Java OOP Basics Explained

Access modifiers and access specifiers in Java are keywords that set the access level for classes, variables, methods, and constructors. They help control the visibility and accessibility of the members of a class from other classes and packages.

Access Modifiers

In Java, access modifiers determine the scope and visibility of a class member. The four types of access modifiers are:

1. **private**
2. **default (no modifier)**
3. **protected**
4. **public**

1. private

- **Scope**: Accessible only within the class in which it is declared.
- **Usage**: Used to encapsulate data and hide implementation details from other classes.

```java
public class MyClass {
    private int data;

    private void display() {
        System.out.println("Data: " + data);
    }
}
```

2. default (no modifier)

- **Scope**: Accessible only within the same package. If no access modifier is specified, it is considered to have package-private access.
- **Usage**: Used when you want the member to be accessible within the package but not from outside the package.

```java
class MyClass {
    int data;

    void display() {
        System.out.println("Data: " + data);
    }
}
```

3. protected

- **Scope**: Accessible within the same package and subclasses in different packages.
- **Usage**: Used when you want to allow subclasses to access the member while still restricting access from other non-subclass classes in different packages.

```java
public class MyClass {
    protected int data;

    protected void display() {
        System.out.println("Data: " + data);
    }
}
```

4. public

- **Scope**: Accessible from any other class.
- **Usage**: Used when you want the member to be accessible from any other class.

```java
public class MyClass {
    public int data;

    public void display() {
        System.out.println("Data: " + data);
    }
}
```

Access Specifiers

In Java, the term "access specifier" is often used interchangeably with "access modifier," though technically, Java has only access modifiers. However, sometimes "access specifier" can refer more broadly to the keywords that specify access levels, including those not traditionally listed as access modifiers, such as `**final**`, `**static**`, and `**abstract**`.

1. final

- **Usage**: Prevents modification. When used with classes, it prevents inheritance. When used with methods, it prevents method overriding. When used with variables, it makes them constants.

```java
public final class FinalClass {
    // This class cannot be extended
}

public class MyClass {
    public final void display() {
        // This method cannot be overridden
    }
```

```
    public void show() {
        final int data = 10;
        // data cannot be modified
    }
}
```

2. static

- **Usage**: Indicates that the member belongs to the class, rather than instances of the class. Static members can be accessed without creating an instance of the class.

java

```
public class MyClass {
    public static int data;

    public static void display() {
        System.out.println("Data: " + data);
    }
}

public class Main {
    public static void main(String[] args) {
        MyClass.data = 10;
        MyClass.display(); // Outputs: Data: 10
    }
}
```

3. abstract

- **Usage**: Indicates that a class cannot be instantiated and may contain abstract methods (methods without a body). Subclasses must provide implementations for the abstract methods.

java

```
public abstract class MyAbstractClass {
    public abstract void display();
}

public class MyClass extends MyAbstractClass {
    @Override
    public void display() {
        System.out.println("Display method implemented");
    }
}
```

Summary

- **Access Modifiers**: Keywords that define the visibility and accessibility of classes, variables, methods, and constructors (`**private**`, `**default**`, `**protected**`, `**public**`).
- **Access Specifiers**: Broader term sometimes used to include access-related keywords like `**final**`, `**static**`, and `**abstract**`, along with traditional access modifiers.

Understanding and using access modifiers correctly is essential for encapsulating and protecting the data in your Java programs, promoting good design practices, and ensuring maintainability and

security.