能不能举一个具体点的例子 直观让我感受他们的区别?

为了更加直观地展示 Java 中 **成员内部类、静态嵌套类、局部内部类** 和 **匿名内部类** 的区别，我们可以通过一个完整的示例来说明它们的定义方式、访问权限、实例化方式以及实际用途。

示例结构说明

我们将定义一个外部类 `OuterClass`，它包含：

- 一个私有变量 `private int outerNum = 10;`
- 一个静态变量 `private static int staticNum = 20;`
- 四种内部类：
  - 成员内部类（MemberInner）
  - 静态嵌套类（StaticNested）
  - 局部内部类（LocalInner，在方法中定义）
  - 匿名内部类（用于实现接口）

完整示例代码

Java

```
public class OuterClass {
    private int outerNum = 10;
    private static int staticNum = 20;

    // ---------------------------
    // 1. 成员内部类（Member Inner Class）
    // ---------------------------
    class MemberInner {
        void accessOuter() {
            System.out.println("MemberInner 访问外部类的非静态字段: " + outerNum);
            System.out.println("MemberInner 访问外部类的静态字段: " + staticNum);
        }
    }

    // ---------------------------
    // 2. 静态嵌套类（Static Nested Class）
    // ---------------------------
    static class StaticNested {
        void accessStaticOuter() {
            System.out.println("StaticNested 访问外部类的静态字段: " + staticNum);
            // ✗ 不能访问非静态字段 outerNum
        }
    }

    // ---------------------------
    // 3. 局部内部类（Local Inner Class）
    // ---------------------------
    void method() {
        final int localVar = 30;

        class LocalInner {
            void printLocal() {
                System.out.println("LocalInner 访问方法中的 final 变量: " + localVar);
                System.out.println("LocalInner 访问外部类的静态字段: " + staticNum);
            }
        }

        LocalInner local = new LocalInner();
        local.printLocal();
    }
```

⍰ 输出结果（运行后）

```
 MemberInner 访问外部类的非静态字段: 10
MemberInner 访问外部类的静态字段: 20
StaticNested 访问外部类的静态字段: 20
LocalInner 访问方法中的 final 变量: 30
LocalInner 访问外部类的静态字段: 20
AnonymousInnerClass: 按钮被点击了
```

❓ 总结对比

类型	是否需要外部类实例	能否访问外部类的非静态字段	能否访问外部类的静态字段	作用
成员内部类	✅ 是	✅ 是	✅ 是	外部类内部
静态嵌套类	❌ 否	❌ 否	✅ 是	外部类内部
局部内部类	✅ 是（在方法内）	✅ 是（需外部类实例）	✅ 是	方法代码块内部
匿名内部类	✅ 是（隐式）	✅ 是	✅ 是	创建时的上下文

✅ 使用建议

- **成员内部类**：适合需要频繁访问外部类成员的情况，但注意内存泄漏风险。
- **静态嵌套类**：适合逻辑独立、不需要外部类实例的场景。
- **局部内部类**：适合封装在方法内部的逻辑，增强代码封装性。
- **匿名内部类**：适合一次性使用的回调、监听器等，简化代码结构。

通过这个示例，你可以清晰地看到每种内部类的定义方式、访问权限、使用方式以及它们之间的区别。这种结构化的对比方式有助于你更好地理解和选择合适的内部类类型。