Declaration & Access Modifiers in Java

Modifiers in Java define **visibility, accessibility, behavior, and constraints** of classes, variables, methods, and constructors.

Oracle certification exams **frequently test**:

- Which modifiers are allowed **where**
- **Illegal combinations**
- **Inheritance and access rules**
- Differences between **access modifiers vs non-access modifiers**

Classification of Modifiers in Java

Java modifiers are broadly classified into:

1. **Access Modifiers**
2. **Non-Access (Declaration) Modifiers**

1️⃣ Access Modifiers

Access modifiers control **who can access a class or class member**.

Java provides **four access levels**:

Modifier	Scope
`private`	Within the same class
(default)	Within the same package
`protected`	Package + subclasses

`public`	Everywhere

1.1 `private`

Characteristics

- Accessible **only within the same class**
- Most restrictive access level
- Used for **encapsulation**

```
class Test {
    private int x = 10;
}
```

❌ Not accessible outside the class, even in the same package.

Inheritance Impact

- **Not inherited**
- Subclasses cannot access private members directly

1.2 Default (Package-Private)

Characteristics

- No keyword used
- Accessible **only within the same package**

```
int x = 10;  // default access
```

✔ Visible to all classes in the same package
❌ Not accessible outside the package

Certification Note

> Default access is **package-level access**, not “friendly” or “public-like”.

1.3 `protected`

Characteristics

- Accessible within:
  - Same package
  - Subclasses in other packages (through inheritance)

```
protected int x = 10;
```

Important Inheritance Rule (Very High Exam Weight)

Outside the package:

- `protected` members can be accessed **only through subclass reference**, not parent reference

```
Parent p = new Child();
p.x;   // ❌ compile-time error (outside package)
```

✔ Correct:

```
Child c = new Child();
c.x;   // valid
```

1.4 `public`

Characteristics

- Accessible **from anywhere**
- Least restrictive

```
public int x = 10;
```

✔ Commonly used for APIs and entry points

Access Modifier Visibility Matrix (Must Memorize)

Modifier	Same Class	Same Package	Subclass (other pkg)	Everywhere
`private`	✔	❌	❌	❌
default	✔	✔	❌	❌
`protected`	✔	✔	✔	❌
`public`	✔	✔	✔	✔

2️⃣ Access Modifiers on Top-Level Classes

Allowed

- `public`
- default

```
public class Test {}
class Sample {}
```

❌ Not allowed:

- `private`
- `protected`

Reason: Top-level classes must be accessible at package or global level.

File Naming Rule (Exam Favorite)

```
public class Test {}
```

✔ File name must be:

```
Test.java
```

✔ At most **one public class per file**

3️⃣ Non-Access (Declaration) Modifiers

These modifiers define **behavior**, not accessibility.

3.1 `static`

Usage

- Variables
- Methods
- Blocks
- Nested classes

Static Variables

- One shared copy per class
- Stored in method area / metaspace
- Created during class loading

Static Methods

- Can access **only static members directly**
- Cannot use `this` or `super`

```
static void m1() {
    // this.x; ❌
}
```

3.2 `final`

Final Variables

- Value cannot be changed after initialization

```
final int x = 10;
```

✔ Must be initialized:

- At declaration
- In constructor
- In initializer block

Final Methods

- Cannot be overridden

```
final void m1() {}
```

Final Classes

- Cannot be inherited

```
final class Test {}
```

✔ Example: `String`, wrapper classes

Final vs Immutability (Important Clarification)

- `final` → reference cannot change
- Object **state may still change**

```
final StringBuilder sb = new StringBuilder("A");
sb.append("B"); // valid
```

3.3 `abstract`

Abstract Methods

- No implementation
- Must be overridden by subclass

```
abstract void m1();
```

Abstract Classes

- Cannot be instantiated
- May contain:
    - Abstract methods
    - Concrete methods
    - Variables

- Constructors

Invalid Combinations (Very High Exam Weight)

Combination	Reason
`abstract final`	Cannot override + cannot inherit
`abstract private`	Cannot be accessed to override
`abstract static`	Static methods cannot be overridden
`abstract synchronized`	Synchronization applies to implementation

3.4 `synchronized`

Purpose

- Thread safety
- Prevents concurrent access

```
synchronized void m1() {}
```

✔ Locks object (`this`) or class (for static)

3.5 `native`

Purpose

- Interface with non-Java code (C/C++)

```
native void m1();
```

✔ No method body
✔ Uses JNI

3.6 `strictfp`

Purpose

- Ensures **platform-independent floating-point calculations**

```
strictfp class Test {}
```

✔ Rare in modern Java
✔ Still exam-relevant

3.7 `transient`

Purpose

- Excludes variable from serialization

```
transient int x;
```

✔ Value not saved during object serialization

3.8 `volatile`

Purpose

- Ensures **visibility across threads**
- Prevents caching in thread-local memory

```
volatile boolean flag;
```

✔ Guarantees visibility
❌ Does NOT guarantee atomicity

4️⃣ Illegal Modifier Combinations (Must Memorize)

Combination	Valid?
`public private`	❌
`abstract final`	❌
`static abstract`	❌
`native synchronized`	❌
`private protected`	❌

5️⃣ Modifiers vs Program Elements

Element	Allowed Modifiers
Class	public, abstract, final, strictfp
Method	public, protected, private, static, final, abstract, synchronized, native, strictfp
Variable	public, protected, private, static, final, transient, volatile
Local Variable	final only
Constructor	public, protected, private

6️⃣ Default Access vs `protected` (Classic Confusion)

Feature	Default	Protected
Same package	✔	✔
Subclass (other pkg)	❌	✔
Inheritance-based	❌	✔

7️⃣ Modern Relevance

- Java 9+ modules add **module-level access control**
- Encapsulation is critical for:
    - API design
    - Security
    - Maintainability
- Oracle exams still focus on **core modifiers**, not modules

---

Certification Takeaways (Must Remember)

- Only `public` or default allowed for top-level classes
- `private` members are not inherited
- `protected` behaves differently across packages
- `final` affects inheritance and overriding
- `abstract` methods cannot be private or static
- `volatile` ≠ thread safety
- `transient` affects serialization only
- Illegal modifier combinations cause **compile-time errors**