# In Java, inner classes are classes defined within another class. They are used to logically group classes that are only used in one place, increase encapsulation, and sometimes improve readability and maintainability of code.

▯ Types of Inner Classes in Java
TypeDescription

1. Non-static Inner Class
   Instance class defined inside another class.
2. Static Nested Class
   Static class defined inside another class.
3. Local Inner Class
   Class defined inside a method or block.
4. Anonymous Inner Class
   Class without a name used for one-time use (usually with interfaces).

✅ 1. Non-static Inner Class (Regular Inner Class)
Can access all members (including private) of outer class.
Requires an object of the outer class to create an instance.
java
CopyEdit

```
class Outer {
int outerVar = 10;
```

```
class Inner {
    void show() {
        System.out.println("Outer variable: " + outerVar);
    }
}

void createInner() {
    Inner inner = new Inner();
    inner.show();
}
```

```
}

public class Main {
public static void main(String[] args) {
```

Outer outer = new Outer();
Outer.Inner inner = outer.new Inner();
inner.show();
}
}

✅ 2. Static Nested Class
Can only access static members of the outer class.
Does not require an outer class object.
java
CopyEdit
class Outer {
static int staticVar = 100;

```
static class StaticInner {
    void display() {
        System.out.println("Static variable: " + staticVar);
    }
}
```

}

public class Main {
public static void main(String[] args) {
Outer.StaticInner inner = new Outer.StaticInner();
inner.display();
}
}

✅ 3. Local Inner Class
Defined inside a method, block, or constructor.
Can access local variables if they are effectively final.
java
CopyEdit
class Outer {
void show() {
int localVar = 50; // must be effectively final

```
    class LocalInner {
        void display() {
            System.out.println("Local var: " + localVar);
        }
    }

    LocalInner inner = new LocalInner();
    inner.display();
}
```

```
}

public class Main {
public static void main(String[] args) {
new Outer().show();
}
}
```

✅ 4. Anonymous Inner Class
Used to override methods or implement interfaces on the fly.
Useful when you need to use a class only once.
java
CopyEdit

```
abstract class Animal {
abstract void sound();
}

public class Main {
public static void main(String[] args) {
Animal dog = new Animal() {
void sound() {
System.out.println("Bark");
}
};
```

```
    dog.sound();  // Output: Bark
}
```

```
}
```

▯ When to Use Inner Classes
To logically group classes only used in one place.
To implement helper classes that are tightly coupled with the outer class.
For event handling (e.g., in GUI code like Java Swing).
To encapsulate functionality without exposing it externally.

▯ Access Modifiers
Inner Class TypeCan Access Outer Class MembersNeeds Outer Class Instance
Non-static Inner Class
✅ Yes (all members)
✅ Yes
Static Nested Class
❌ Only static members
❌ No
Local Inner Class
✅ Yes (effectively final vars)
⚠ Only in method/block

Anonymous Inner Class
✅ Yes
⚠ Defined and used instantly generate a pdf this matter claer structured way with codes also

Below is a clear, well-structured layout on Java inner classes, including descriptions, code examples, usage scenarios, and concise comparison—formatting is suitable for a PDF document.

## Inner Classes in Java

Java inner classes are classes defined within another class. They provide logical grouping, encapsulation, and can improve code clarity[1] [2] [3].

## Types of Inner Classes

| Type | Description |
|---|---|
| **Non-static Inner** | Regular instance class within another class. |
| **Static Nested** | Static class within another class (like a static member). |
| **Local Inner** | Class defined inside a method or code block. |
| **Anonymous Inner** | Nameless class, typically used for one-off implementations. |

## 1. Non-static Inner Class (Member Inner Class)

- Can access all members (including private) of the outer class.
- Requires an instance of the outer class to instantiate.

```java
class Outer {
    int outerVar = 10;

    class Inner {
        void show() {
            System.out.println("Outer variable: " + outerVar);
        }
    }
}

public class Main {
    public static void main(String[] args) {
        Outer outer = new Outer();
        Outer.Inner inner = outer.new Inner();
        inner.show(); // Output: Outer variable: 10
    }
}
```

## 2. Static Nested Class

- Can only directly access static members of the outer class.
- Can be instantiated without an outer class object.

```
class Outer {
    static int staticVar = 100;

    static class StaticInner {
        void display() {
            System.out.println("Static variable: " + staticVar);
        }
    }
}

public class Main {
    public static void main(String[] args) {
        Outer.StaticInner inner = new Outer.StaticInner();
        inner.display(); // Output: Static variable: 100
    }
}
```

## 3. Local Inner Class

- Defined within a method, block, or constructor.
- *Can access local variables only if they are effectively final.*

```
class Outer {
    void show() {
        int localVar = 50; // must be effectively final

        class LocalInner {
            void display() {
                System.out.println("Local var: " + localVar);
            }
        }

        LocalInner inner = new LocalInner();
        inner.display(); // Output: Local var: 50
    }
}

public class Main {
    public static void main(String[] args) {
        new Outer().show();
    }
}
```

### 4. Anonymous Inner Class

- No class name; used for one-time implementations (usually interfaces or abstract classes).
- Syntax links class definition and instantiation.

```
abstract class Animal {
    abstract void sound();
}

public class Main {
    public static void main(String[] args) {
        Animal dog = new Animal() {
            void sound() {
                System.out.println("Bark");
            }
        };
        dog.sound();  // Output: Bark
    }
}
```

### When to Use Inner Classes

- When grouping logically related classes used only in one place.
- For helper/util classes that tightly integrate with the outer class.
- In event handling (e.g., GUI programming).
- To encapsulate implementation details[4] [2].

### Access Considerations

| Inner Class Type | Access to Outer Class Members | Needs Outer Class Instance |
|---|---|---|
| Non-static Inner | Yes (all members) | Yes |
| Static Nested | Only static members | No |
| Local Inner | Yes (effectively final locals only) | Only within method/block |
| Anonymous Inner | Yes | Instantiated instantly |

### Key Points

- Inner classes **enhance encapsulation** and code organization[1] [2] [3].
- Use only when the inner class is tightly connected to the outer class.
- Anonymous and local classes are often used for event handlers and concise one-time implementations.

This structure is ready for PDF generation. If you need it as a formatted PDF file, use this layout in your document tool of choice (MS Word, Google Docs, LaTeX, etc.), then export/save as PDF.

⁂

1. https://www.geeksforgeeks.org/java/inner-class-java/
2. https://www.codingshuttle.com/java-programming-handbook/inner-classes-in-java
3. https://data-flair.training/blogs/java-inner-class/
4. https://stackoverflow.com/questions/11398122/what-are-the-purposes-of-inner-classes