```
private bool AddDept(string name)
{
    bool op = false;
    string connStr = "Data Source=.;Initial Catalog=HomeWorke;Integrated
Security=True;";
    try
    {
        using (SqlConnection con = new SqlConnection(connStr))
        {
            con.Open();
            string sql = "INSERT INTO dbo.Department (dname) VALUES (@name)";
            using (SqlCommand cmd = new SqlCommand(sql, con))
            {
                // 明确指定类型和长度，避免 AddWithValue 的潜在问题
                cmd.Parameters.Add("@name", SqlDbType.NVarChar, 100).Value =
name ?? (object)DBNull.Value;

                int rows = cmd.ExecuteNonQuery();
                op = rows > 0;
            }
        }
    }
    catch (SqlException ex)
    {
        MessageBox.Show("数据库异常: " + ex.Message);
    }
    return op;
}
```

整体代码如上

```
private bool AddDept(string name)
```

定义一个私有bool类型方法“AddDept”————对应添加部门的业务逻辑
入参string name 要插入的部门名称
bool是为了告诉调用方“插入操作失败/成功”

private 意味着这个方法只有在这个窗体[1]中才能调用，是类的内部实现细节，对外隐藏。

```
 bool op = false;
```

定义布尔变量op并初始化为false，作为操作结果的标记。在编程逻辑里，“先假定失败，操作成功后再置为true”是最稳妥的设计——因为数据库操作可能因网络、权限、语法等问题失败，默认失败能避免“未执行操作却返回成功”的逻辑漏洞。

异常捕获的try模块

核心数据库操作

```
try
    {
        using (SqlConnection con = new SqlConnection(connStr))
        {
            con.Open();
            string sql = "INSERT INTO dbo.Department (dname) VALUES (@name)";
            using (SqlCommand cmd = new SqlCommand(sql, con))
            {
                // 明确指定类型和长度，避免 AddWithValue 的潜在问题
                cmd.Parameters.Add("@name", SqlDbType.NVarChar, 100).Value =
name ?? (object)DBNull.Value;

                int rows = cmd.ExecuteNonQuery();
                op = rows > 0;
            }
        }
    }
    catch (SqlException ex)
    {
        MessageBox.Show("数据库异常: " + ex.Message);
    }
```

try包裹可能出错的代码
catch(SqlException)：专门捕获数据库相关的异常
并通过MessageBox把错误信息传递给用户

using(SqlConnection con=...)
using语句是C#的资源自动释放语法。
关闭连接、释放内存，避免手写conn.Close()导致的资源泄露；

SqlConnection是“非托管资源”(占用数据库连接池、系统句柄)，如果手动忘记关闭，会导致数据库连接泄露，最终耗尽连接池。
using会在代码块执行完毕后，自动调用con.Dispose()释放资源，是工业级代码的标准写法。

con.Open()：主动打开数据库连接。SqlConnection实例化后默认是关闭状态，必须调用Open()才能与数据库建立通信通道。

参数化SQL与命令执行

```
string sql = "INSERT INTO dbo.Department (dname) VALUES (@name)";
using (SqlCommand cmd = new SqlCommand(sql, con))
{
    cmd.Parameters.Add("@name", SqlDbType.NVarChar, 100).Value = name ??
(object)DBNull.Value;
    int rows = cmd.ExecuteNonQuery();
    op = rows > 0;
}
```

这是整个方法的核心业务逻辑，是工业级代码与新手代码的核心区别

1. INSERT语句设计：
   因为数据库中did是自增列，所以SQL语句只插入dname列；@name是参数占位符，而非直接拼接字符串，这是杜绝SQL注入攻击的核心手段.
   （新手常犯的错误是用string.Format拼接SQL，会被黑客利用输入恶意SQL语句篡改/删除数据）。
2. SqlCommnd的作用：
   是“SQL执行器”，接收两个参数要执行的SQL语句、已打开的数据库连接，负责把SQL指令发送给数据库并接收执行结果。
3. 参数赋值的细节：
   `cmd.Parameters.Add("@name", SqlDbType.NVarChar, 100).Value =`
   显示指定参数类型`SqlDbType.NVarChar`和长度100
   避免AddWithValue的“类型推断漏洞”（）

`name ?? (object)DBNull.Value;`：空值处逻辑 C＃中的若干问题中详解
如果用户输入的name是NULL，直接传入null会抛出异常。

？？ 是C#的空合并运算，“如果name值为NULL，就用DBNull.Value替代
（DBNull.Value是数据库认可的空值类型）

4. ExecuteNonQuery():
   执行非查询类SQL （ INSERT / UPDATE / DELETE ）返回值是受影响的行数。
   eg. 插入一条数据，插入成功，返回1，失败返回0。
5. op=rows>0
   根据受影响的行数判断操作是否成功。

---

1. 本项目是DeptAddForm窗体类 ↩