INTRODUCTION TO PROGRAMMING LANGUAGE II(JAVA)

Fariha Zahin

Reference type vs. Object type

Reference Type

- The **declared type** of a variable.
- Determines:
  - What methods and fields are **accessible** at **compile-time**.
  - Whether the code compiles at all.
- Comes from the **left side** of the assignment:

```
Parent p = new Child();  // Reference type: Parent
```

Object Type

- The **actual type** of the object in memory, created using new

Determines:

- What **overridden methods** run at **runtime**.
- Comes from the **right side** of the assignment:

```
Parent p = new Child();  // Object type: Child
```

Dynamic Dispatch in Java is the process by which a call to an **overridden method** is resolved **at runtime** (not at compile time).
It happens when a **parent class reference variable refers to a child class object**, and the method that gets executed depends on the **object type**, not the reference type.

Key Point:
- Variables are resolved **at compile time**.
- Methods (if overridden) are resolved **at runtime** → this is **dynamic dispatch** (a.k.a runtime polymorphism).

Operation	Determined By	Notes
Which method gets called (if overridden)	✅ Object Type (Runtime)	Dynamic dispatch applies
Which method is visible (if not overridden)	❌ Reference Type (Compile Time)	Compiler checks reference type
Which field is accessed	❌ Reference Type (Compile Time)	Fields are not polymorphic
Casting allowed?	✅ Reference Type	You can only call subclass methods/fields after casting

```java
class Parent {
    int x = 10;
    void speak() { System.out.println("Parent speaks"); }
}

class Child extends Parent {
    int x = 20;
    @Override
    void speak() { System.out.println("Child speaks"); }
    void onlyChildDoes() { System.out.println("Child-only method"); }
}
```

```java
public class Test {
    public static void main(String[] args) {
        Parent p = new Child();  // ref: Parent, obj: Child

        System.out.println(p.x);     // → 10 (reference type wins)
        p.speak();                   // → Child speaks (object type wins)

        // p.onlyChildDoes();       // ❌ Compile error (Parent doesn't have this method)

        ((Child) p).onlyChildDoes(); // ✅ OK after casting
    }
}
```

```java
class Animal {
    String name = "Animal";

    void sound() {
        System.out.println("Animal makes a sound");
    }
}

class Dog extends Animal {
    String name = "Dog";

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

```java
class Cat extends Animal {
    String name = "Cat";

    @Override
    void sound() {
        System.out.println("Cat meows");
    }
}
```

```java
public class Main {
    public static void main(String[] args) {
        Animal a1 = new Dog();   // parent reference, child object
        Animal a2 = new Cat();   // parent reference, child object
        Animal a3 = new Animal();// parent reference, parent object

        // --- Dynamic Dispatch in action ---
        a1.sound();  // Dog barks
        a2.sound();  // Cat meows
        a3.sound();  // Animal makes a sound

        // --- Variable behavior ---
        System.out.println(a1.name);  // Animal
        System.out.println(a2.name);  // Animal
        System.out.println(a3.name);  // Animal
    }
```