METHOD OVERRIDING

In Java, method overriding allows a subclass to provide its own implementation of a method that is already defined in its superclass. This allows the subclass to modify the behavior of the inherited method according to its specific requirements. Method overriding is a fundamental concept in object-oriented programming and promotes code reusability and polymorphism.

Here's an example program that demonstrates method overriding in Java:

```
// Superclass

class Animal {

    public void makeSound() {

        System.out.println("The animal makes a sound");

    }

}



// Subclass

class Cat extends Animal {

    @Override

    public void makeSound() {

        System.out.println("Meow");

    }

}



// Subclass

class Dog extends Animal {

    @Override

    public void makeSound() {

        System.out.println("Woof");
    }

}



// Main class

public class Main {

    public static void main(String[] args) {

        Animal animal = new Animal();

        Cat cat = new Cat();

        Dog dog = new Dog();



        animal.makeSound();  // Output: The animal makes a sound

        cat.makeSound();     // Output: Meow

        dog.makeSound();     // Output: Woof

    }

}
```

In this example, we have a superclass called Animal with a method makeSound(). The Cat and Dog classes are subclasses of Animal that override the makeSound() method with their own implementations.

When we create an object of each class and invoke the makeSound() method, the respective overridden method in the subclass is called, producing the desired output.

Note the use of the @Override annotation before the makeSound() methods in the subclasses. This annotation is optional but recommended as it helps catch errors at compile-time if the method doesn't actually override a method in the superclass.