Polimorphism

Pertemuan 8

Zulkifli Arsyad, S.Kom., M.T.

Topics

Polymorphism References

Polymorphism via Inheritance

Polymorphism via Interfaces

Polymorphic References

This occurs when a superclass reference is used to refer to subclass objects. The reference type is the superclass, but the actual object is a subclass. This allows for dynamic method invocation at runtime.

Polymorphic References example

```
class Employee {
    void calculateSalary() {
        System.out.println("Calculating salary for a generic employee");
    }
}

class FullTimeEmployee extends Employee {
    @Override
    void calculateSalary() {
        System.out.println("Calculating salary for a full-time employee");
    }
}

class PartTimeEmployee extends Employee {
    @Override
    void calculateSalary() {
        System.out.println("Calculating ary for a part-time employee");
    }
```

Polymorphic References example

```java
public class Main {
    public static void main(String[] args) {
        // Superclass reference to subclass objects
        Employee emp1 = new FullTimeEmployee();
        Employee emp2 = new PartTimeEmployee();

        emp1.calculateSalary();  // Output: Calculating salary for a full-time employee
        emp2.calculateSalary();  // Output: Calculating salary for a part-time employee
    }
}
```

Key Points

1. **The reference is of type Employee, but the actual object is either FullTimeEmployee or PartTimeEmployee.**
2. **The method calculateSalary() is invoked based on the actual object type at runtime, showing runtime polymorphism.**

Polymorphic Inheritance

In this form of polymorphism, subclasses inherit behavior from a superclass and may override methods to provide specific behavior.

Polymorphic Inheritance

```
class Employee {
    String name;

    public Employee(String name) {
        this.name = name;
    }

    void calculateSalary() {
        System.out.println(name + ": Calculating salary for a generic employee");
    }
}

class FullTimeEmployee extends Employee {
    public FullTimeEmployee(String name) {
        super(name);
    }

    @Override
    void calculateSalary() {
        System.out.println(name + ": Calculating salary for a full-time employee with bene
    }
}
```

Polymorphic Inheritance

```
class PartTimeEmployee extends Employee {
    public PartTimeEmployee(String name) {
        super(name);
    }

    @Override
    void calculateSalary() {
        System.out.println(name + ": Calculating salary for a part-time employee with hour
    }
}

public class Main {
    public static void main(String[] args) {
        FullTimeEmployee fullTimeEmp = new FullTimeEmployee("John");
        PartTimeEmployee partTimeEmp = new PartTimeEmployee("Sarah");

        fullTimeEmp.calculateSalary();  // Output: John: Calculating salary for a full-tim
        partTimeEmp.calculateSalary();  // Output: Sarah: Calculating salary for a part-ti
    }
}
```

Key Points

- Inheritance: Both FullTimeEmployee and PartTimeEmployee inherit from the Employee class.
- Overriding: Each subclass provides a specialized implementation of calculateSalary().
- Polymorphism: Even though each employee is treated as an Employee, their behavior is determined by the subclass they belong to.

Polymorphism via Interfaces

In this form of polymorphism, different classes implement a common interface, and objects of these classes are referred to using the interface type.

```java
interface Employee {
    void calculateSalary();
}

class FullTimeEmployee implements Employee {
    private String name;

    public FullTimeEmployee(String name) {
        this.name = name;
    }

    @Override
    public void calculateSalary() {
        System.out.println(name + ": Calculating salary for a full-time employee");
    }
}
```

Copy code

```
class PartTimeEmployee implements Employee {
    private String name;

    public PartTimeEmployee(String name) {
        this.name = name;
    }

    @Override
    public void calculateSalary() {
        System.out.println(name + ": Calculating salary for a part-time employee");
    }
}

public class Main {
    public static void main(String[] args) {
        // Interface reference to the implementing class
        Employee emp1 = new FullTimeEmployee("John");
        Employee emp2 = new PartTimeEmployee("Sarah");

        emp1.calculateSalary();  // Output: John: Calculating salary for a full-time emplo
        emp2.calculateSalary();  // Output: Sarah: Calculating salary for a part-time empl
    }
}
```

Key Points

- The Employee interface defines the method calculateSalary(), which is implemented by both FullTimeEmployee and PartTimeEmployee.
- The interface reference (Employee emp1, Employee emp2) is used to hold objects of the classes that implement the interface.
- Polymorphism via interfaces is useful when you want to decouple the implementation from the interface (allowing multiple classes to implement the same behavior without class inheritance).