Learning Objectives: Variable Scope

- **Differentiate between global and local scope**
- **Explain the purpose of the keyword “static”**
- **Explain what the keyword “final” does**

Local Scope

Local Scope

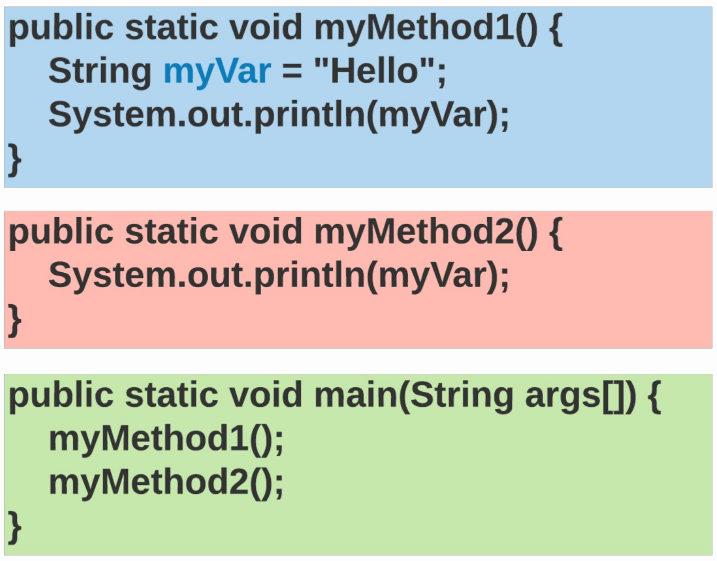
Take a look at the code below. The first method declares the variable `myVar` and then prints it. The second method also prints `myVar`. What do you think the output will be?

```
public static void myMethod1() {
  String myVar = "Hello";
  System.out.println(myVar);
}

public static void myMethod2() {
  System.out.println(myVar);
}

public static void main(String args[]) {
  myMethod1();
  myMethod2();
}
```

Java returns an error such as `error: cannot find symbol` at the line containing `System.out.println(myVar);` within the second method. This happens because variables declared inside a method have **local** scope. Variables with local scope can **only** be used within that method. Outside of that method, those local variables cannot be accessed. In the image below, the light blue box represents the scope of `myVar`. Since `myMethod2` (denoted in a light red box) is outside the scope of `my_var`, an error occurs.



```java
public static void myMethod1() {
    String myVar = "Hello";
    System.out.println(myVar);
}
```

```java
public static void myMethod2() {
    System.out.println(myVar);
}
```

```java
public static void main(String args[]) {
    myMethod1();
    myMethod2();
}
```

.guides/img/LocalScope

challenge

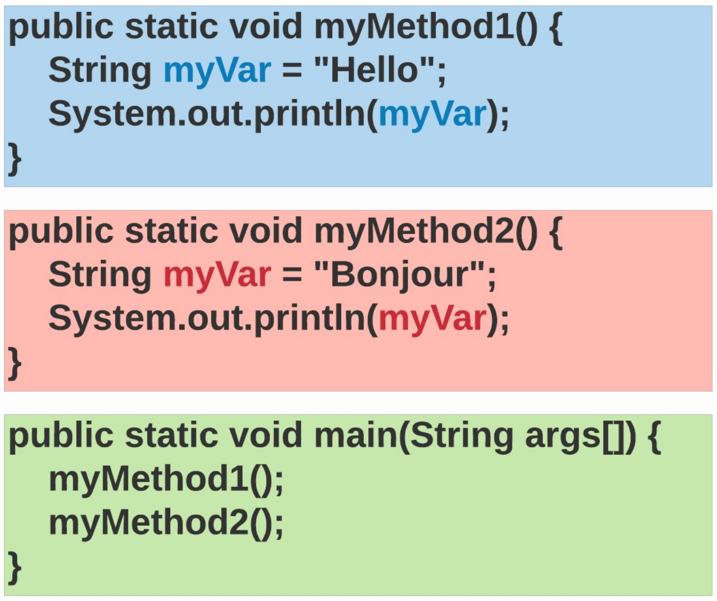
What happens if you:

- Change `myMethod2` to look like this:

```java
public static void myMethod2() {
  String myVar2 = "Hello";
  System.out.println(myVar2);
}
```

More Local Scope

Each method has its own local scope. That means you can declare two variables with the same name as long as they are in separate methods. The blue `myVar` exists only in the light blue box, and the red `myVar` exists only in the light red box. The boundaries of local scope keep Java from overwriting the value of the first variable with the contents of the second.



```java
public static void myMethod1() {
    String myVar = "Hello";
    System.out.println(myVar);
}
```

```java
public static void myMethod2() {
    String myVar = "Bonjour";
    System.out.println(myVar);
}
```

```java
public static void main(String args[]) {
    myMethod1();
    myMethod2();
}
```

.guides/img/LocalScope2

```java
public static void myMethod1() {
  String myVar = "Hello";
  System.out.println(myVar);
}

public static void myMethod2() {
  String myVar = "Bonjour";
  System.out.println(myVar);
}

public static void main(String args[]) {
  myMethod1();
  myMethod2();
}
```

challenge

What happens if you:

- Declare `myMethod3()` as:

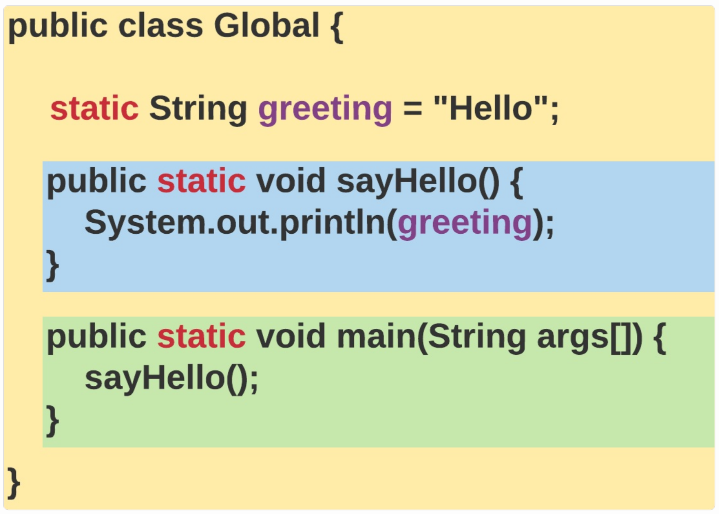
```java
public static void myMethod3() {
  String myVar = "Hola";
  System.out.println(myVar);
}
```

and call it by including `myMethod3();` within the `main()` method.

Global Scope

Global Scope - Referencing Variables

When a variable is declared inside a method, it has local scope. When a variable is declared in the main program within **class**, it has global scope. Global variables are declared outside of methods, but can be referenced inside a method. Look at the image below. The yellow block holds everything within the program including the variable `greeting`. This enables all methods within the program to access that variable since it is considered to be **global**. Copy and paste the code below and then click `TRY IT`.



```
public class Global {

  static String greeting = "Hello";

  public static void sayHello() {
     System.out.println(greeting);
  }

  public static void main(String args[]) {
     sayHello();
  }

}
```

.guides/img/GlobalScope

```
static String greeting = "Hello";

public static void sayHello() {
  System.out.println(greeting);
}

public static void main(String args[]) {
  sayHello();
}
```

Notice how the keyword `static` is used. This is due to the fact that both the `sayHello()` and `main()` methods are `static` methods. Only **static** variables can be accessed by static methods.

challenge

What happens if you:

- Remove the keyword `static` from `static String greeting = "Hello";`?

Global Scope - Modifying Variables

Once a global variable (whether static or not) becomes available, a method can modify the content of that variable as needed.

```java
static String greeting = "Hello";

public static void sayHello() {
  greeting = "Bonjour";
  System.out.println(greeting);
}

public static void main(String args[]) {
  sayHello();
}
```

challenge

What happens if you:

- Change `greeting = "Bonjour";` within `sayHello()` to `greeting = "Hola";`?

Global vs. Local Scope

Global vs. Local Scope

If a variable is declared and initialized both locally and globally, that variable will retain its content depending on how it is used. In the example below, `myVar` is declared and initialized globally as `global scope` and locally as `local scope`. Since the variable has differing scopes, it retains its value when called or printed.

```
static String myVar = "global scope";

public static void printScope() {
  String myVar = "local scope";
  System.out.println(myVar);
}

public static void main(String args[]) {
  printScope();
  System.out.println(myVar);
}
```

The exception to this rule is when a method modifies a global variable. In such a case, the content of the global variable is changed.

```
static String myVar = "global scope";

public static void printScope() {
  myVar = "local scope";
  System.out.println(myVar);
}

public static void main(String args[]) {
  printScope();
  System.out.println(myVar);
}
```

challenge

What happens if you:

- Change the code to:

```
static String myVar = "global scope";

public static void printScope(String myVar) {
  myVar = "local scope";
  System.out.println(myVar);
}

public static void main(String args[]) {
  printScope(myVar);
  System.out.println(myVar);
}
```

When a global variable is also a method parameter, it is considered to be the same as if the method declared and initialized its own local variable. In this case, the variable has both a local and global scope and will retain its value depending on its scope.

The “final” Keyword

If you want a global variable to remain unchanged throughout the program, you can declare the variable as `final`. `final` variables are also referred to as **constants**. Constants never change and are denoted in all uppercase and underscores (_).

```
static final String MY_VAR = "I NEVER CHANGE";

public static void printScope() {
  MY_VAR = "I CAN'T CHANGE";
  System.out.println(MY_VAR);
}

public static void main(String args[]) {
  printScope();
  System.out.println(MY_VAR);
}
```

challenge

What happens if you:

- Remove the keyword `final` from the code?