FACULTY OF ENGINEERING
Department of Telecommunications and Electronics Engineering

JAVA OOP

Basic Java Course

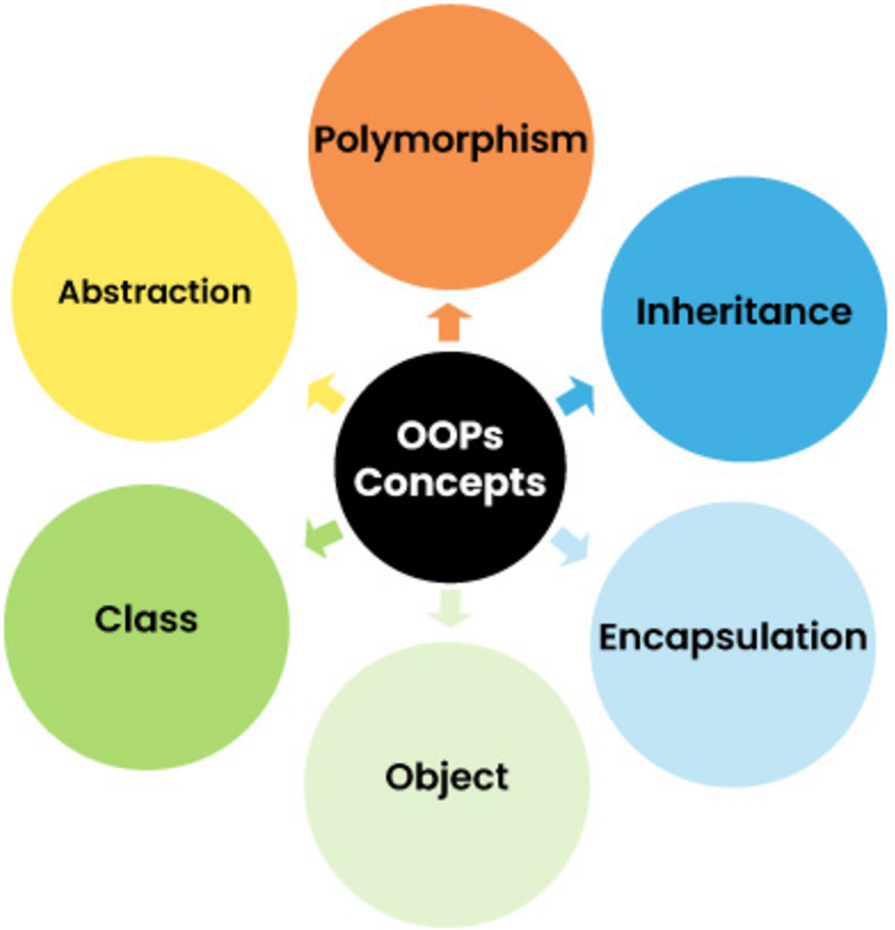
Part2

Advisor: Prof. Thap Thareoun

Present By : Ngouch HongCheng
: Saing LymChhun

INTRODUCTION

Object-Oriented Programming (OOP) in Java is a programming paradigm that uses objects and classes to structure software in a way that models real-world entities and relationships.



INTRODUCTION

Some Important uses of OOP in java

- **Modularity**: Encapsulation keeps data and methods together, making code more modular and secure.
- **Code Reusability**: Inheritance allows you to reuse and extend existing code easily.
- **Flexibility**: Polymorphism lets methods work differently based on the object, enhancing flexibility.
- **Maintainability**: The organized structure makes code easier to maintain and debug.
- **Collaboratio**n: Modular design allows team members to work independently on different parts of a project.

INHERITANCE

A. DEFINITION

- **Inheritance** is one of the key features of OOP that allows us to create a new class from an existing class.
- The new class that is created is known as subclass (child or derived class) and the existing class from where the child class is derived is known as superclass (parent or base class).
- The *extends* keyword is used to perform inheritance in Java.

INHERITANCE

B. IMPORTANCE OF INHERITANCE

- **Code Reusability**: Inheritance allows you to reuse existing code, saving time and effort.
- **Maintainability:** It makes updating and maintaining code easier by centralizing common functionality.
- **Organized Structure:** Inheritance helps organize related classes into a clear hierarchy, making the code easier to understand.
- **Polymorphism:** It enables flexible and dynamic code by allowing objects of different subclasses to be treated as objects of their superclass.
- **Avoids Duplication:** Inheritance reduces code duplication by placing shared behavior in a common parent class.
- **Extensibility:** It allows for easy addition of new features by extending existing classes without modifying them.

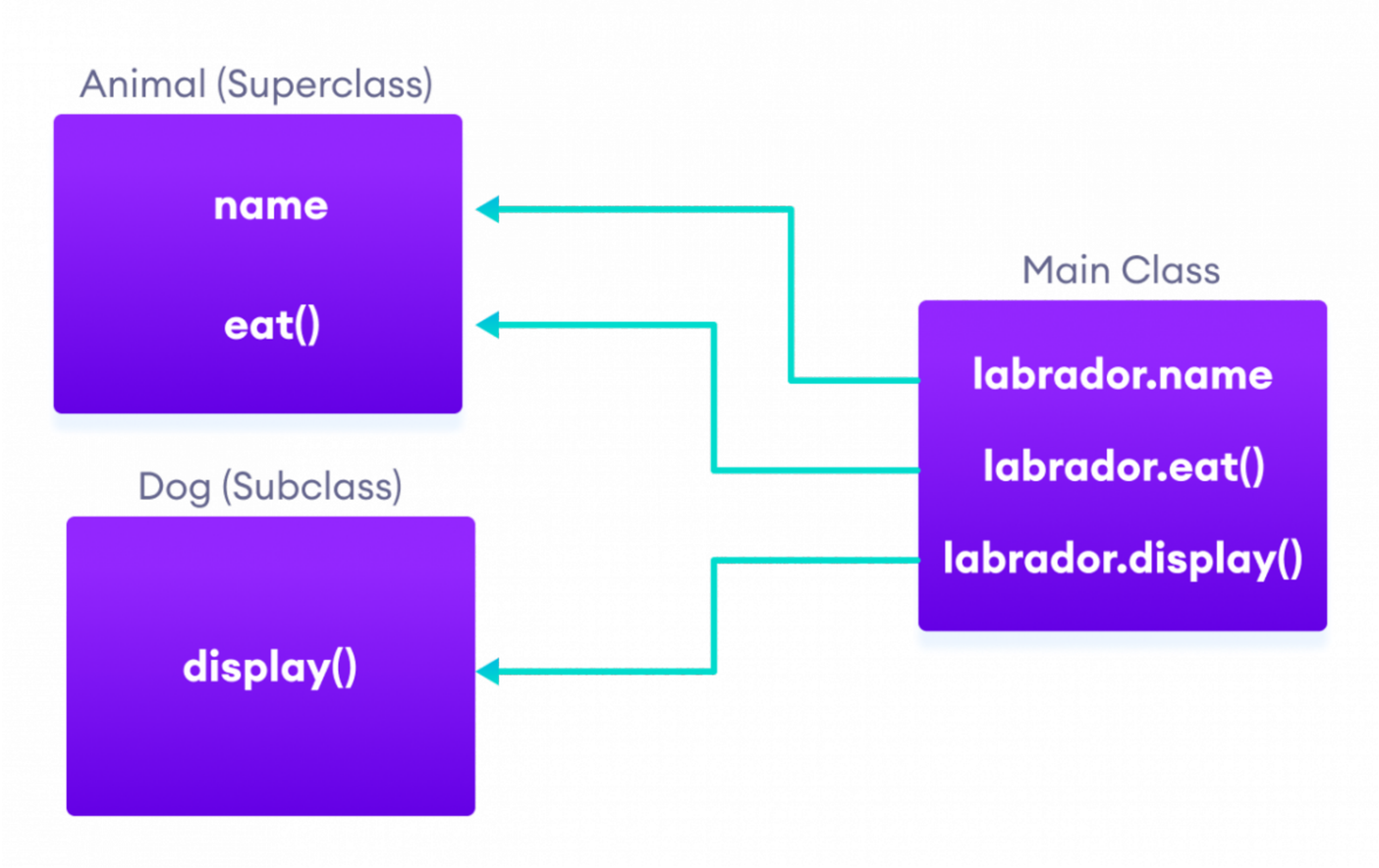
INHERITANCE

C. HOW INHERITANCE WORKS IN JAVA

- **Define the Superclass:** Create a class with common fields and methods.
- **Define the Subclass:** Use the extends keyword to create a class that inherits from the superclass.
- **Access Inherited Members:** Use the inherited fields and methods from the superclass in the subclass.
- **Create and Use Objects:** Instantiate the subclass and utilize its and inherited features.

INHERITANCE

C. HOW INHERITANCE WORKS IN JAVA



INHERITANCE

D. ‘EXTENDS’ KEYWORD

- In Java, the extends keyword is used to establish an inheritance relationship between classes.
- It allows a class (the subclass or derived class) to inherit fields and methods from another class (the superclass or base class).

Syntax

```
class Subclass extends Superclass {
// Fields, constructors, and methods of Subclass
}
```

INHERITANCE

D. ‘EXTENDS’ KEYWORD

Advantages of ‘Extends’ keyword

- **Code Reusability:** It allows you to reuse code from an existing class rather than writing the same code again in multiple classes. This promotes DRY (Don't Repeat Yourself) principles.
- **Organize and Simplify Code:** It helps in organizing code in a hierarchical manner, making it easier to manage and understand. You can create more specific classes that extend general ones.
- **Extend and Customize Functionality:** It allows you to extend or customize the functionality of an existing class by adding new fields or methods or by overriding existing methods.

INHERITANCE

E. “OVERRIDE” ANOTATION

- **@Override** is an annotation in Java that is used to indicate that a method is intended to override a method declared in a superclass.
- When you use the @Override annotation, the compiler checks to ensure that you are correctly overriding a method from the superclass.
- If the method in the subclass does not correctly override a method from the superclass (due to a typo, incorrect method signature, etc.), the compiler will generate an error.

Syntax

```
@Override
public void methodName() {
    // Method implementation
}
```

INHERITANCE

E. “OVERRIDE” ANOTATION

Key Points of ***@Override:***

- Ensures Correct Overriding: Helps catch errors during compilation by ensuring that the method in the subclass matches a method in the superclass.

- Improves Code Clarity: Makes the code easier to understand by clearly indicating which methods are overridden.

- Supports Polymorphism: Allows a subclass to provide a specific implementation of a method that will be called through a reference of the superclass type.

INHERITANCE

F. “SUPER” KEYWORD

- **super** is a keyword that refers to the superclass (parent class) of the current object.
- It is used within a subclass to access methods, constructors, and fields defined in its superclass.

Syntax

```
super(arguments);
super.methodName(arguments);
super.fieldName;
```

INHERITANCE

F. “SUPER” KEYWORD

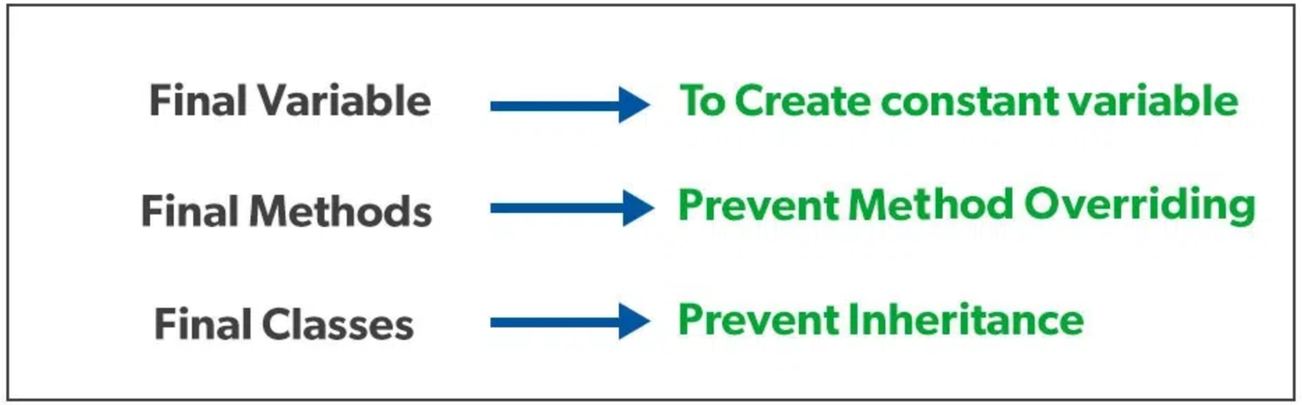
The **super** keyword allows you to:

- Call the superclass's constructor from a subclass constructor.
- Invoke a superclass method that has been overridden in the subclass.
- Access a superclass field that has been hidden by a field in the subclass.

INHERITANCE

G. “FINAL” KEYWORD

- The **Final** method in Java is used as a non-access modifier applicable only to a variable, a method, or a class.
- It is used to restrict a user in Java.
- The following are different contexts where the final is used with variable, method, and class.



INHERITANCE

G. “FINAL” KEYWORD - VARIABLE

- When you declare a variable as final, you are essentially making it a constant.
- Once a final variable is assigned a value, it cannot be modified.

Syntax

```
final int MAX_SPEED = 120;
```


- When you declare a variable as final, you are essentially making it a constant.
- Once a final variable is assigned a value, it cannot be modified.


```
final int MAX_SPEED = 120;
MAX_SPEED = 150; // Error: cannot assign a value to final variable MAX_SPEED
```

INHERITANCE

G. “FINAL” KEYWORD - VARIABLE

- When a reference variable (like an object) is declared as final, the reference itself cannot be changed, but the object's state can still be modified.

```
final Car myCar = new Car();
myCar = new Car(); // Error: cannot assign a new object to final variable myCar
myCar.speed = 150; // Allowed: modifying the object's state
```

INHERITANCE

G. “FINAL” KEYWORD - METHOD

- A method declared with the final keyword cannot be overridden by subclasses.
- This is useful when you want to lock down the behavior of a method to ensure it is not modified by any subclass.

Syntax

```
class Vehicle {
    final void display() {
        System.out.println("Vehicle display");
    }
}
```

INHERITANCE

G. “FINAL” KEYWORD - METHOD

- Attempting to override a final method in a subclass will cause a compilation error.

```
class Car extends Vehicle {
    // Error: Cannot override the final method from Vehicle
    void display() {
        System.out.println("Car display");
    }
}
```

- The JVM can optimize the execution of final methods because it knows they won’t be overridden. However, the actual performance gain is typically minor.

INHERITANCE

G. “FINAL” KEYWORD - CLASS

- A class declared as final cannot be subclassed.
- This is useful when you want to ensure that the implementation of the class cannot be altered or extended.

Syntax

```
final class Vehicle {
    // Class content
}
```

INHERITANCE

G. “FINAL” KEYWORD - CLASS

- Any attempt to inherit from a final class will result in a compilation error.

```
// Error: Cannot inherit from final Vehicle
class Car extends Vehicle {
    // Class content
}
```

- The String class in Java is an example of a final class.
- Making String final ensures that the behavior of string manipulation in Java remains consistent and secure.

END