# Polymorphism – Complete Notes (Detailed)

## 1️⃣ Polymorphism – Definition

• এক নাম, একাধিক রূপ (Many forms)
• দুই ধরনের:
• **Compile-time (Static) Polymorphism** → Method Overloading
• **Runtime (Dynamic) Polymorphism** → Method Overriding, Abstract class, Interface
• Purpose: **Code reusability, flexibility, dynamic behavior**

## 2️⃣ Method Overloading (Compile-time Polymorphism)

• Same method name, different parameter list (number or type)
• Return type may vary (but not used for differentiation)
• Access modifier can vary

```
class MathOperation {
    int sum(int a, int b) { return a + b; }
    double sum(double a, double b) { return a + b; }
    int sum(int a, int b, int c) { return a + b + c; }
}
```

## 3️⃣ Method Overriding (Runtime Polymorphism)

• Child class overrides parent class method
• Dynamic method dispatch → runtime এ child method call হয়
• Rules:
• Method name + parameter same
• Return type same or covariant
• Access modifier same/wider
• Checked exception same/subtype

```
class Animal {
    void sound() { System.out.println("Animal makes sound"); }
}
class Dog extends Animal {
    @Override
    void sound() { System.out.println("Dog barks"); }
}
```

## 4️⃣ Covariant Return Type

- Child method return type parent return type এর subtype হতে পারে
- Example:

```
class Animal {
    Animal getAnimal() { return new Animal(); }
}
class Dog extends Animal {
    @Override
    Dog getAnimal() { return new Dog(); }
}
```

- Advantage: Allows more specific return types while maintaining polymorphism

## 5️⃣ Abstract Class and Abstract Method

- Abstract class → blueprint, cannot instantiate directly
- Abstract method → must be implemented by child
- Can have concrete methods too

```
abstract class Animal {
    abstract void sound();
    void sleep() { System.out.println("Animal sleeps"); }
}
class Cat extends Animal {
    @Override
    void sound() { System.out.println("Cat meows"); }
}
```

## 6️⃣ `@Override` Annotation

- Indicates method override to compiler
- Helps catch errors if method signature mismatches

```
@Override
void sound() { System.out.println("Dog barks"); }
```

## 7️⃣ Polymorphism using Array of Objects

- Parent type reference holds multiple child objects
- Fixed size array

```
Animal[] animals = new Animal[2];
animals[0] = new Dog();
animals[1] = new Cat();
for(Animal a : animals) a.sound();
```

**Output:**

```
Dog barks
Cat meows
```

## 8️⃣ Polymorphism using Collection of Objects (ArrayList)

- Dynamic size collection

```
import java.util.ArrayList;
ArrayList<Animal> animals = new ArrayList<>();
animals.add(new Dog());
animals.add(new Cat());
for(Animal a : animals) a.sound();
```

**Output:**

```
Dog barks
Cat meows
```

- Benefits over array: dynamic size, easier insertion/deletion, generic support

## 9️⃣ Key Points / Rules

1. Method Overloading → compile-time polymorphism
2. Method Overriding → runtime polymorphism
3. Covariant return type → child method return type parent return type এর subtype
4. Polymorphic array/collection → parent reference → multiple child objects → runtime method dispatch
5. Abstract method → child class implement করতে বাধ্য
6. `@Override` → compiler-friendly, code readable
7. Array → fixed size, Collection → dynamic size
8. Allows flexibility, code reusability, dynamic behavior

## 🔟 Shortcut

Polymorphism = flexibility + dynamic behavior → same reference দিয়ে multiple child behavior handle করা যায়