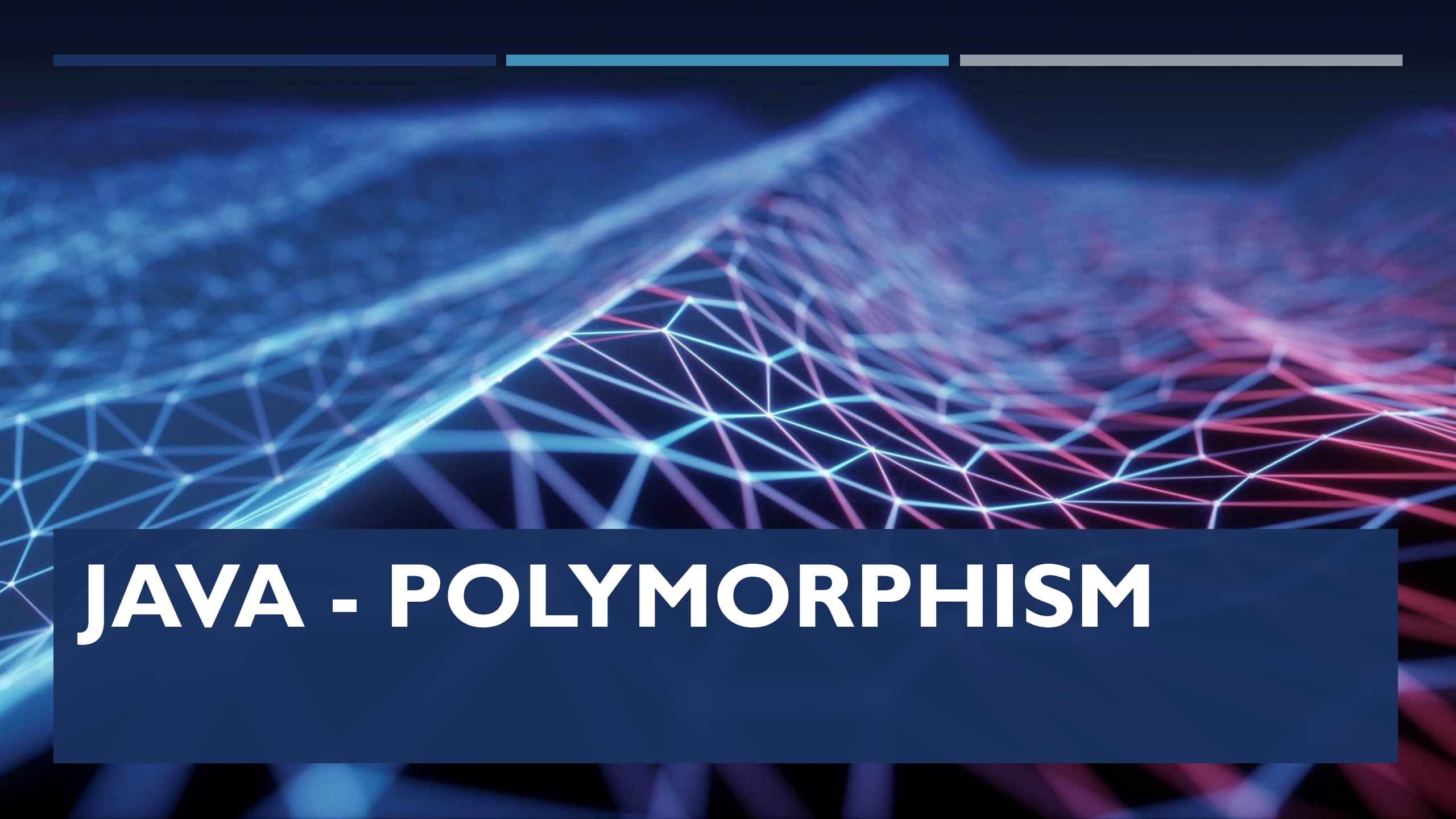


JAVA - POLYMORPHISM

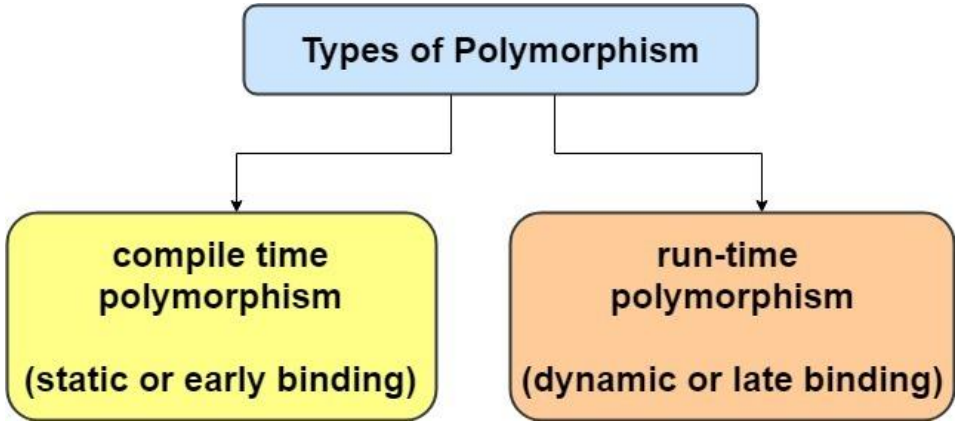
POLYMORPHISM

- Polymorphism is derived from 2 greek words: poly and morphs. The word "poly" means many and "morphs" means forms.
- So Polymorphism means the ability to take many forms. is a concept by which we can perform a ***single task*** *in* ***different ways***.
- It is one of the most striking features of Object Oriented Programming in Java.
- In terms of Java Programming Polymorphism is the capability of a method to do different things based on the object that it is acting upon.
- In other words, polymorphism allows you define one interface and have multiple implementations.

TYPES OF POLYMORPHISM

- There are two types of polymorphism in java: **compile time polymorphism(static or early binding)** and **runtime polymorphism(dynamic or late binding)**.

  - Compile Time Polymorphism – Method Overloading
  - Run Time Polymorophism – Method Overriding



RUN TIME POLYMORPHISM

- **Runtime polymorphism** or **Dynamic Method Dispatch** is a process in which a call to an overridden method is resolved at runtime rather than compile-time.
- In this process, an overridden method is called through the reference variable of a **superclass**. Thus this happens only when Inheritance is implemented.
- The method to be called is based on the object being referred to by the reference variable.

RUN TIME POLYMORPHISM

- When an overridden method is called through a superclass reference, Java determines which version(superclass/subclasses) of that method is to be executed based upon the type of the object being referred to at the time the call occurs. Thus, this determination is made at run time.
- At run-time, it depends on the type of the object being referred to (not the type of the reference variable) that determines which version of an overridden method will be executed
- A superclass reference variable can refer to a subclass object. This is also known as **upcasting**. Java uses this fact to resolve calls to overridden methods at run time.

A SIMPLE JAVA PROGRAM TO DEMONSTRATE METHOD OVERRIDING IN JAVA

```
// Base Class
class Parent {
void show() {
System.out.println("Parent's show()"); } }
// Inherited class
class Child extends Parent {
// This method overrides show() of Parent
void show() {
System.out.println("Child's show()"); } }
// Driver class
class Main {
public static void main(String[] args) {
// If a Parent type reference refers
// to a Parent object, then Parent's
// show is called
Parent obj1 = new Parent();
obj1.show();
// If a Parent type reference refers
// to a Child object Child's show()
// is called. This is called RUN TIME
// POLYMORPHISM.
Parent obj2 = new Child();
obj2.show(); } }
```

SOME RULES TO FOLLOW IN METHOD OVERRIDING

- **Overriding and Access-Modifiers :** The access modifier for an overriding method can allow more, but not less, access than the overridden method. For example, a protected instance method in the super-class can be made public, but not private, in the subclass.
- **Final methods can not be overridden.**
- **Static methods can not be overridden.**
- **Private methods can not be overridden.**
- **The overriding method must have same return type (or subtype)**
- **Invoking overridden method from sub-class :** We can call parent class method in overriding method using **super** keyword.
- **Overriding and constructor :** We can not override constructor as parent and child class can never have constructor with same name(Constructor name must always be same as Class name).
- **Overriding and abstract method :** Abstract methods in an interface or abstract class are meant to be overridden in derived concrete classes otherwise compile-time error will be thrown.

UPCASTING VS DOWNCASTING IN JAVA

- Upcasting in Java –
  - ***Upcasting*** *is* ***casting*** *a* ***subtype*** *to a* ***supertype, upward*** *to the* ***inheritance tree***.
- Downcasting in Java –
  - When Subclass type refers to the object of Parent class, it is known as **downcasting**. If we perform it directly, compiler gives Compilation error. If you perform it by typecasting, ClassCastException is thrown at runtime. But if we use instanceof operator, downcasting is possible.