

Main method explanation



```
package com.sunbeam;

import java.util.Scanner;

public class Program {

    public static void main1(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number :");
        int num = sc.nextInt();
        System.out.println("Square : "+ num * num);
        sc.close();
    }

    public static void main2(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number : ");
        int num = sc.nextInt();
        System.out.println("Square : "+ num * num);
        sc.close();
        System.out.println("Resource closed...");
    }

    public static void main3(String[] args) {
        Scanner sc = new Scanner(System.in);
        try {
            System.out.print("Enter the number : ");
            int num = sc.nextInt();
            System.out.println("Square : "+ num * num);
            sc.close();
        }
        finally { // Finally compulsarily gets executed
            sc.close();
            System.out.println("Resource closed..");
        }
    }

    public static void main4(String[] args) {
        // try-resource ---- java 7.0
        try(Scanner sc = new Scanner(System.in)){
            System.out.print("Enter the number : ");
            int num = sc.nextInt();
            System.out.println("Square : "+ num * num);
        } // sc.close() will be called automatically
    }

    public static void main5(String[] args) {
```

```
// custom-resource class
class MyResource implements AutoCloseable{

    public MyResource() {
        System.out.println("MyResource created");
    }
    @Override
    public void close() {
        System.out.println("MyResource closed..");
    }

}

try(MyResource myResource = new MyResource()){
    System.out.println("MyResource used...");
} //myResource.close() called automatically;

}

}
```

explain me all the main methods and working

Perfect, this code is a **demonstration of different ways of handling resources** (like `Scanner`, files, or custom resources) in Java. Let's go method by method.

1. main1

```
java

public static void main1(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter the number : ");
    int num = sc.nextInt();
    System.out.println("Square : "+ num * num);
    sc.close();
}
```

✓ Working:

- Creates a `Scanner` object to take input.
- Reads an integer and prints its square.
- Closes the scanner at the end.

⚠ Problem: If an exception occurs before `sc.close()`, then `close()` will **not be executed**, and the resource may remain open (resource leak).

2. main2

```
java

public static void main2(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter the number : ");
    int num = sc.nextInt();
    System.out.println("Square : "+ num * num);
    sc.close();
    System.out.println("Resource closed...");
}
```

✓ Working:

- Same as `main1`, but prints an **extra message** "Resource closed..." after `sc.close()`.

⚠ **Problem:** Same issue as above → If exception occurs before `sc.close()`, resource may not be closed.

3. main3

```
java

public static void main3(String[] args) {
    Scanner sc = new Scanner(System.in);
    try {
        System.out.print("Enter the number : ");
        int num = sc.nextInt();
        System.out.println("Square : "+ num * num);
        sc.close();
    }
    finally {
        sc.close();
        System.out.println("Resource closed..");
    }
}
```

✓ Working:

- Resource (`Scanner`) is created.
- Inside `try`, number is read and square is printed.
- In the `finally` block, `sc.close()` is called again.

⚠ Issues here:

- **Redundant `sc.close()`:** It is already inside `try`, so calling again in `finally` will cause `IllegalStateException` (because `Scanner` is already closed once).
 - Correct usage would be → remove `sc.close()` from `try`, only keep it in `finally`.
-

4. main4

```
java

public static void main4(String[] args) {
    try(Scanner sc = new Scanner(System.in)){
        System.out.print("Enter the number : ");
        int num = sc.nextInt();
        System.out.println("Square : "+ num * num);
    }
```

```
}
}
```

✓ Working:

- This is **try-with-resources** (introduced in Java 7).
- Resource **Scanner** is created inside **try()**.
- After try block execution (whether normal or exception), the resource is **automatically closed**.
- Equivalent to:

```
java

Scanner sc = new Scanner(System.in);
try {
    // use sc
} finally {
    sc.close();
}
```

💡 **Best way** for resources like **Scanner**, **FileReader**, **BufferedReader**, etc.

5. main5

```
java

public static void main5(String[] args) {
    class MyResource implements AutoCloseable{
        public MyResource() {
            System.out.println("MyResource created");
        }
        @Override
        public void close() {
            System.out.println("MyResource closed..");
        }
    }
    try(MyResource myResource = new MyResource()){
        System.out.println("MyResource used...");
    }
}
```


✓ Working:

- Defines a **custom resource class** **MyResource** implementing **AutoCloseable**.
- Any class that implements **AutoCloseable** can be used inside try-with-resources.
- Execution order:
 1. **MyResource created** (when object is created).
 2. **MyResource used...** (inside try block).
 3. **MyResource closed..** (automatically called at end of try).

💡 This shows how we can define **our own resource type** that will automatically release/cleanup (like DB connection, file handles, sockets, etc.).

Summary

- `main1` & `main2`: Manual resource closing (not safe if exceptions occur).
 - `main3`: Uses `finally` to ensure closing → but code is wrong due to double `close()`.
 - `main4`: **Best practice** → `try-with-resources` (automatic close).
 - `main5`: Demonstrates custom resources with `AutoCloseable`.
-

 Do you want me to **fix `main3` properly** (so it works safely without double closing)?