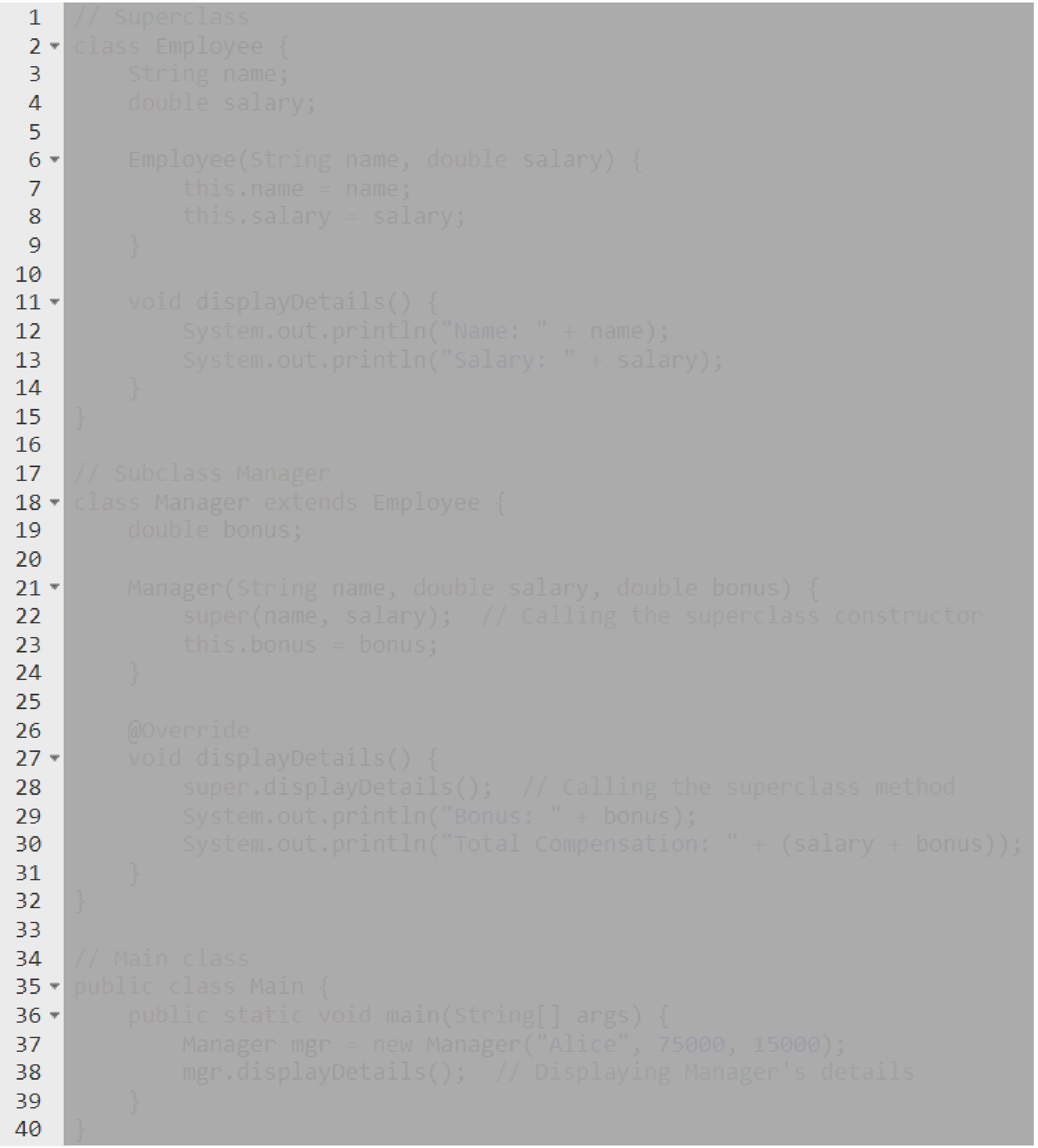
Problem Statement

Create a class `Employee` with attributes `name` and `salary`. Then create a subclass `Manager` that adds an additional attribute `bonus`. Write a method `displayDetails()` in both classes to display the details of the employee and manager. In the main class, create an object of `Manager`, set its attributes, and display its details.

Solution



```java
// Superclass
class Employee {
    String name;
    double salary;

    Employee(String name, double salary) {
        this.name = name;
        this.salary = salary;
    }

    void displayDetails() {
        System.out.println("Name: " + name);
        System.out.println("Salary: " + salary);
    }
}

// Subclass Manager
class Manager extends Employee {
    double bonus;

    Manager(String name, double salary, double bonus) {
        super(name, salary);  // Calling the superclass constructor
        this.bonus = bonus;
    }

    @Override
    void displayDetails() {
        super.displayDetails();  // Calling the superclass method
        System.out.println("Bonus: " + bonus);
        System.out.println("Total Compensation: " + (salary + bonus));
    }
}

// Main class
public class Main {
    public static void main(String[] args) {
        Manager mgr = new Manager("Alice", 75000, 15000);
        mgr.displayDetails();  // Displaying Manager's details
    }
}
```

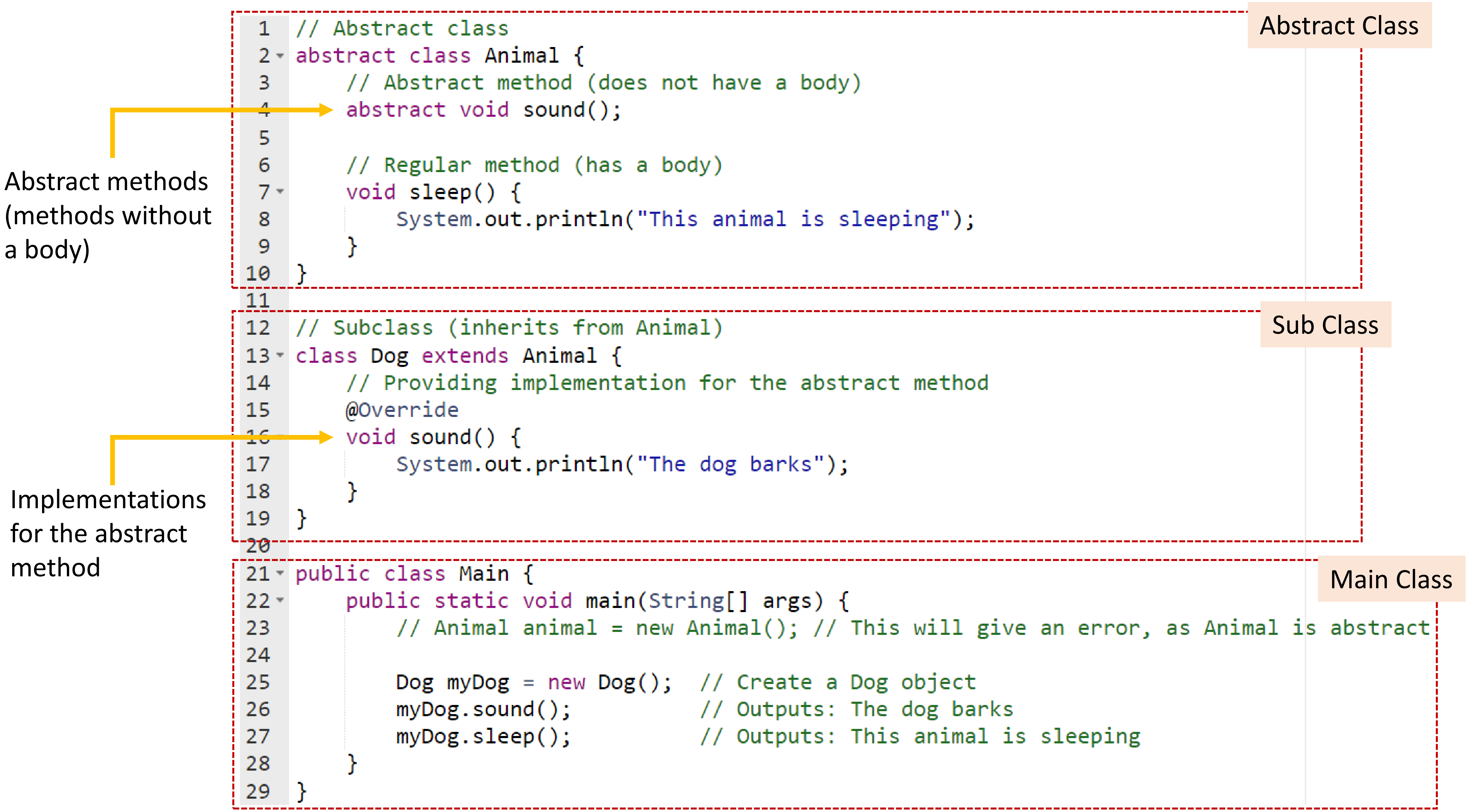
Abstract Class

- An **abstract class** in Java is a class that *cannot be instantiated directly*.
- It can contain *abstract methods* (methods without a body) as well as *non-abstract methods* (methods with a body).
- Abstract classes are used to *provide a base class for other classes to extend* and are particularly useful when you want to define a common interface for all subclasses but *allow each subclass to provide its specific implementation*.

Abstract Class

Key Points

- An abstract class is declared using the `abstract` keyword.
- Abstract classes cannot be instantiated directly. You must *create a subclass that extends the abstract class and provides implementations for its abstract methods*.
- An abstract class can have both abstract methods and non-abstract methods.
- If a class has *at least one abstract method*, the class itself must be declared as abstract.
- A subclass that extends an abstract class must implement all its abstract methods unless the subclass is also abstract.



Abstract Class

```java
// Abstract class
abstract class Animal {
    // Abstract method (does not have a body)
    abstract void sound();

    // Regular method (has a body)
    void sleep() {
        System.out.println("This animal is sleeping");
    }
}
```

Sub Class

```java
// Subclass (inherits from Animal)
class Dog extends Animal {
    // Providing implementation for the abstract method
    @Override
    void sound() {
        System.out.println("The dog barks");
    }
}
```

Main Class

```java
public class Main {
    public static void main(String[] args) {
        // Animal animal = new Animal(); // This will give an error, as Animal is abstract

        Dog myDog = new Dog();  // Create a Dog object
        myDog.sound();          // Outputs: The dog barks
        myDog.sleep();          // Outputs: This animal is sleeping
    }
}
```

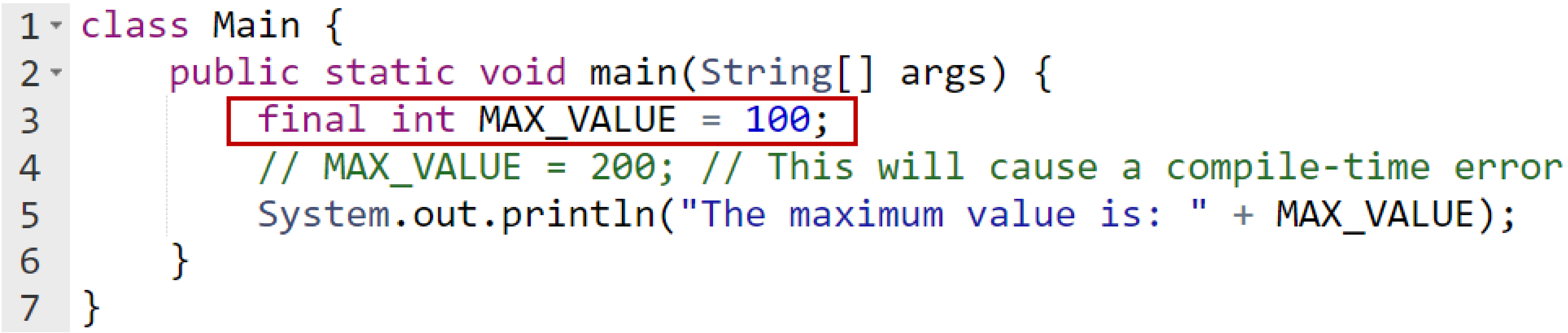
Abstract methods (methods without a body)

Implementations for the abstract method

The `final` keyword in Java

Final Variables

When a variable is declared with the final keyword, its value cannot be modified after it is initialized. This makes the variable *a constant*.

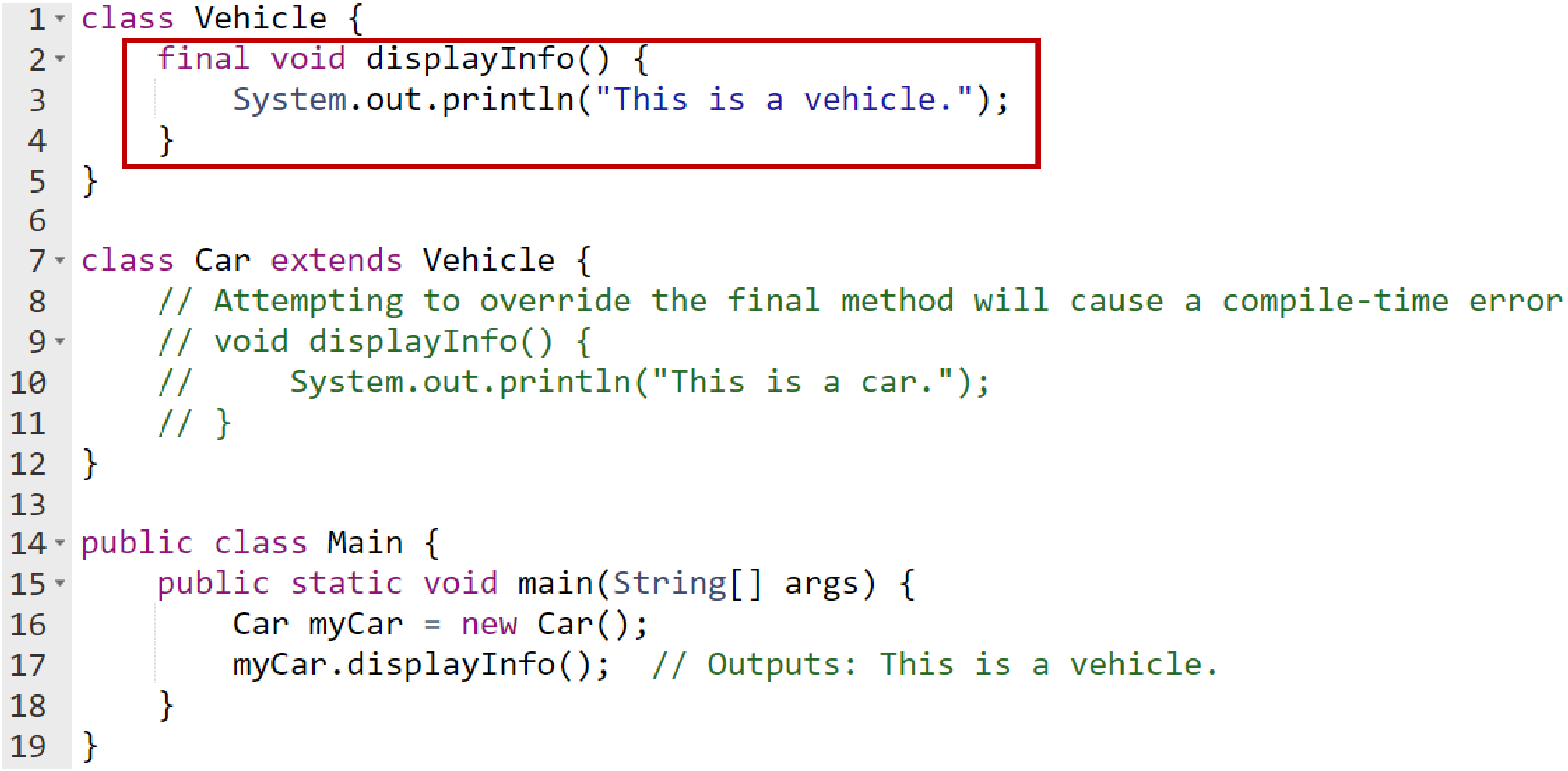


```java
class Main {
    public static void main(String[] args) {
        final int MAX_VALUE = 100;
        // MAX_VALUE = 200; // This will cause a compile-time error
        System.out.println("The maximum value is: " + MAX_VALUE);
    }
}
```

The `final` keyword in Java

Final Method

When a method is declared as final, it *cannot be overridden by subclasses*. This is useful when you want to prevent subclasses from altering the behavior of a method.

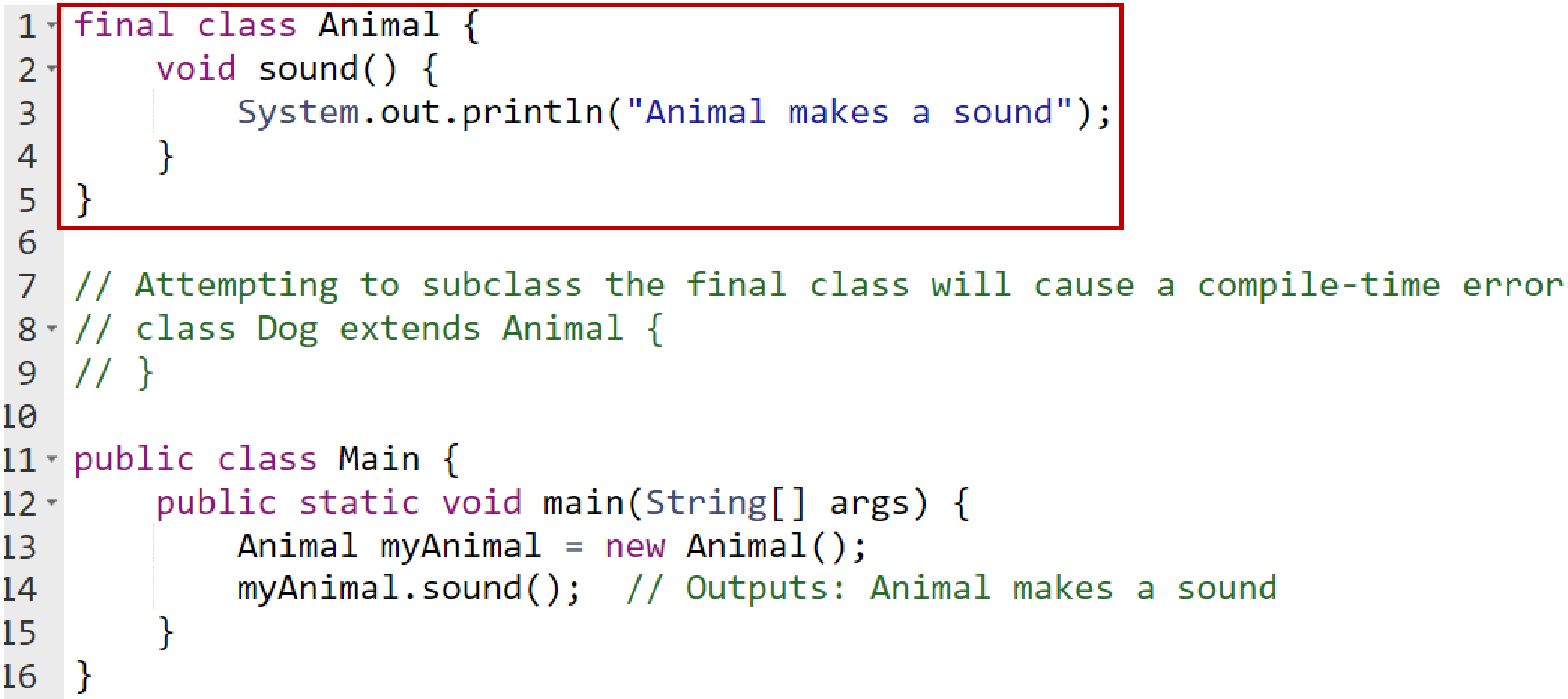


```
class Vehicle {
    final void displayInfo() {
        System.out.println("This is a vehicle.");
    }
}

class Car extends Vehicle {
    // Attempting to override the final method will cause a compile-time error
    // void displayInfo() {
    //     System.out.println("This is a car.");
    // }
}

public class Main {
    public static void main(String[] args) {
        Car myCar = new Car();
        myCar.displayInfo();  // Outputs: This is a vehicle.
    }
}
```

The `final` keyword in Java

Final Class

When a class is declared as final, it *cannot be subclassed*. This is useful when you want to create an immutable class or prevent inheritance for security or design reasons.



```java
final class Animal {
    void sound() {
        System.out.println("Animal makes a sound");
    }
}

// Attempting to subclass the final class will cause a compile-time error
// class Dog extends Animal {
// }

public class Main {
    public static void main(String[] args) {
        Animal myAnimal = new Animal();
        myAnimal.sound();  // Outputs: Animal makes a sound
    }
}
```