**Introduction**

Polymorphism is one of the core concepts of Object-Oriented Programming (OOP). It allows methods to do different things based on the object it is acting upon, even though they share the same name. Polymorphism provides a way to perform a single action in different forms. In Java, polymorphism can be achieved through method overloading and method overriding.

**Table of Contents**

1. What is Polymorphism?
2. Types of Polymorphism
3. Method Overloading
4. Method Overriding
5. Real-World Examples of Polymorphism
6. Example: Polymorphism with Method Overloading
7. Example: Polymorphism with Method Overriding
8. Example: Payment Processing Example
9. Conclusion

**1. What is Polymorphism?**

Polymorphism means "many shapes" or "many forms." In Java, it refers to the ability of a single method or class to take on multiple forms. This is achieved through method overloading (compile-time polymorphism) and method overriding (runtime polymorphism).

**2. Types of Polymorphism**

* **Compile-time Polymorphism (Method Overloading)**: This type of polymorphism is resolved during compile time. Method overloading allows a class to have more than one method with the same name, provided their parameter lists are different.
* **Runtime Polymorphism (Method Overriding)**: This type of polymorphism is resolved during runtime. Method overriding allows a subclass to provide a specific implementation of a method that is already defined in its superclass.

**3. Method Overloading**

Method overloading occurs when a class has multiple methods with the same name but different parameter lists (different types or numbers of parameters).

**Example:**

public class MathUtils {

public int add(int a, int b) {

return a + b;

}

public double add(double a, double b) {

return a + b;

}

public int add(int a, int b, int c) {

return a + b + c;

}

}

**4. Method Overriding**

Method overriding occurs when a subclass provides a specific implementation of a method that is already defined in its superclass.

**Example:**

class Animal {

public void makeSound() {

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

}

}

class Dog extends Animal {

@Override

public void makeSound() {

System.out.println("Dog barks");

}

}

class Cat extends Animal {

@Override

public void makeSound() {

System.out.println("Cat meows");

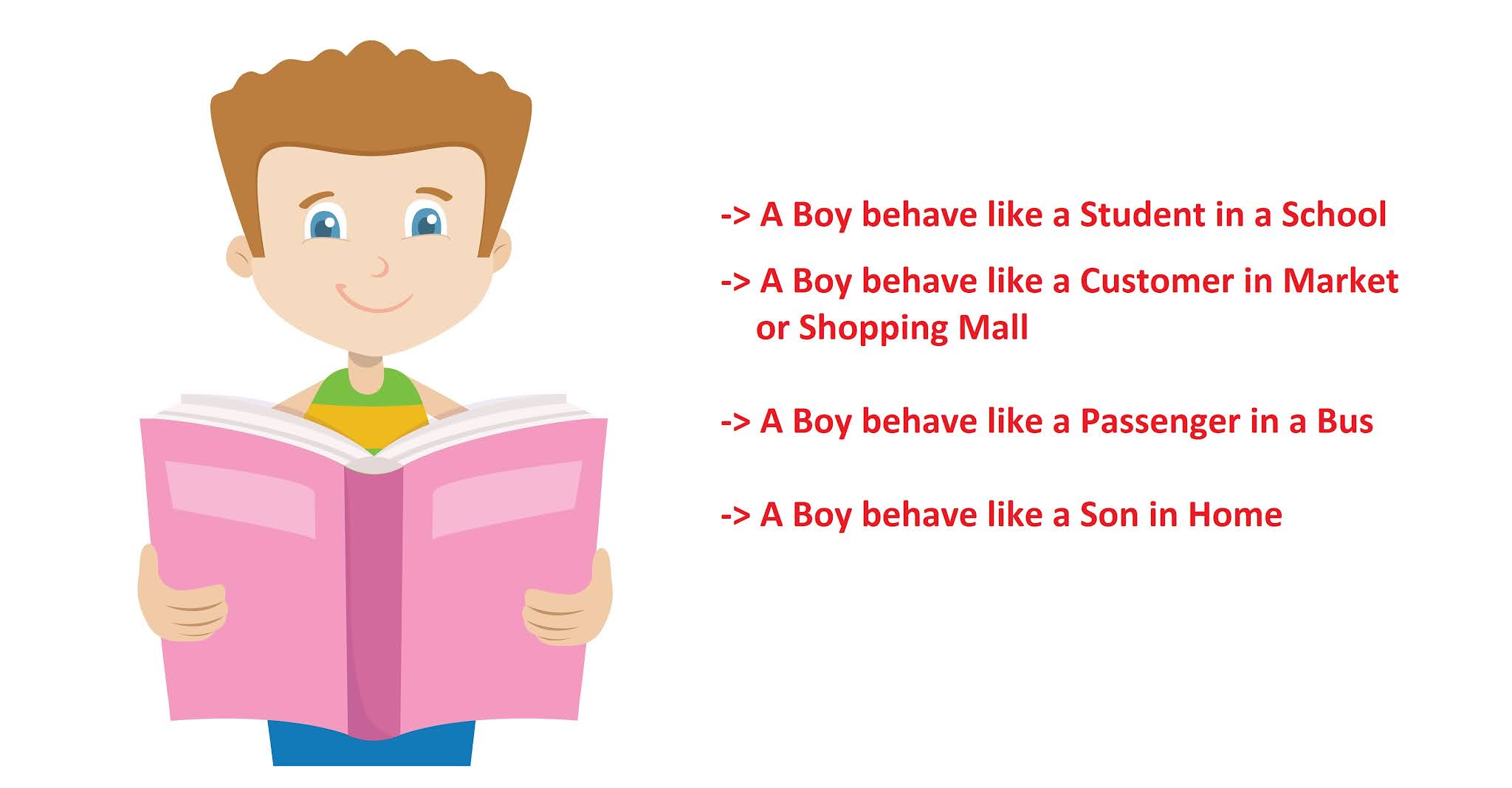
}

}

**5. Real-World Examples of Polymorphism**

**Example 1: Different Behaviors of a Person**

Suppose you are in a classroom, you behave like a student. When you are in the market, you behave like a customer. When you are at home, you behave like a son or daughter. Here, one person exhibits different behaviors in different contexts.

**[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEirSQr7FFnEA6vrECTkfcu1sGSCfxVFF3ZAgphGLAXZQF2ec0E5bViFk_Pm4MbP-Qz-eDkDvlwp9KQGORcXN9NU5umzZyYV7AV1cPG0VaMHURxVjMfdVvq_TBpmN_tr-4F_dPQVDN6DSVI/s2048/Polymorphism+Real-World+Example.jpg)**

**Example 2: Payment Processing System**

In a payment processing system, different payment methods such as credit card, debit card, and PayPal have different processing steps.

**6. Example: Polymorphism with Method Overloading**

**Example:**

public class Printer {

public void print(String message) {

System.out.println(message);

}

public void print(int number) {

System.out.println(number);

}

public void print(double number) {

System.out.println(number);

}

public static void main(String[] args) {

Printer printer = new Printer();

printer.print("Hello, World!"); // Output: Hello, World!

printer.print(123); // Output: 123

printer.print(3.14); // Output: 3.14

}

}

**Explanation:**

* **Printer**: Class with overloaded print methods to handle different types of input.
* **Main method**: Demonstrates the use of overloaded print methods.

**7. Example: Polymorphism with Method Overriding**

**Example:**

class Animal {

public void makeSound() {

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

}

}

class Dog extends Animal {

@Override

public void makeSound() {

System.out.println("Dog barks");

}

}

class Cat extends Animal {

@Override

public void makeSound() {

System.out.println("Cat meows");

}

}

public class Main {

public static void main(String[] args) {

Animal myDog = new Dog();

Animal myCat = new Cat();

myDog.makeSound(); // Output: Dog barks

myCat.makeSound(); // Output: Cat meows

}

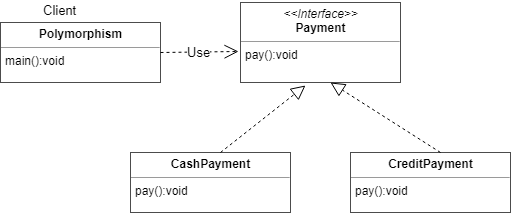
}

**Explanation:**

* **Animal**: Base class with a makeSound method.
* **Dog and Cat**: Subclasses that override the makeSound method.
* **Main method**: Demonstrates runtime polymorphism by calling the overridden methods on Dog and Cat objects.

**8. Example: Payment Processing Example**

**Class Diagram**

**[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEiKGHuIRKne8p6kcn4IhrKaGY3Gzk1lGABYCixriNyv5q8XYTiDvKMm4IU3_cZXE07kWI49UeokII3tOwUoYIDxuLt4PH7j6t1c7hwp72A0CeMfPXGdnqOM0a_ag_ZZtW6xUnlvHDnXUp8/s1600/cashpayment.png)**

**Code Example:**

**Payment Interface:**

interface Payment {

void pay();

}

**CashPayment Class:**

class CashPayment implements Payment {

@Override

public void pay() {

System.out.println("Payment made using cash.");

}

}

**CreditPayment Class:**

class CreditPayment implements Payment {

@Override

public void pay() {

System.out.println("Payment made using credit card.");

}

}

**Client Class:**

public class Polymorphism {

public static void main(String[] args) {

Payment payment;

payment = new CashPayment();

payment.pay(); // Output: Payment made using cash.

payment = new CreditPayment();

payment.pay(); // Output: Payment made using credit card.

}

}

**Explanation:**

* **Payment**: Interface defining the pay method.
* **CashPayment and CreditPayment**: Classes implementing the Payment interface and providing their own implementations of the pay method.
* **Polymorphism**: Client class demonstrating polymorphism by using the Payment interface to call the pay method on different types of payment objects.

**9. Conclusion**

Polymorphism in Java is a powerful concept that allows methods to perform different tasks based on the object they are acting upon. It enhances flexibility and maintainability in code by allowing a single method or class to take on multiple forms. Method overloading and method overriding are two ways to achieve polymorphism in Java. Real-world examples like different behaviors of a person in different contexts and various payment processing methods further illustrate the usefulness of polymorphism.