Java Programming

Arash Habibi Lashkari

Ph.D. Candidate of UTM University

Kuala Lumpur, Malaysia

Grading:

Assignment	20%
Presentation	10%
Practical Exam	30%
Final	40%

Java Basics

- This slide teaches you the **basic language elements** and syntax for the java programming language. Once you get these basic language concepts you can continue with the other object oriented programming language concepts.

Keywords

•There are certain words with a specific meaning in java which tell (help) the compiler what the program is supposed to do. These Keywords cannot be used as variable names, class names, or method names. Keywords in java are case sensitive, all characters being lower case.

•Keywords are reserved words that are predefined in the language; see the table (Taken from Sun Java Site). All the keywords are in lowercase.

Keywords

abstract	default	if	private	this
boolean	do	implements	protected	throw
break	double	import	public	throws
byte	else	instanceof	return	transient
case	extends	int	short	try
catch	final	interface	static	void
char	finally	long	strictfp	volatile
class	float	native	super	while
const	for	new	switch	
continue	goto	package	synchronized	

Comments

- Comments are descriptions that are added to a program to make code easier to understand. The compiler ignores comments and hence its only for documentation of the program.
- Java supports three comment styles.
- *Block style* comments begin with /* and terminate with */ that spans multiple lines.
- *Line style* comments begin with // and terminate at the end of the line. (Shown in the above program)
- *Documentation style* comments begin with /** and terminate with */ that spans multiple lines. They are generally created using the automatic documentation generation tool, such as javadoc.

Variable & **data type**

- **Variables** are used for data that change during program execution. All variables have a name, a type, and a scope.
- The programmer assigns the names to variables. Name must be unique within a scope of the Java program.
- Variables have a **data type**, that indicates the kind of value they can store.
- The **data type** indicates the attributes of the variable, such as the range of values that can be stored and the operators that can be used to manipulate the variable. Java has four main primitive data types built into the language. You can also create your own composite data types.

Variable & **data type**

Java has four main primitive data types built into the language. We can also create our own data types.

- ***Integer:*** byte, short, int, and long.
- ***Floating Point:*** float and double
- ***Character:*** char
- ***Boolean:*** variable with a value of true or false.

Variable & **data type**

- When we declare a variable we assign it a name and a data type.
- For Example
- String message = “hello world”
- In the above statement, String is the **data type** for the **name** message. If you don’t specify a value when the variable is declared, it will be assigned the default value for its data type.

Variable & **data type**

Naming Rules

- Can consist of upper and lower case letters, digits, dollar sign ($) and the underscore (_) character.
- Must begin with a letter, dollar sign, or an underscore
- Are case sensitive
- Keywords cannot be used as variable name
- Within a given section of your program or scope, each user defined item must have a unique name
- Can be of any length.

Variable & **data type**

Example of Variable Initialization

```
public class MainClass {
  public static void main(String args[]) {
    double a = 3.0, b = 4.0;

    // c is dynamically initialized
    double c = Math.sqrt(a * a + b * b);

    System.out.println("Hypotenuse is " + c);
  }
}
```

Variable & **data type**

- Demonstrate lifetime of a variable
-
```
public class MainClass {
  public static void main(String args[]) {
    int x;

    for (x = 0; x < 3; x++) {
      int y = -1; // y is initialized each time block is entered
      System.out.println("y is: " + y); // this always prints -1
      y = 100;
      System.out.println("y is now: " + y);
    }
  }
}
```

Variable & **data type**

- y is: -1
- y is now: 100
- y is: -1
- y is now: 100
- y is: -1
- y is now: 100

Local variable

A local variable is a variable which is either a variable declared within the function or is an argument passed to a function. As you may have encountered in your programming, if we declare variables in a function then we can only use them within that function. This is a direct result of placing our declaration statements inside functions.

Local variable

helloMessage variable is declared as a local variable:

```
public class MainClass
    {
        public static void main(String[] args)
        {
            String helloMessage;
            helloMessage = "Hello, World!";
            System.out.println(helloMessage);
        }
    }
```

Global variable

- A global variable is a variable which is accessible in multiple scopes. It is important to note that global variables are only accessible after they have been declared.

Global variable

helloMessage variable is declared as a global variable:

```
public class MainClass
    {
    String helloMessage;

        public static void main(String[] args)
        {
            helloMessage = "Hello, World!";
            System.out.println(helloMessage);
        }
    }
```

CONTROL STATEMENTS

CONTROL STATEMENTS

- Selection Statements
  - Using `if` and `if...else`
  - Nested `if` Statements
  - Using `switch` Statements
  - Conditional Operator
- Repetition Statements
  - Looping: `while`, `do-while`, and `for`
  - Nested loops
  - Using `break` and `continue`

Selection Statements

- `if` Statements
- `switch` Statements
- Conditional Operators

if Statement

```
if (condition) {
  statement(s);
}
```

Example:

```
if ( (i > 0) && (i < 10) ) {
  System.out.println("i is an integer between 0 and 10");
}
```

if Statement

Adding a semicolon at the end of an if clause is a common mistake.

```
if (radius >= 0);
{
  area = radius*radius*3.14;
  System.out.println("The area for the circle of radius " + radius + " is " +
area);
}
```

This mistake is hard to find, because it is not a compilation error or a runtime error, it is a logic error.

This error often occurs when you use the next-line block style.

if...else Statement

```
if (condition) {
  statement(s)-for-the-true-case;
}
else {
  statement(s)-for-the-false-case;
}
```

if...else Statement

```
if (radius >= 0) {
  area = radius*radius*3.14;

    System.out.println("The area for the “ + “circle of radius " + radius +
    " is " + area);
}
else {
  System.out.println("Negative input");
}
```

Multiple Alternative if Statements

```
if (score >= 90)
  grade = ‘A’;
else if (score >= 80)
  grade = ‘B’;
else if (score >= 70)
  grade = ‘C’;
else if (score >= 60)
  grade = ‘D’;
else
  grade = ‘F’;
```

switch Statement

- Switch statements are shorthands for a certain kind of if statement. It is not uncommon to see a stack of if statements all relate to the same quantity like this:

```
if (x == 0) doSomething0();
 else if (x == 1) doSomething1();
 else if (x == 2) doSomething2();
 else if (x == 3) doSomething3();
 else if (x == 4) doSomething4();
 else doSomethingElse();
```

switch Statement

- Java has a shorthand for these types of multiple if statements, the switch-case statement. Here's how you'd write the above using a switch-case:

```
switch (x) {
 case 0: doSomething0(); break;
 case 1: doSomething1(); break;
 case 2: doSomething2(); break;
 case 3: doSomething3(); break;
 case 4: doSomething4(); break;
 default: doSomethingElse();
 }
```

switch Statement

The switch-expression must yield a value of char, byte, short, or int type and must always be enclosed in parentheses.

The value1, ..., and valueN must have the same data type as the value of the switch-expression. The resulting statements in the case statement are executed when the value in the case statement matches the value of the switch-expression. (The case statements are executed in sequential order.)

The keyword break is optional, but it should be used at the end of each case in order to terminate the remainder of the switch statement. If the break statement is not present, the next case statement will be executed.

Conditional Operator

```
if (x > 0)
    y = 1;
else
    y = -1;
```

is equivalent to

```
y = (x > 0) ? 1 : -1;
```

Conditional Operator

```
if (num % 2 == 0)
  System.out.println(num + “is even”);
else
  System.out.println(num + “is odd”);
```

is equivalent to

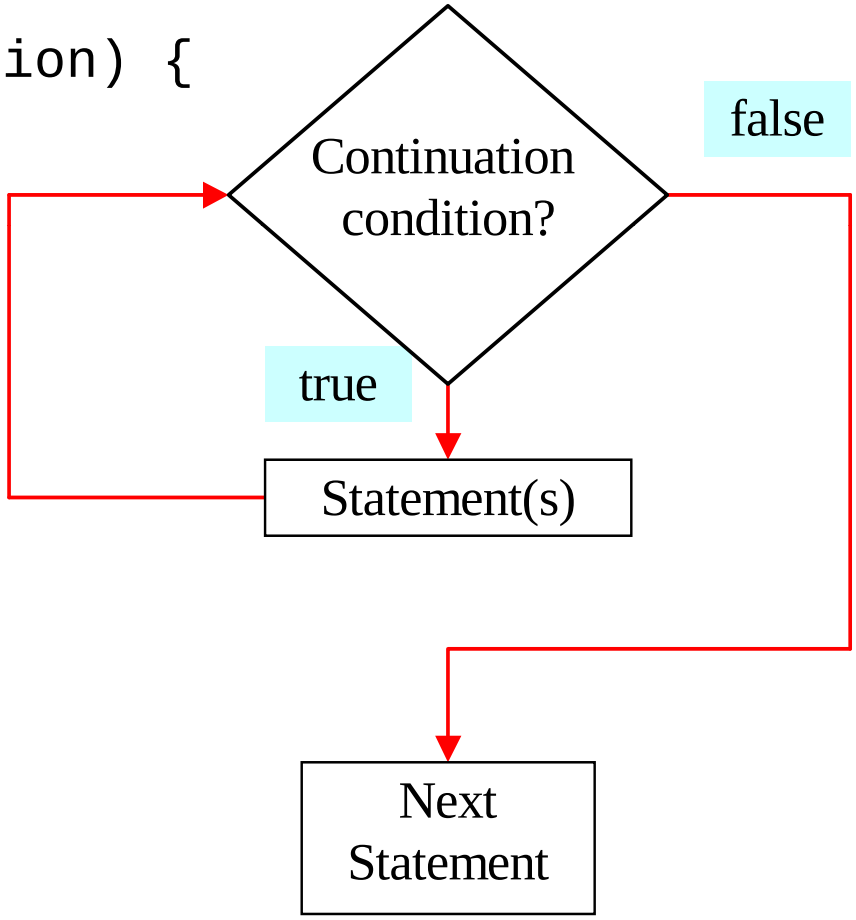
```
System.out.println( (num % 2 == 0)? num + “is even” :  num + “is odd”);
```

Repetitions

- `while` Loops
- `do-while` Loops
- `for` Loops
- `break` and `continue`

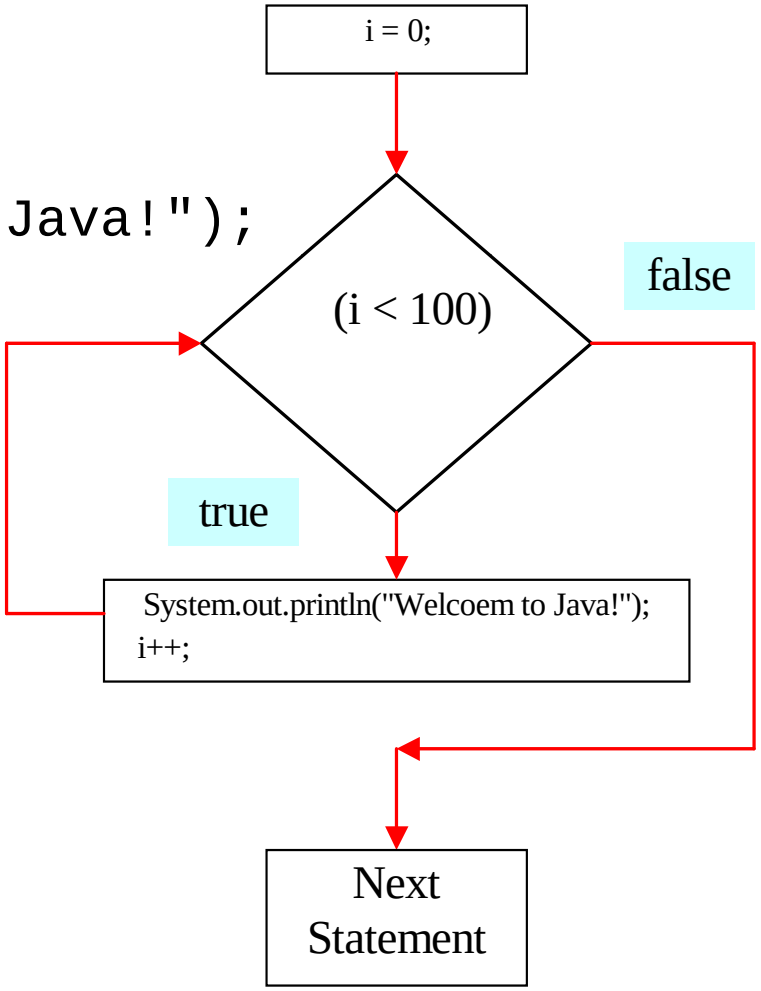
while Loops

```
while (continuation-condition) {
 // loop-body;
}
```



while Loops

```
int i = 0;
while (i < 100) {
  System.out.println("Welcome to Java!");
  i++;
}
```



while Loops

```
Try this :
public static void main() {
    int counter = 1;                    // 1.  declare and initialise variables
    while(counter < 5 ) {               // 2.  now do the while loop
          System.out.println("counter  is equal to " + counter);
          counter = counter + 1 ;       // 3.  add 1 to the value of counter
    }                                   // 4.  the while loop has finished,
//so program continues on to here
    System.out.println("Goodbye");
}
```

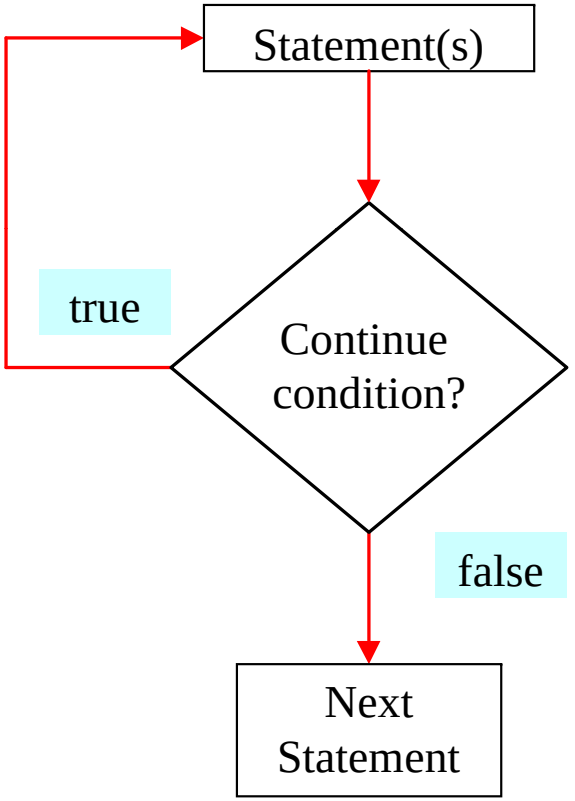
while Loops

Notes on Above code

- Initialise the counter to 1
- the while loop(the statements inside the curly brackets after while) will repeat as long as the statement after it is true. If the statement is false(counter = 5 or greater) then the loop stops
- the counter adds one to itself . Each time the loop goes through, one is added to the counter until the condition (x<5) is false. I.e. x = 6 . If the condition is false the loop is bypassed.
- Goes on to the next line and prints "goodbye"., and ends the program.

do-while Loop

```
do {
  // Loop body;
} while (continue-condition);
```



do-while Loop

- do while loops are similar to while loops except that the condition is tested after the code to be repeated or looped. The general form of a do .. while loop is as follows:

```
do{
    // execute code between these brackets
}while(true or false(boolean) statement);
```

For Loop

Java for loops are a short hand way of:

- initializing the counter ,
- setting the Boolean condition, and
- incrementing the counter

All in one statement.

Before we carry on we have to learn about the:

Java INCREMENT operator (++)

For Loop

Java INCREMENT operator (++)

x++; is the same as x = x + 1;

X++ increments (or increases) the value of X by 1

Here is an example that we could use in a while loop:

```
int x = 0 ;          // declare and initialize counter x
while( x < 5) {
        System.out.println(" x = " + x );
        x++;      // add 1 to the value of x, same as (x = x + 1)
        }
```

For Loop

Similarly , the:

DECREMENT operator is (--)

y-- decrements (or decreases) y by 1

y-- ; is the same as y = y - 1 ;

Try this :

```
int y = 10 ;
while( y > 0){
        System.out.println(" and " + y);
        y-- ;
}
```

For Loop

- So, back to the for Loop.
  A for Loop sets all the conditions of a loop inside the parantheses after the for statement.
  Example A.

```
for(int x = 0 ; x < 10 ; x++){
        System.out.println("x = " + x);
} The form of the for loop is:
```

```
for(  INITIALIZE counter  ;  boolean CONDITION ; ITERATION){
    // repeated code to be executed.
}
```

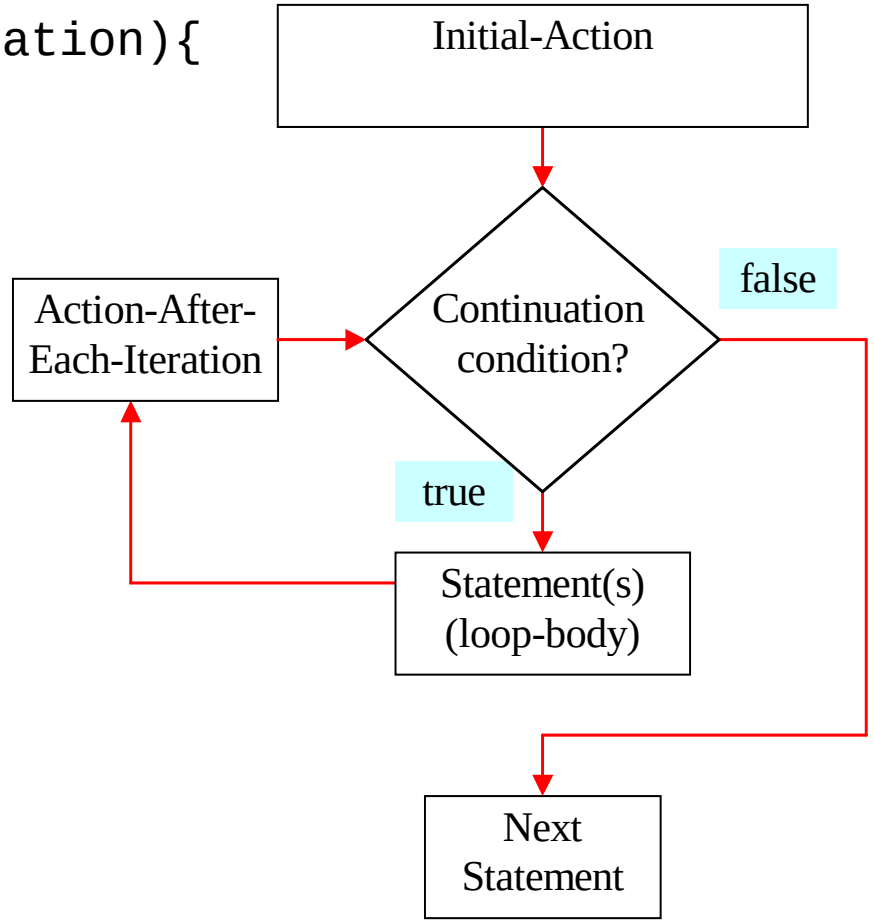
For Loop

- iteration means repetition, or repeating. In actual fact it is either usually increased by 1 (x++) or decreased by 1 (y--).
  In example A above , the Initialisation statement is int x = 0;
  this statement declares x as an int and initializes x to be equal to a value of 0 in one go.
  The boolean condition is x < 10 ; This statement tests x and if it is true , then the loop is allowed to execute, If it is false (x=10 or x > 10), then the loop is bypassed and the program execution continues after the { curly brackets } following the for statement.
- Example B. Count down from 10 to 1

```
for(int x = 10 ;  x>0 ; x--){
        System.out.println(x);
}
```

For Loop

```
for (initial; loop-condition; iteration){
   //loop body;
}
```



For Loop

Which Loop to Use?

- The three forms of loop statements, while, do, and for, are expressively equivalent; that is, you can write a loop in any of these three forms.
- I recommend that you use the one that is most intuitive and comfortable for you. In general, a for loop may be used if the number of repetitions is known, as, for example, when you need to print a message 100 times. A while loop may be used if the number of repetitions is not known, as in the case of reading the numbers until the input is 0. A do-while loop can be used to replace a while loop if the loop body has to be executed before testing the continuation condition.

For Loop

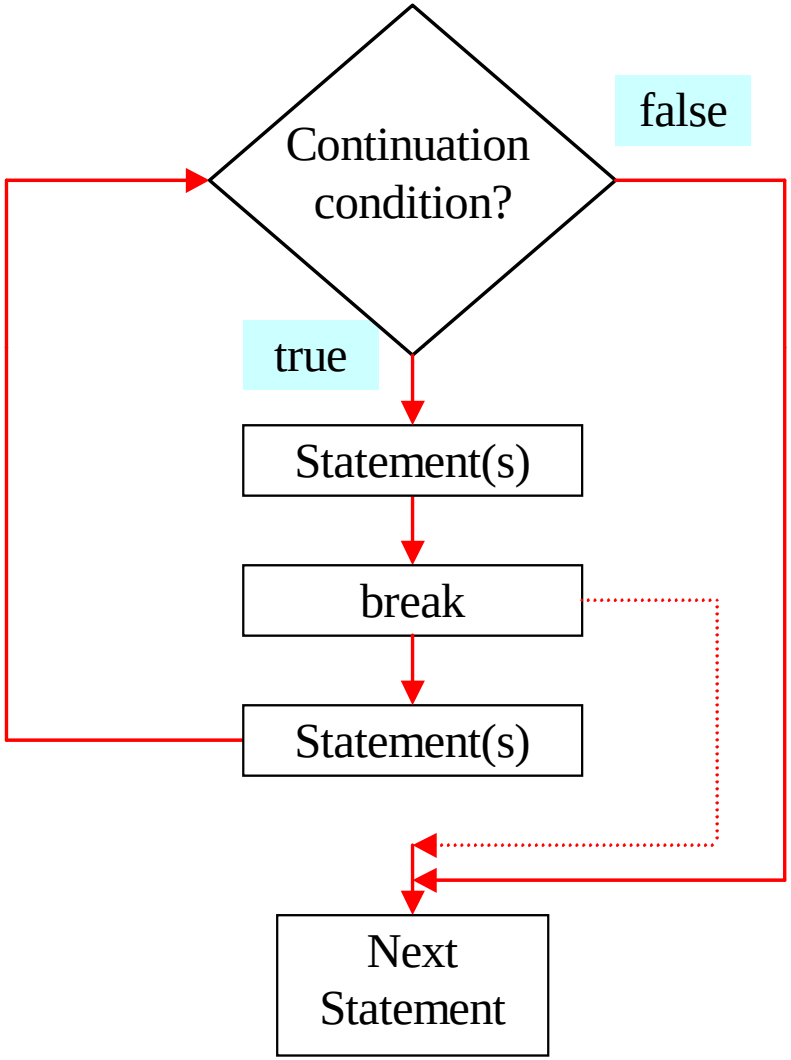
Adding a semicolon at the end of the for clause before the loop body is a common mistake, as shown below:

```
for (int i=0; i<10; i++);
{
  System.out.println("i is " + i);
}
```

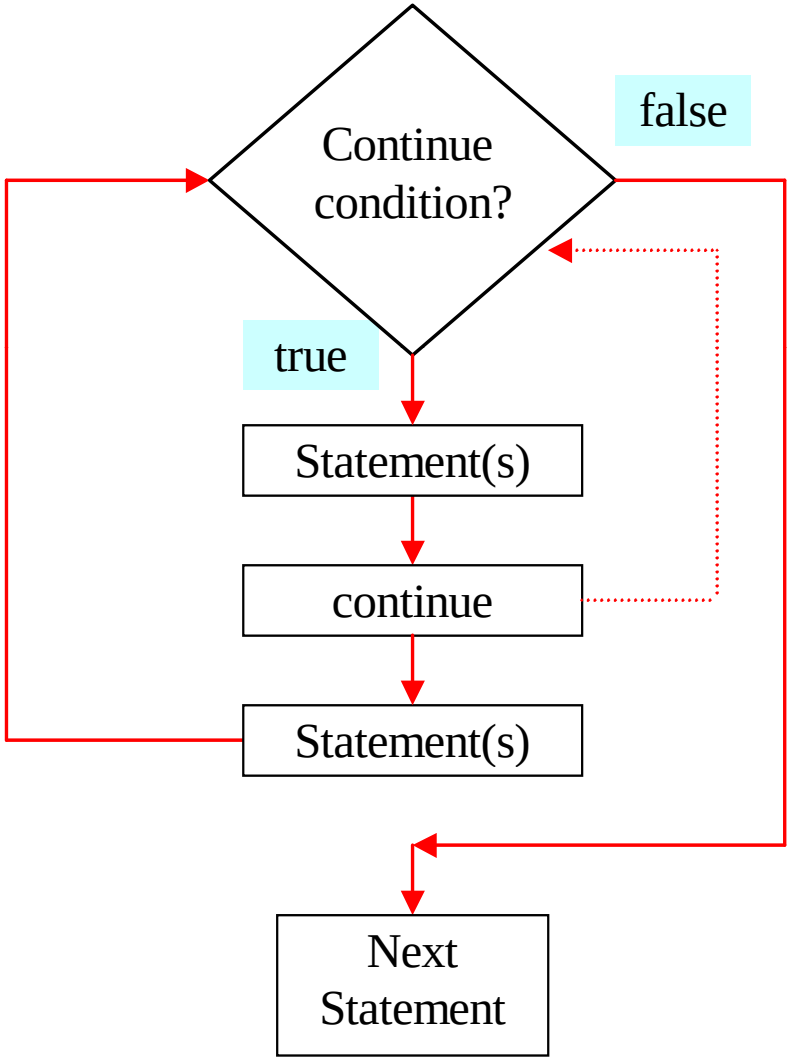
Break and Continue

- You can also control the flow of the loop inside the body of any of the iteration statements by using **break** and **continue**. **break** quits the loop without executing the rest of the statements in the loop. **continue** stops the execution of the current iteration and goes back to the beginning of the loop to begin the next iteration.

The break Keyword



The continue Keyword



Questions

THANK YOU

Arash Habibi Lashkari

PHD. Candidate of UTM

Kuala Lumpur, Malaysia

Feb, 2010

THE END