Anonymous Classes in Java

In Java, **anonymous classes** allow you to create a one-time-use class without explicitly defining a named class. These classes are ideal for simplifying code when you need to extend a class or implement an interface for a specific use case.

---

1. What is an Anonymous Class?

An **anonymous class** is an inner class without a name. It is created and instantiated simultaneously, usually to provide a specific implementation for an abstract class or interface.

Key Features

1. **No Explicit Name:** The class does not have a name.
2. **One-Time Use:** Primarily used for a single instance.
3. **Direct Instantiation:** Created using the `new` keyword.

---

2. Syntax of Anonymous Class

```
new SuperClassOrInterface() {
    // Overridden methods or new implementations
};
```

Example: Implementing an Interface

```
Runnable r = new Runnable() {
    @Override
    public void run() {
        System.out.println("Running using an anonymous clas
s!");
```

```
    }
};
```

3. Why Use Anonymous Classes?

1. **Compact Code:** Simplifies code by eliminating the need to create a separate named class.
2. **Specialized Behavior:** Provides unique implementations for interfaces or abstract classes for specific cases.
3. **Convenience:** Useful for GUI event handling, thread creation, or callback implementations.

4. Example Scenarios of Anonymous Classes

Example 1: Extending a Class

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

public class AnonymousExample {
    public static void main(String[] args) {
        Animal dog = new Animal() { // Anonymous class extend
ing Animal
            @Override
            void sound() {
                System.out.println("Woof Woof!");
            }
        };

        dog.sound(); // Outputs: Woof Woof!
    }
}
```

Example 2: Implementing an Interface

```
interface Greeting {
    void sayHello();
}

public class AnonymousExample {
    public static void main(String[] args) {
        Greeting greeting = new Greeting() { // Anonymous cla
ss implementing Greeting
            @Override
            public void sayHello() {
                System.out.println("Hello, world!");
            }
        };

        greeting.sayHello(); // Outputs: Hello, world!
    }
}
```

Example 3: Using for Event Handling (GUI Example)

```
import javax.swing.*;

public class ButtonClickExample {
    public static void main(String[] args) {
        JButton button = new JButton("Click Me");

        // Adding an ActionListener using an anonymous class
        button.addActionListener(new java.awt.event.ActionLis
tener() {
            @Override
            public void actionPerformed(java.awt.event.Action
Event e) {
                System.out.println("Button clicked!");
```

```
            }
        });

        JFrame frame = new JFrame("Anonymous Class Example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.add(button);
        frame.setSize(200, 200);
        frame.setVisible(true);
    }
}
```

5. Limitations of Anonymous Classes

1. **One-Time Use Only:** Cannot be reused, as they have no name.
2. **Code Complexity:** If the implementation is large, it can make code harder to read.
3. **Cannot Define Constructors:** Since the class is unnamed, you cannot define constructors.

6. Practical Use Cases

1. **Thread Creation**

```
Thread thread = new Thread(new Runnable() {
    @Override
    public void run() {
        System.out.println("Thread running using an anonym
ous class!");
    }
});
thread.start();
```

2. **Callbacks**

```
interface Callback {
    void execute();
}

public class CallbackExample {
    public static void main(String[] args) {
        performAction(new Callback() {
            @Override
            public void execute() {
                System.out.println("Callback executed!");
            }
        });
    }

    static void performAction(Callback callback) {
        callback.execute();
    }
}
```

3. **GUI Programming**
    - Common in Swing or JavaFX applications for event handling.

---

7. Anonymous Class vs Lambda Expressions

Differences

Feature	Anonymous Class	Lambda Expression
Syntax	More verbose	Concise
Type	Can extend a class or implement an interface	Can only implement functional interfaces
Behavior	Allows methods and fields in the implementation	Cannot define methods or fields
Backward Compatibility	Works with all Java versions	Introduced in Java 8

8. Common Mistakes

1. **Overcomplicating Code:**
   Avoid using anonymous classes for complex logic, as they can make the code harder to maintain.
2. **Misusing in Non-Reusable Contexts:**
   If the logic is reusable, prefer creating a named class.

9. Practice Problems

1. Create an anonymous class to sort an array of integers in descending order.
2. Implement an anonymous class for a button click listener in a Swing application.
3. Use an anonymous class to define custom behavior for a callback interface.

By understanding **anonymous classes**, you'll be able to write more concise and effective Java code, especially when dealing with interfaces, abstract classes, or quick event handling.