While Loop:

- Python

```
while expression:
    statements
```

- Example

```
while i < 10:
    f = f * i
    i += 1
```

- Java

```
while (expression)
    statement or block
```

- Example

```
while (i < 10) {
    f = f * i;
    ++i;
}
```

- Common Problem:
  - Putting a semi-colon after the boolean expression:

    ```
    while ( number != -1 ); {
    ```

  - This will cause the compiler to loop over an empty statement (do nothing each time through the loop).

<u>Do/While Loop</u>:

- The condition for the **`do`**/**`while`** loop is evaluated at the end of the loop.
- The **`do`**/**`while`** loop executes the loop body <u>at least once</u>.
  - Compare to the **`while`** loop which might not execute the loop body at all.
- Syntax for do/while loop:

```
// initialize variables
do {
   // body of the loop goes here
   // as many lines of code as needed
} while ( boolean expression );
// process the results
```

Unlike the **`while`** loop, the **`{ }`**'s are <u>always</u> required.

The loop body can be empty. (But, the **`{ }`**'s are still required.)

do/while Example:

- Consider a guessing program that generates a random integer then accepts guesses until the user gets it right.

```
import java.util.*;

public class GuessingInts {
  public static void main(String[] args) {
    Random guess = new Random();
    Scanner inputScan = new Scanner( System.in );
    int userNumber, numGuesses = 0;
    int findIt = guess.nextInt(20) + 1;

    do {
      System.out.print("Guess a value between 1 and 20: ");
      userNumber = inputScan.nextInt();
      numGuesses++;
      if ( userNumber > findIt )
        System.out.println("Too large\n");
      if ( userNumber < findIt )
        System.out.println("Too small\n");
    } while ( userNumber != findIt );

    System.out.println("Got it!");
    System.out.println("Took you " + numGuesses + " attempts.");
  } // end of method main
} // end of class GuessingInts
```

for Loop Syntax:

- Syntax of the **`for`** loop:

  ```
  for ( initialization;  loop condition;  loop update ) {
     // loop body
     // goes here
  }
  ```

  - Semi-colons separate terms in the loop header.
  - There is not a semi-colon after the loop headers.
  - The **`{ }`**'s are required only if more than one statement is in the loop body.
- The **`for`** loop flow of control:
  - The initialization statement is executed exactly once.
  - Repeat: The loop condition is evaluated. If the condition is true:
    - The loop body is executed.
    - The loop update is executed.
  - When the loop condition is evaluated as false:
    - Continue with code after the **`for`** loop.

<u>Example</u>:

- Consider a **`for`** loop that prints the even numbers from **`0`** to **`20`**:

```
int i;
for (i = 0;  i <= 20;  i = i + 2 ) {
   System.out.print(i + "  ");
}
System.out.println();
```

- Can this be written as a while loop?

```
int i = 0;
while (i <= 20) {
   System.out.print(i + " ");
   i = i + 2;
}
System.out.println();
```

- A for loop may always be written as a while loop.

for Loop Syntax:

- Remember:

```
for ( initialization;  loop condition;  loop update ) {
   // loop body
   // goes here
}
```

  - Note the initialization and the loop update can have more than one statement.
    - Multiple initialization or loop update statements are separated by commas.
- Example:

```
for (i = 1, j = 2; i <= 10 && j <= 30; i += 1, j += i) {
```

- It is not good programming style to include statements that are not part of the loop control.

for Loop Syntax:

- It is not good programming style to include statements that are not part of the loop control. For example:

```
int i;
System.out.println("The even integers are:");
for (i = 0; i <= 20; i += 2)
  System.out.print(i + " ");
```

- Can be, but should NOT be written as:

```
for (System.out.println("The even integers are:"), int i = 0;
     i <= 20; System.out.print(i + " "), i +=2) ;
```

for Loop Syntax:

- Also, the statements can be blank, for example:

```
for ( ; i < 10 ; ) {  //not good style
```

- Is the same as

```
while (i < 10) {
```

- Also

```
for ( ; ; ) {   //but I would never write this
```

- Is the same as

```
while (true) {
```

break

- Just like in Python, **break** will cause you to exit from the innermost loop.

```
while ( expr ) {
   // loop body
   break; // causes you to exit from loop body
}
```

  - This works for all loop types (while, for, do/while).

continue

- Just like in Python, continue will cause you to go immediately to the end of the loop body, performing the .

```
while ( expr ) {
   // loop body
   continue; // causes you to immediately go to the top
             // and re-evaluate exp
}
```

  - This also works for all loop types (while, for, do/while).
  - Note when used with for loops, when a continue statement is executed, the loop update statements are executed next, and then the expression evaluated. For example the following (poor) code prints out the even numbers between 0 and 10.

```
for (i = 0; i <= 20; ++i) {
  if (i % 2 == 1) continue;
  System.out.println(i + " ");
}
```

break and continue:

- Note that break and continue are never needed. Any code that uses them can be rewritten without them.
- Some programmers consider their use to be bad style. They can be used, but use them only when it makes the code clearer.
- Generally I use break for special cases (like receiving bad data).

example

- This code uses a break. Some would say it makes the reason you exit the loop not as clear as it should be.

```
while(years <= 100) {
  balance += payment;
  double interest = balance * interestRate/100;
  balance += interest;
  if (balance >= goal) break;
  years++;
}
```

- This is really about calculating how long it takes to which your goal. It will normally exit before years reaches 100, but just looking at the while it looks like the loop should go until years is 101.

example (continued)

- The code on the previous slide could have been written:

```
while(years <= 100 && balance < goal) {
  balance += payment;
  double interest = balance * interestRate/100;
  balance += interest;
  if (balance < goal)
    years++;
}
```

- This makes is clear from reading the while what the data will look like when the loop exits. However, balance < goal is checked twice per iteration. It is debatable which code is better. I would probably write something like this, but change how years is calculated.

<u>**Block Scope**</u>:

- The *scope* of a variable is the region of code within a program where the variable can be referenced (or used).
- In Java scope is determined by the *block* of code containing the variable declaration.
- Code blocks:
  - The **`main`** method is a code block.
  - Code in the *true* clause of an **`if`** statement is a block.
  - Code in the *false* clause of an **`if`** statement is a block.
  - The statements inside a while loop is a block.
  - Code inside **`{  }`**'s is a block.

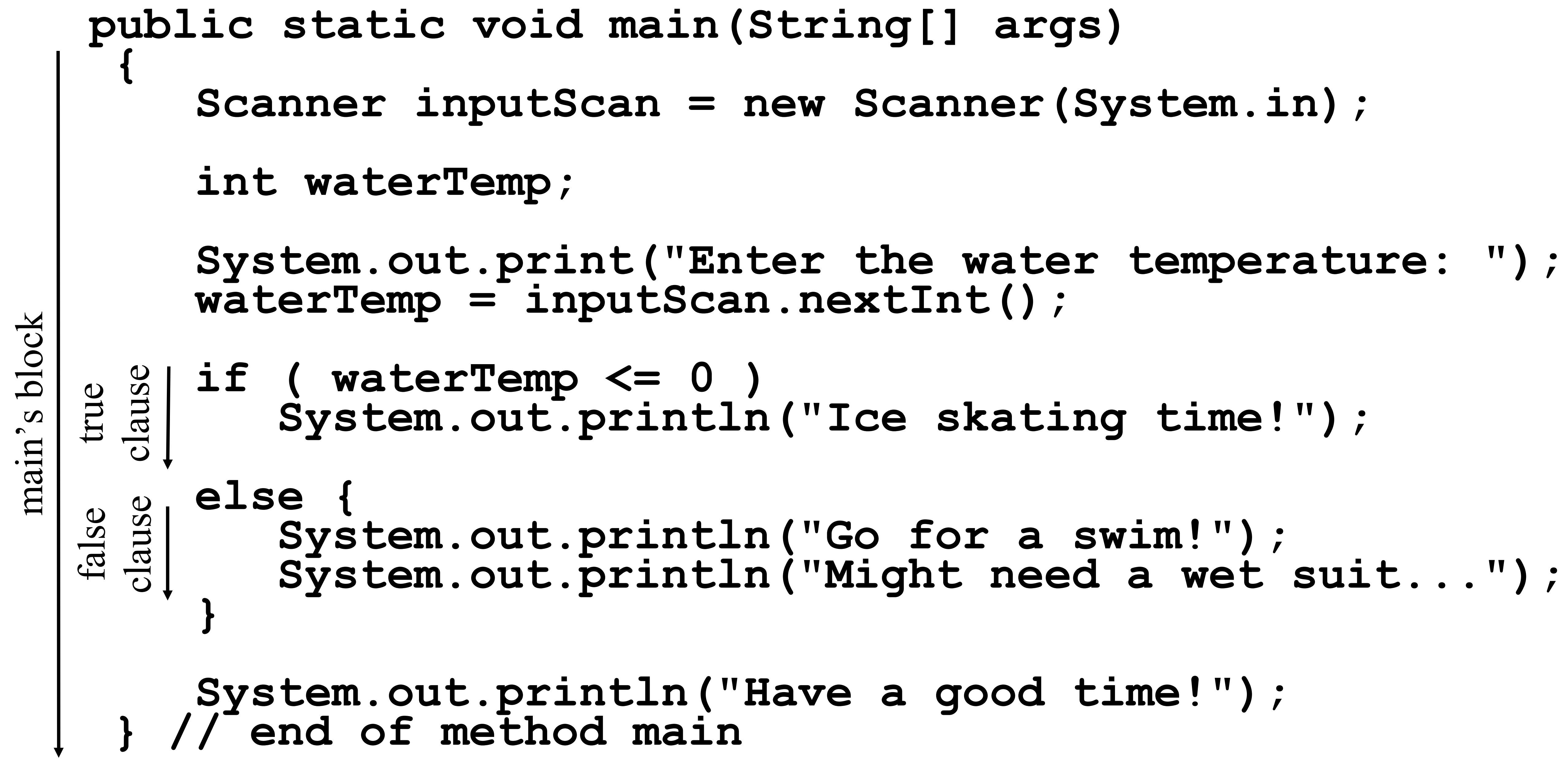
Scope:

- Example:

```
{ int i = 3;
  System.out.println(i);   //this works
}
i = 7;  //this causes a compile time error, i is not in scope
```

- The above code is strange and one doesn't usually add code blocks that are not associated with some structure, but you can.

Block Scope (continued):



```
public static void main(String[] args)
 {
    Scanner inputScan = new Scanner(System.in);

    int waterTemp;

    System.out.print("Enter the water temperature: ");
    waterTemp = inputScan.nextInt();

    if ( waterTemp <= 0 )
       System.out.println("Ice skating time!");

    else {
       System.out.println("Go for a swim!");
       System.out.println("Might need a wet suit...");
    }

    System.out.println("Have a good time!");
 } // end of method main
```

<u>Block Scope</u> (continued):

inputScan's Scope

waterTemp's Scope

```
public static void main(String[] args)
 {
    Scanner inputScan = new Scanner(System.in);

    int waterTemp;

    System.out.print("Enter the water temperature: ");
    waterTemp = inputScan.nextInt();

    if ( waterTemp <= 0 )
       System.out.println("Ice skating time!");

    else {
       System.out.println("Go for a swim!");
       System.out.println("Might need a wet suit...");
    }

    System.out.println("Have a good time!");
 } // end of method main
```

- This code works. There are two distinct variables, both named **area**. Each exists in a different scope.

```
import java.util.Scanner;
public class Area
{
   public static void main(String[] args)
   {
      Scanner inputScan = new Scanner( System.in );
      int width, height;

      System.out.print("Enter the width and height: ");
      width = inputScan.nextInt();
      height = inputScan.nextInt();

      if ( width == height ) {
  area   int area = width * width;
         System.out.println("Area of square = " + area);
      }
      else {
  area   int area = width * height;
         System.out.println("Area of rectangle = " + area);
      }

   } // end of method main
} // end of class Area
```

- This code does not work.
  - There are (still) two distinct variables, both named **`area`**. Each exists in a different scope.
  - The reference to **`area`** after the **`if`** is invalid, since it lies outside the scope of both **`area`** variables.

```
Scanner inputScan = new Scanner( System.in );
int width, height;
System.out.print("Enter the width and height: ");
width = inputScan.nextInt();
height = inputScan.nextInt();

if ( width == height ) {
  int area = width * width;
  System.out.println("Area of square = " + area);
} else {
  int area = width * height;
  System.out.println("Area of rectangle = " + area);
}
int inchesArea;
inchesArea = area * 144;  // 144 sq inches in 1 sq foot
System.out.println("Area in square inches = " + inchesArea);
} // end of method main
```

area

area

Compiler Error:
cannot find symbol

Variable Scope and for Loops:

- Many Java programmers like to declare the loop index in the **`for`** statement:

```
for (int i = 0; i < 10; ++i) {
   num = in.nextInt();
   sum += num;
}
double ave = sum / 10.0;
```

- If you do this, you can declare i again in the same method:

```
for (int i = 0; i < k; i += 2) {
```

Variable Scope and for Loops:

- However , what if you wanted to protect against input errors:

```
for (int i = 0; i < 10; ++i) {
   if (!in.hasNextInt()) break;
   num = in.nextInt();
   sum += num;
}
double ave = (double) sum / i;  // calculate average even if less
                                // than 10 numbers entered
```

- If you do this, you get a compiler error because you're trying to access **`i`** out of scope