Identifier Scope Within a Class

Definitions & Facts

- Local identifier:
  - identifier declared within a method or block,
  - it is **visible** only within that method or block.

Definitions & Facts

- Block:
  - a set of statements enclosed within braces
  - the body of a loop or an if statement also forms a block
- Nesting:
  - Blocks can be nested:
    - one block can contain other blocks
  - Methods can NOT be nested:
    - you cannot include the definition of one method in the body of another method.

Declaration of an identifier

- **Within a class (to be accessible by the whole class):**
  - it can occur **anywhere outside** of every method definition (and every block), even if it is in the middle or at the end of the class.
- **Within a method/block**:
  - it must occur before it can be used
  - when blocks are nested:
    - an identifier used to name a variable in the outer block **cannot** be used to name any other variable in an inner block of the same method
    - (see example - illegal)

Example - illegal


```
public static void illegalIdentifierDeclaration()
{
    int x;

    //block
    {
        double x;     //illegal declaration,
                      //x is already declared
          ...
    }
}
```

Scope Rules within a Method/Block

- An identifier declared within a method/block is accessible:
  - Only within the block from the point at which it is declared until the end of the block.
  - By inner blocks; blocks that are nested within that outer block.
- An identifier declared within a method/block **overrides** any identifier with the same name declared outside the method. (see override example)

Scope Rules within a Class

- Identifiers declared within a class, and outside every method/block:
  - These identifiers can be static or non-static.
  - Static identifiers are accessible by **all** methods provided that the methods do not have any other identifier with the same name. (see override example)
  - Non-static identifiers are accessible **only** by non-static methods provided that the methods do not have any other identifier with the same name.

Example - override


```
static int x = 0;
       int y = 0;

public static void overrideIdentifierDeclaration()
{    x++;        // legal, increments static x
     double x;   // legal declaration,
                 // method will NOW see local x (double)
                 // method will NOT see global static x anymore
     y ++;       // illegal because y is non-static,
                 //         but the method is static
}

public void nonStaticMethodScopeExample()
{     x++;      // legal, increments global static x
      y++;      // legal, because this is a non-static method
}
```

Scope Rules (continued)

- Suppose X is an identifier declared within a class and outside of every method's definition (block)
  - If X is declared **without** the reserved word `static` (such as a named constant or a method name), then it **cannot** be accessed in a **static method**
  - If X is declared **with** the reserved word `static` (such as a named constant or a method name), then it **can** be accessed within **any method** (block), provided the method (block) does not have any other identifier named X

Scope Rules (continued) Example 7-11

```
public class ScopeRules
{
    static final double rate = 10.50;
    static int z;
    static double t;
    public static void main(String[] args)
    {
        int num;
        double x, z;
        char ch;
        //...
    }

    public static void one(int x, char y)
    {
        //...
    }
    public static int w;
    public static void two(int one, int z)
    {
        char ch;
        int a;

        { // block three
            int x = 12;
          }
        //...
    }
}
```

ONE	TWO	THREE	MAIN
rate	rate	rate	rate
z	~~z~~	~~z~~	~~z~~
t	t	t	t
main	main	main	main
			args
			num
			x
			z
			ch
one	one	one	one
x			
y			
w	w	w	w
two	two	two	two
	one	one	
	z	z	
	ch	ch	
	a	a	
		x	

Scope Rules: Demonstrated

TABLE 7-3 Scope (Visibility) of the Identifiers

Identifier	Visibility in `one`	Visibility in `two`	Visibility in block `three`	Visibility in `main`
`rate` (before `main`)	Y	Y	Y	Y
`z` (before `main`)	Y	N	N	N
`t` (before `main`)	Y	Y	Y	Y
`main`	Y	Y	Y	Y
local variables of `main`	N	N	N	Y
`one` (method name)	Y	Y	Y	Y
`x` (`one`'s formal parameter)	Y	N	N	N

Scope Rules: Demonstrated (continued)

TABLE 7-3 Scope (Visibility) of the Identifiers (continued)

Identifier	Visibility in `one`	Visibility in `two`	Visibility in block `three`	Visibility in `main`
`y` (`one`'s formal parameter)	Y	N	N	N
`w` (before method `two`)	Y	Y	Y	Y
`two` (method name)	Y	Y	Y	Y
`one` (`two`'s formal parameter)	N	Y	Y	N
`z` (`two`'s formal parameter)	N	Y	Y	N
local variables of `two`	N	Y	Y	N
`x` (Block `three`'s local variable)	N	N	Y	N

Scope Rules – Scanner Example

```
public class ScopeRules {
 static Scanner read=new Scanner (System.in);
 public static void main(String[] args){
      int number = read.nextInt();
        //…    }
 }
```

Can be accessed from **any** method (static & non-static) including main

```
public class ScopeRules {
 public static void main(String[] args){
    Scanner read=new Scanner (System.in);
    int number = read.nextInt();
        //…    }
 }
```

'Local' accessed from main **only**

```
public class ScopeRules {
   Scanner read=new Scanner (System.in);
 public static void main(String[] args){
      int number = read.nextInt();
        //…    }
 }
```

Can be accessed from non-static method only