JAVA PROGRAMMING BASICS

Module 1: Java Overview

Training program

1. Java Fundamentals
2. Start programming with Java, create simple console application
3. Classification of Data Types
4. Primitive types in java
5. Control Flow Statements
6. Arrays

Module contents

- Control Flow Statements
  - Identifiers and Literals
  - Local variables: initialization and lifetime
  - Declaring a Variable as a Constant
  - The if-then and if-then-else statements
  - The switch statement
  - Loops: the while, do-while and for statements
  - The break and continue statements
  - The goto keyword
  - Program exit

Identifiers and literals

- Identifiers must start with a letter, a currency character ($), or a connecting character
- After the first character, identifiers can contain any combination of letters, currency characters, connecting characters, or numbers
- Identifier can be any length
- Identifier cannot be a Java keyword
- Identifiers in Java are case-sensitive
- Identifier cannot be true, false or null.

Keywords in the Java programing language

abstract	double	int	provides…with♦♦	throws
assert***	else	interface	public	transitive♦♦
boolean	enum♦	long	record■	transient
break	extends	module♦♦	requires♦♦	true
byte	false	native	return	try
case	final	non-sealed■■	sealed■■	uses♦♦
catch	finally	null	short	var♦♦
char	float	new	static	void
class	for	open♦♦	strictfp**+*	volatile
const*	goto*	opens…to♦♦	super	while
continue	if	package	switch	yield■
exports♦♦	implements	permits■■	synchronized	
default	import	private	this	
do	instanceof	protected	throw	

* not used, ** 1.2 added, *** 1.4 added, ♦ 5 added, ♦ ♦ 9 added, ■14 added, ■ ■15 added

Java Code Conventions: Variable

- A variable name should be short and meaningful
- The first letter should be in lowercase and follow the “CamelCase” format for words that are linked together
- Examples of Java variables:

```
nama
buttonWidth
accountBalance
myString
```

Module contents

- Control Flow Statements
  - Identifiers and Literals
  - Local variables: initialization and lifetime
  - Declaring a Variable as a Constant
  - The if-then and if-then-else statements
  - The switch statement
  - Loops: the while, do-while and for statements
  - The break and continue statements
  - The goto keyword
  - Program exit

Local variables: initialization and lifetime

1. Local variables are declared in methods, constructors, or blocks.
2. Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor or block.
3. Access modifiers cannot be used for local variables.
4. Local variables are visible only within the declared method, constructor or block.
5. Local variables are implemented at stack level internally.
6. There is no default value for local variables so local variables should be declared and an initial value should be assigned before the first use.
7. The compiler forces local variable to be initialized when it is used in other statements.

Local variables: initialization and lifetime

-
  ```
  public static void main(String[] arg){

      int x = 10;

      {
        int y =20;
      }

      System.out.print(x);
      System.out.print(y);
  }
  ```

Variable not visible outsite block

Local variables: initialization and lifetime

```
public static void main(String[] args) {
   /*The local variable must be explicitly initialized.*/
   int localVar = 0;
   /*The compiler forces it to be initialized when it is used
      in other statements.*/
   System.out.println("Local int localVar= " + localVar);
}

void anotherMethod() {
   /*The local variable must be explicitly initialized.
   int a = 0;
   /*The compiler forces it to be initialized when it is used
      in other statements.*/
   System.out.println("a=" + a);
}
```

Module contents

- Control Flow Statements
  - Identifiers and Literals
  - Local variables: initialization and lifetime
  - Declaring a Variable as a Constant
  - The if-then and if-then-else statements
  - The switch statement
  - Loops: the while, do-while and for statements
  - The break and continue statements
  - The goto keyword
  - Program exit

Declaring a Variable as a Constant

- Constant represent permanent data that will never change
- To declare a constant need to use the **final** keyword
- Java constants should be named using uppercase letters with underscore characters as separators

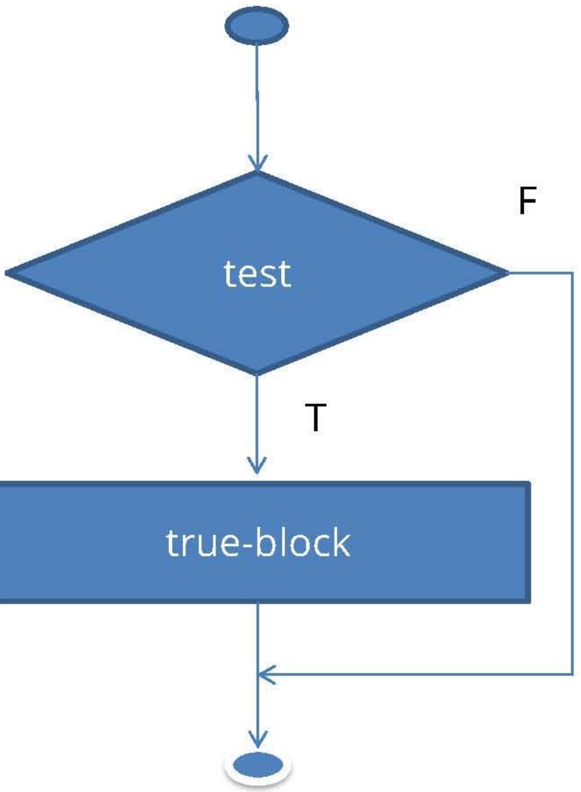
For example:

```
final double PI = 3.14159;
final int NUMBER_OF_HOURS_IN_A_DAY = 24;
```

Module contents

- Control Flow Statements
  - Identifiers and Literals
  - Local variables: initialization and lifetime
  - Declaring a Variable as a Constant
  - The if-then and if-then-else statements
  - The switch statement
  - Loops: the while, do-while and for statements
  - The break and continue statements
  - The goto keyword
  - Program exit

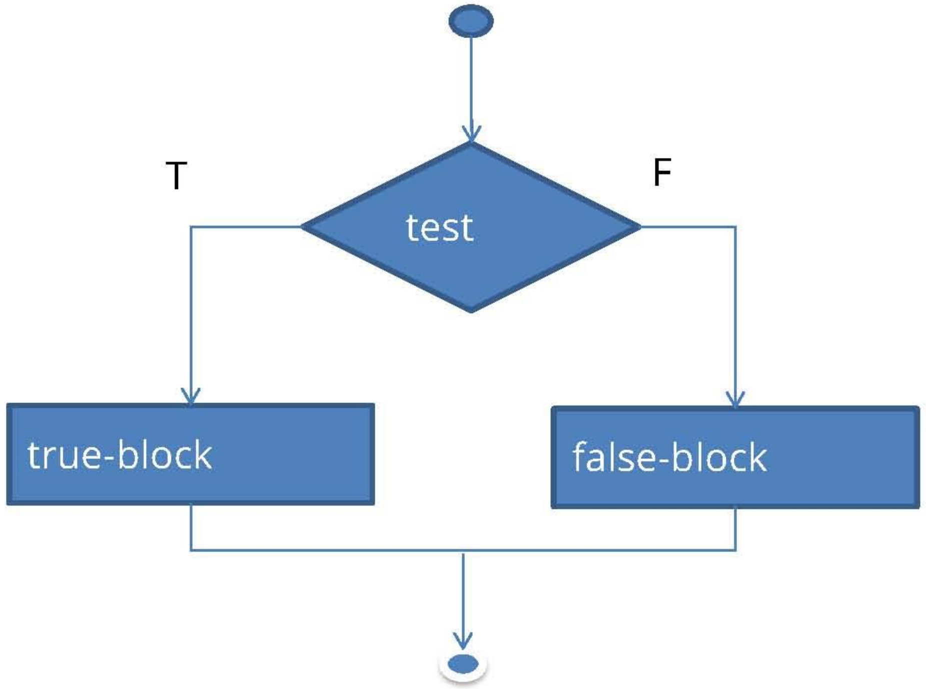
The if Statement 1/2



The if Statement 2/2

```java
package com.mycompany.conditionals;
public class TestIfElse {
   public static void main(String[] args) {
      int x = 10;
      int y = 20;
      //Single option
      if (x<y) {
         //...
         System.out.println("x<y -> true");
      }
   }
}
```

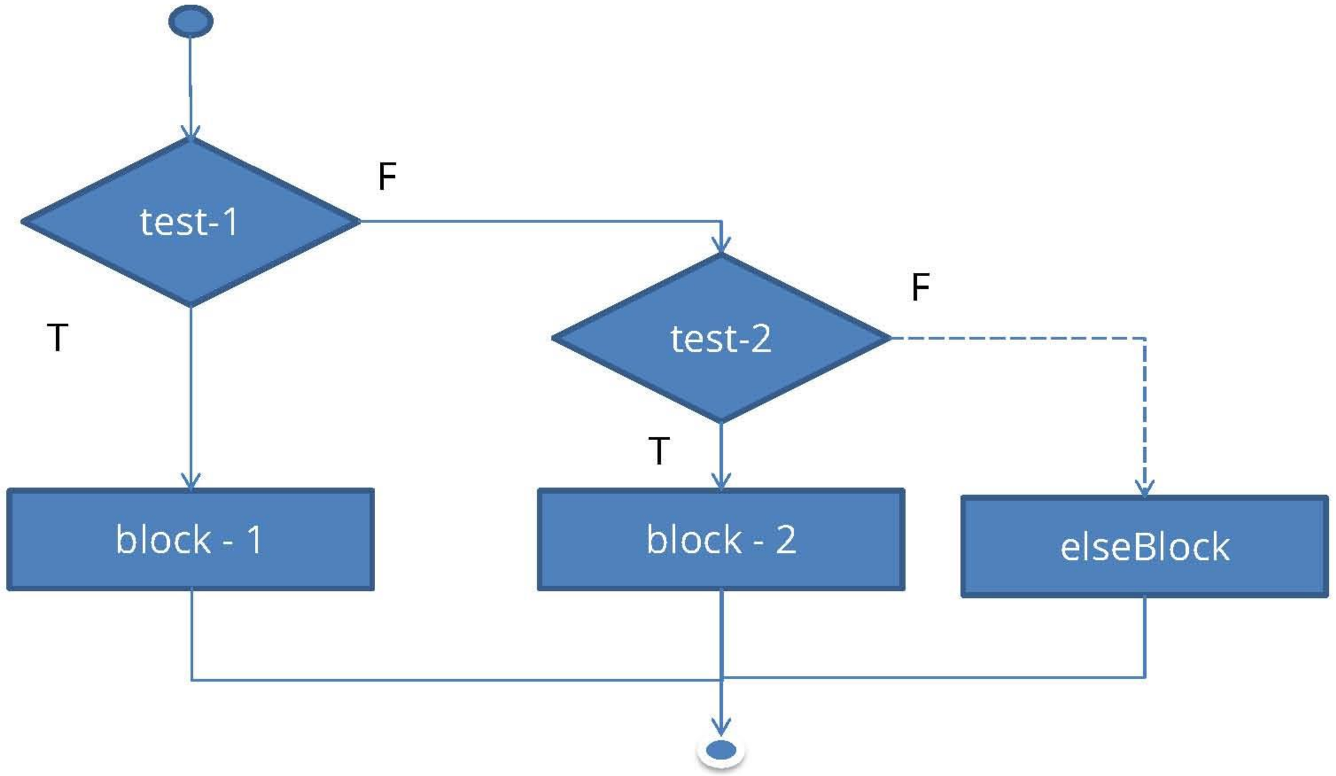
The if-else Statements 1/3



The if-else Statement 2/3

```java
int x = 10;
int y = 20;
boolean boolExpr = x < y;
// Two options
    if (boolExpr) {
//...
      System.out.println("x<y -> true");
    } else {
//...
      System.out.println("x<y -> false");
    }
```

The if-else Statements 3/3



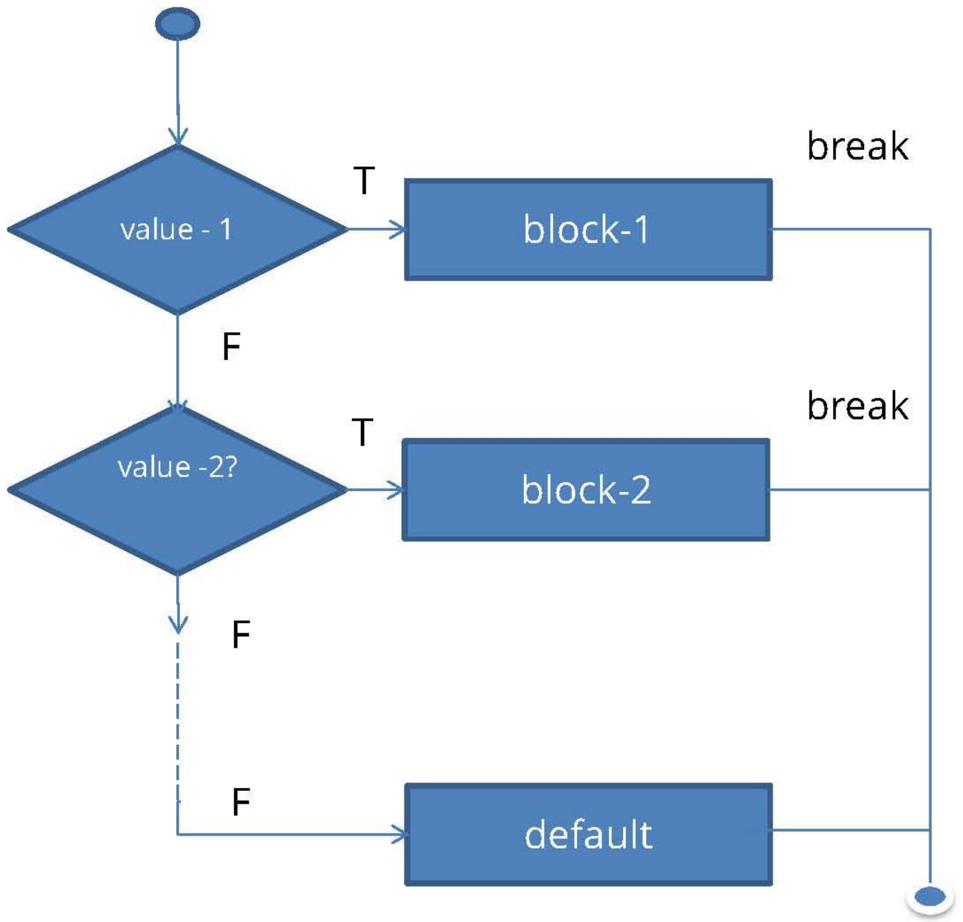
The if-then and if-then-else Statements

```
int x = 10;
int y = 20;
// Multiple options
        if (x < y) {
//...
            System.out.println("x<y -> true");
        } else if (x == 0) {
//...
            System.out.println("x<y -> false, and x=0");
        } else if (y == 1) {
//...
            System.out.println("x<y -> false, and x!=0, y=1");
        } else {
//...
        }
```

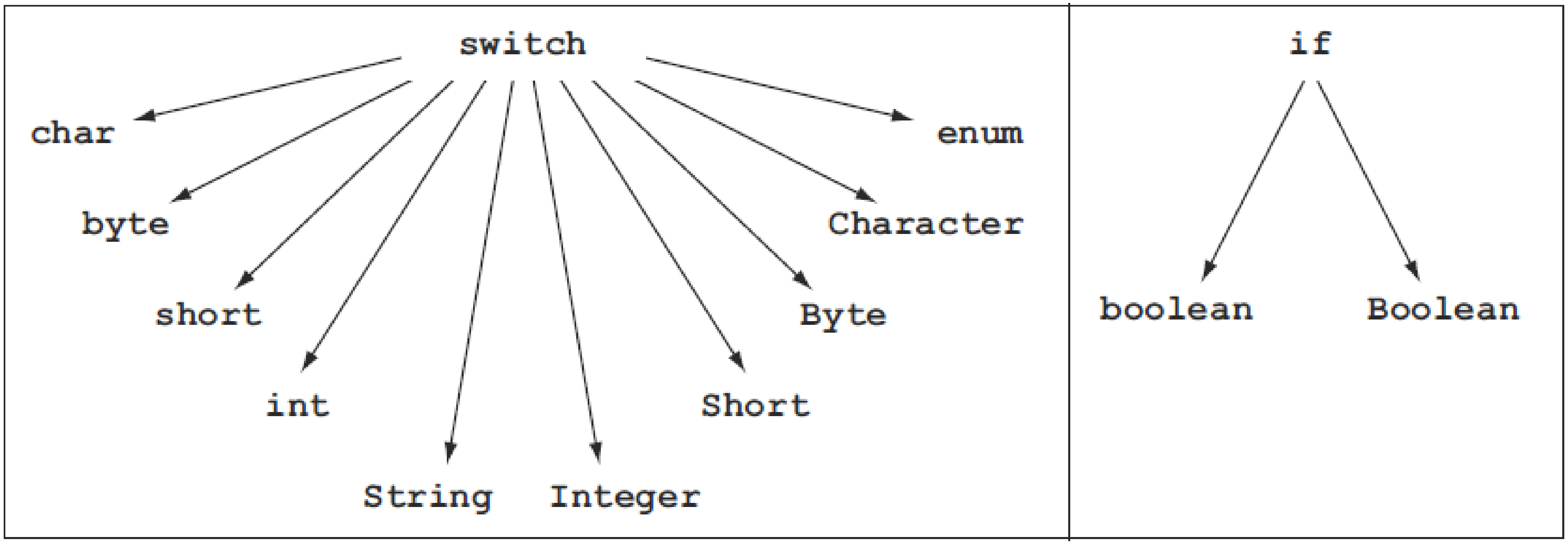
Module contents

- Control Flow Statements
  - Identifiers and Literals
  - Local variables: initialization and lifetime
  - Declaring a Variable as a Constant
  - The if-then and if-then-else statements
  - The switch statement
  - Loops: the while, do-while and for statements
  - The break and continue statements
  - The goto keyword
  - Program exit

The switch Statement 1/2



Argument types passed to a switch and if statements



```
int a=10, b=20, c=30;    //if variable has final modifier then it is constant
switch (a) {
   /*The value of a case label must be a compile-time constant value*/
   case b+c: System.out.println(b+c); break;
   case 10*7: System.out.println(10*7512+10); break;
}
```

The switch Statement 2/2

```java
int month = 5;
String monthStr;
switch(month) {
   case 1: monthStr = "January";
   break;
   case 2: monthStr = "February";
   break;
   case 3: monthStr = "March";
   break;
  //...
   default: monthStr = "Invalid month";
   break;
}
System.out.println(monthStr);
```

The switch Statement 2/3

```
int month = 2, year = 2000, numDays = 0;
boolean hYear = true;
switch (month) {
   case 1: case 3: case 5:
   case 7: case 8: case 10:
   case 12:
      numDays = 31; break;
   case 4: case 6:
   case 9: case 11:
      numDays = 30; break;
   case 2:
      if (hYear) numDays = 29;
      else numDays = 28;
      break;
   default:
      System.out.println("Invalid month.");
      break;
}
System.out.println("Number of Days = “ + numDays);
```

The enhanced switch

```
public static void main(String[] args) {
    int month = 5;
    String monthStr;
    monthStr = switch (month) {
        case 1 -> "January";
        case 2 -> "February";
        case 3, 4, 5 ->{
            System.out.println("The goup of months");
            yield "Spring";
        }
        default -> {
            System.out.println("Invalid month");
            yield "";
        }
    };
    System.out.println(monthStr);
}
```

enhanced switch is a statement

terminates the switch and returns result

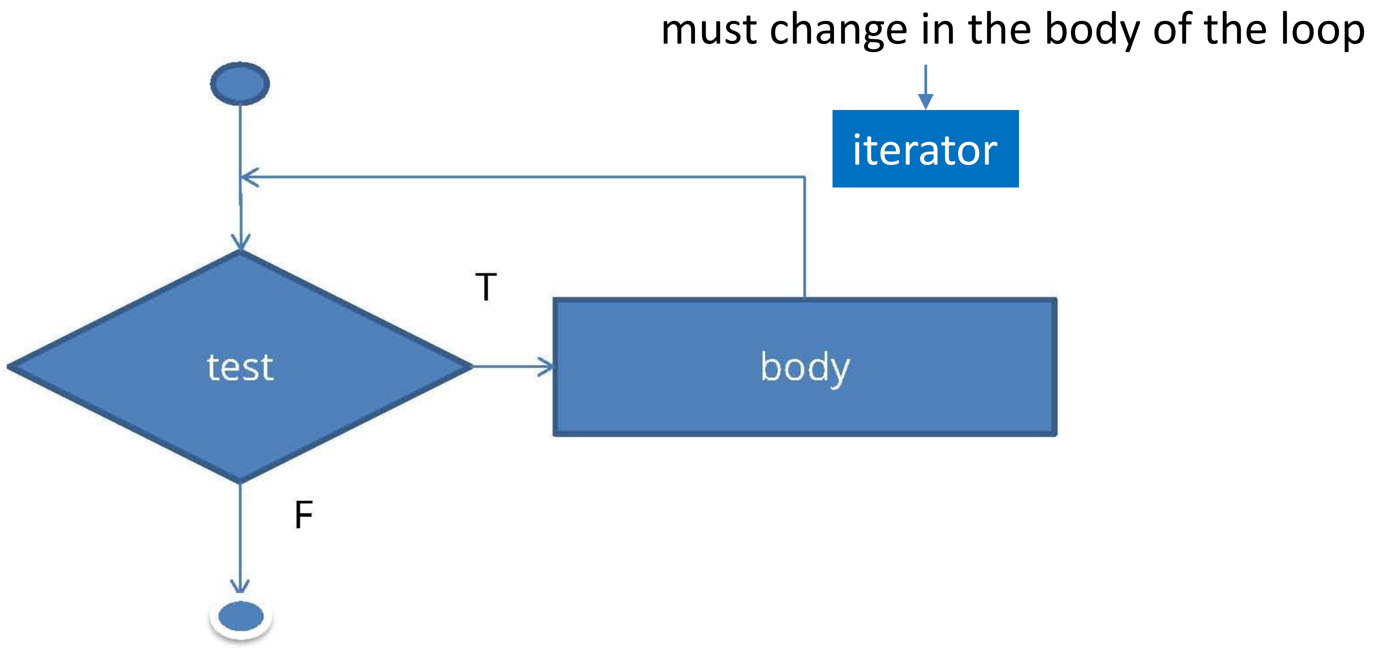
a statement have to end with ;

since JDK 14

Module contents

- Control Flow Statements
  - Identifiers and Literals
  - Local variables: initialization and lifetime
  - Declaring a Variable as a Constant
  - The if-then and if-then-else statements
  - The switch statement
  - Loops: the while, do-while and for statements
  - The break and continue statements
  - The goto keyword
  - Program exit

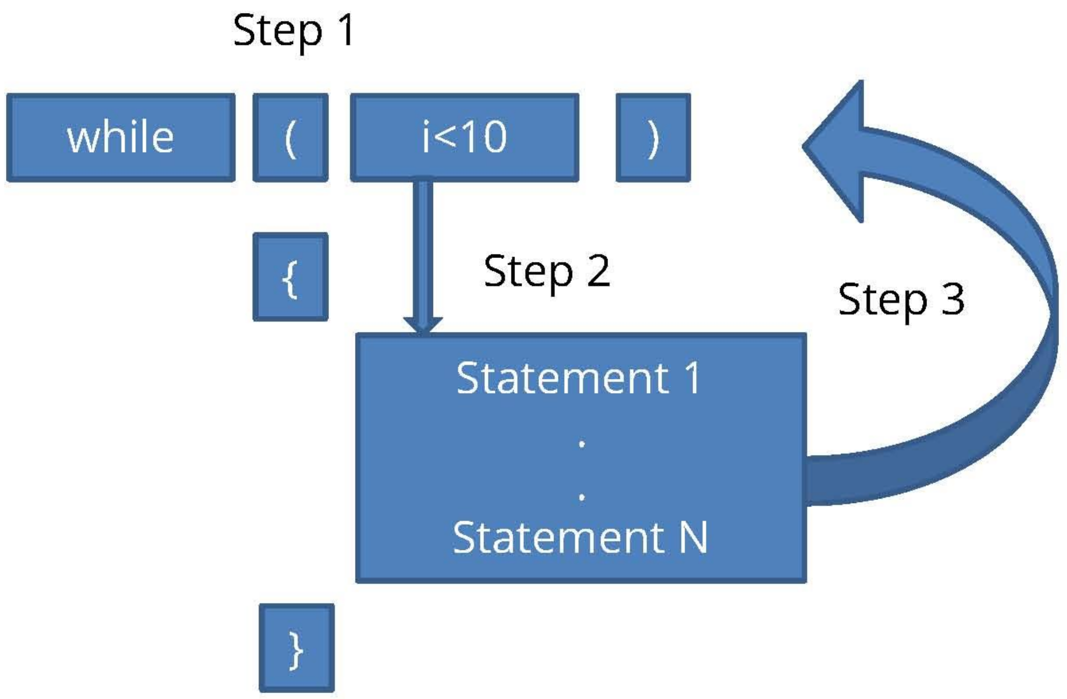
Loops: The while Statements



while 1/2

```java
int i = 0;
    while (i < 10) {
        //...
        System.out.println("Iteration: " + i);
        i++;
    }
```

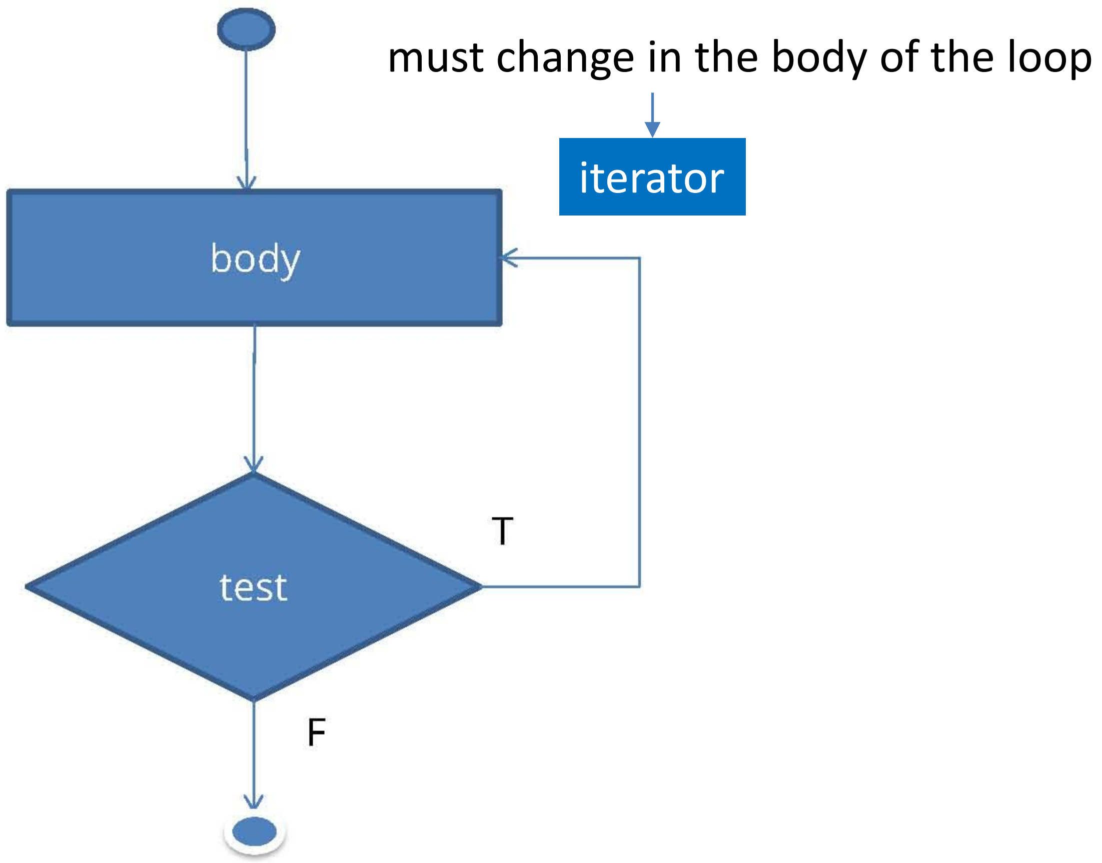
while 2/2



Output:

```
Iteration: 0
Iteration: 1
Iteration: 2
Iteration: 3
Iteration: 4
Iteration: 5
Iteration: 6
Iteration: 7
Iteration: 8
Iteration: 9
```

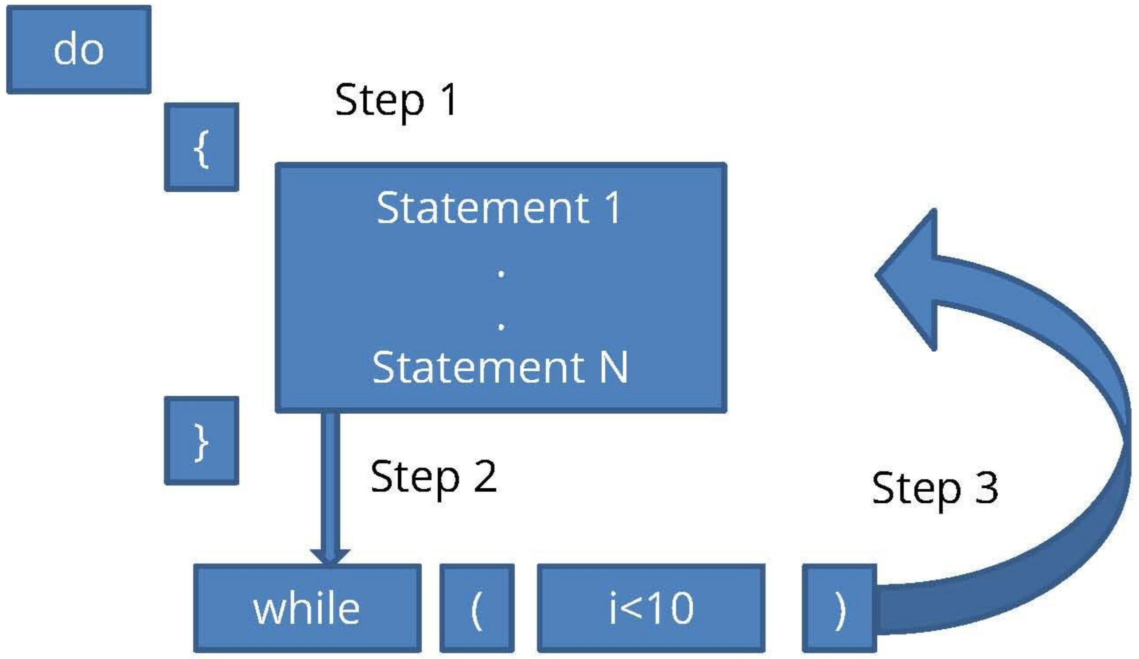
Do while 1/3



Do while 2/3

```java
int i = 0;
        do {
//...
            System.out.println("Iteration: " + i);
            i++;
        } while (i < 10);
```

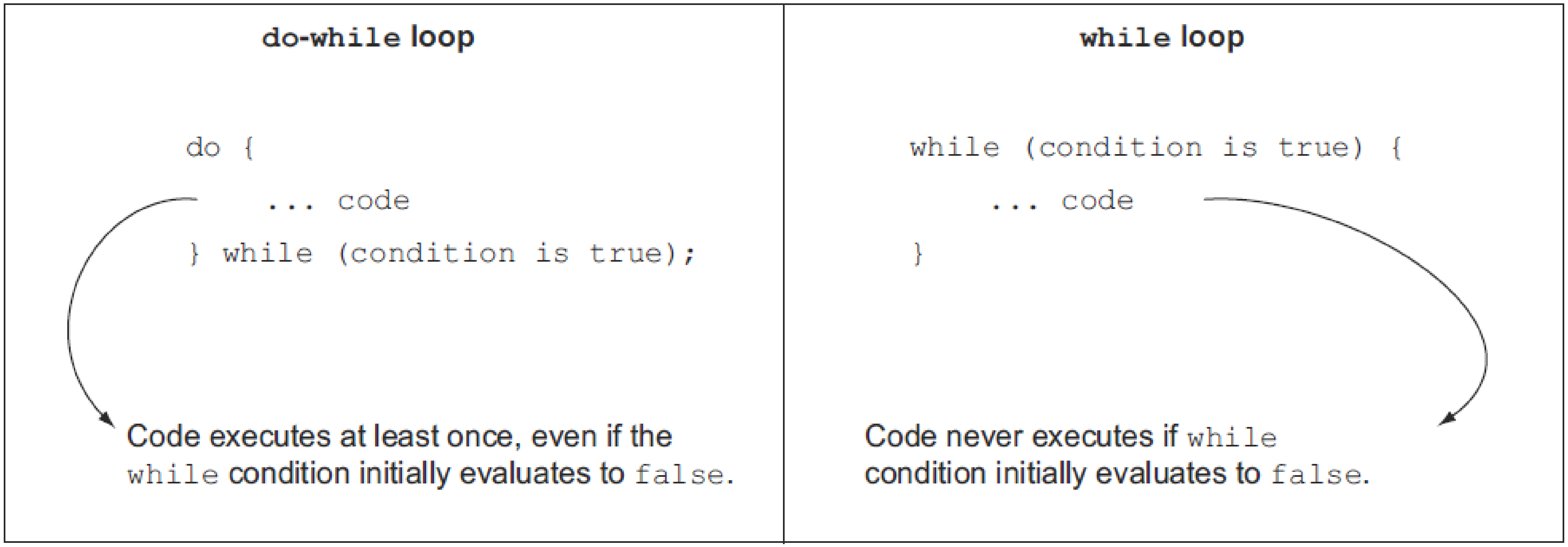
Do while 3/3



Output:

```
Iteration: 0
Iteration: 1
Iteration: 2
Iteration: 3
Iteration: 4
Iteration: 5
Iteration: 6
Iteration: 7
Iteration: 8
Iteration: 9
```

while vs do-while loop



`do-while` loop

```
do {
    ... code
} while (condition is true);
```

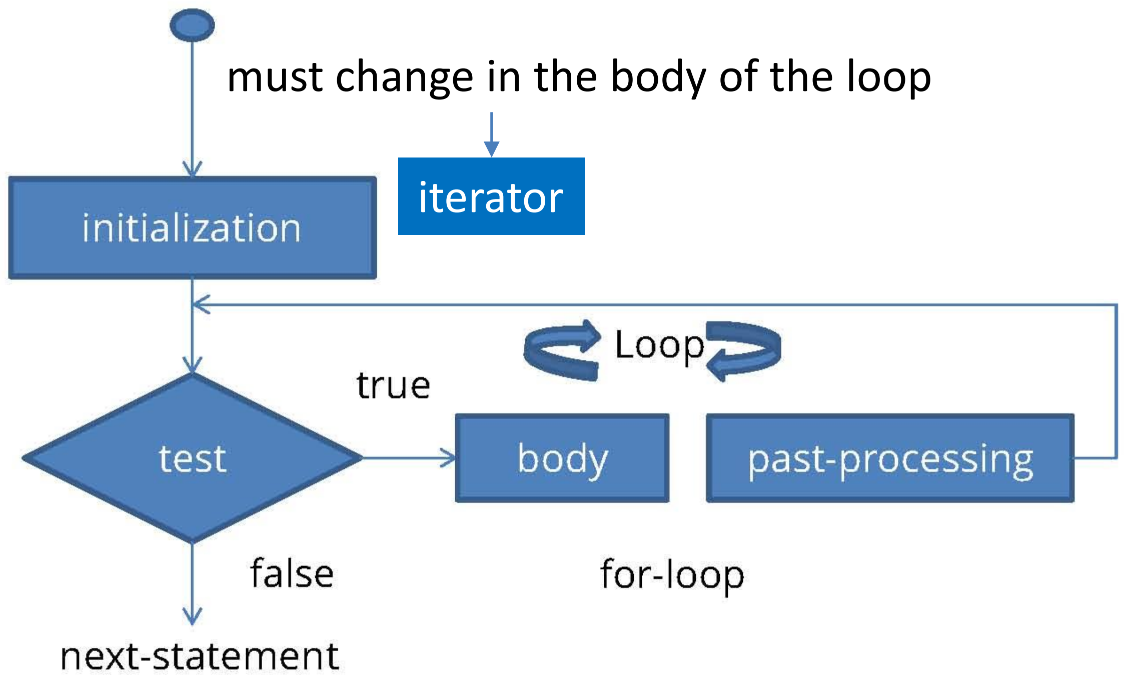
Code executes at least once, even if the `while` condition initially evaluates to `false`.

`while` loop

```
while (condition is true) {
    ... code
}
```

Code never executes if `while` condition initially evaluates to `false`.

for 1/3

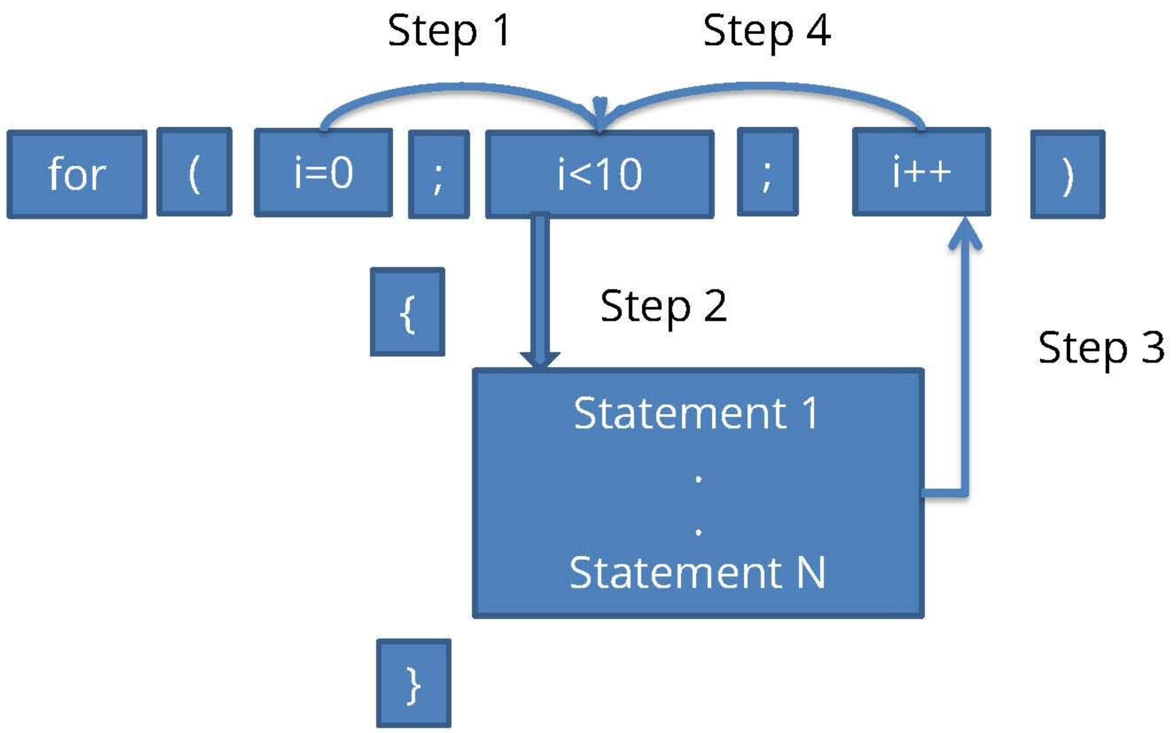

must change in the body of the loop
iterator
initialization
Loop
true
test
body
past-processing
false
for-loop
next-statement

for 2/3

```java
for (int i = 0; i < 10; i++) {
//...
            System.out.println("Iteration: " + i);
        }
```

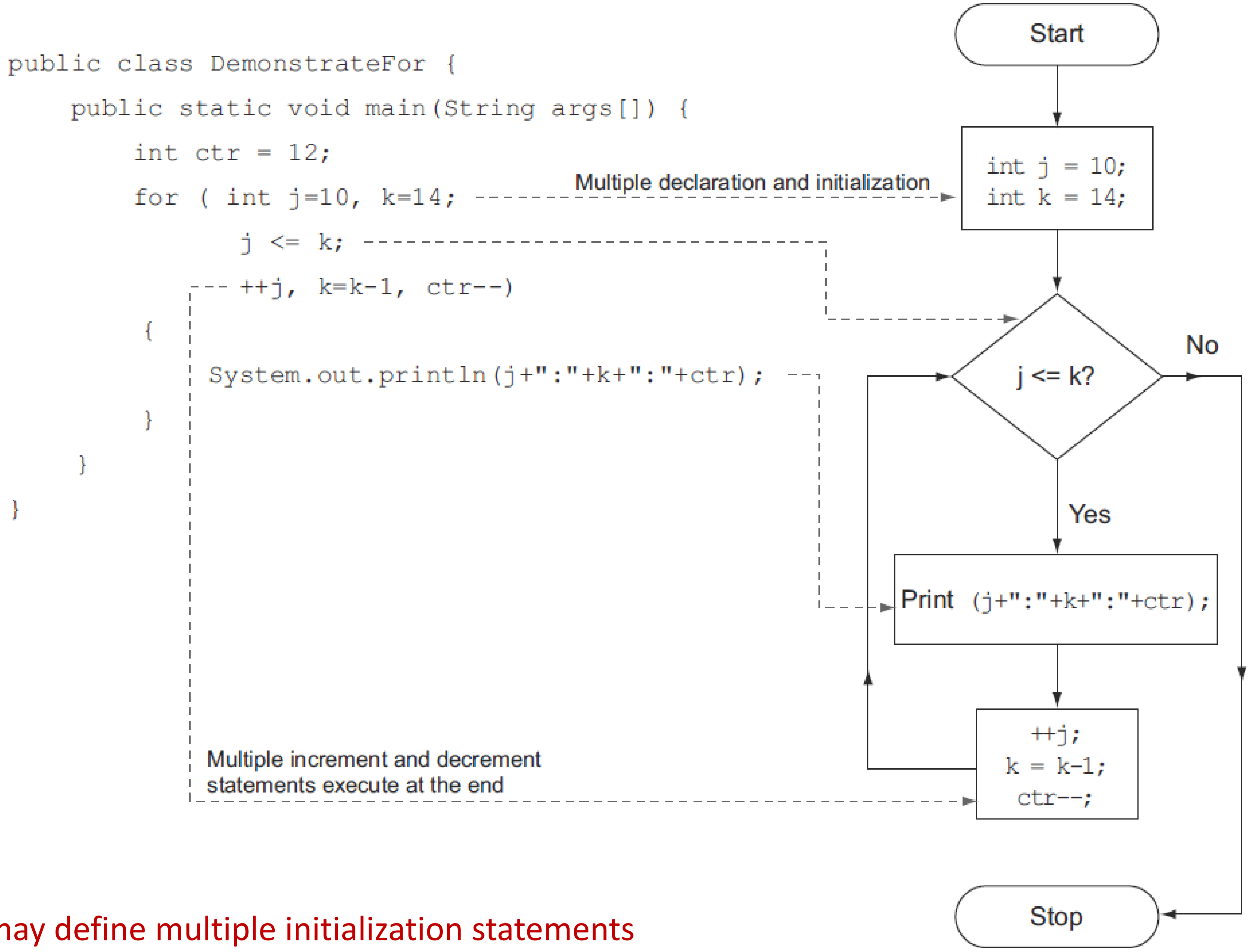
for 3/3

Control Flow of for-loop



```
Output:

Iteration: 0
Iteration: 1
Iteration: 2
Iteration: 3
Iteration: 4
Iteration: 5
Iteration: 6
Iteration: 7
Iteration: 8
Iteration: 9
```



```
public class DemonstrateFor {
    public static void main(String args[]) {
        int ctr = 12;
        for ( int j=10, k=14;
                j <= k;
              ++j, k=k-1, ctr--)
         {
          System.out.println(j+":"+k+":"+ctr);
         }
    }
}
```

Multiple declaration and initialization
Multiple increment and decrement statements execute at the end
Start
int j = 10;
int k = 14;
j <= k?
No
Yes
Print (j+":"+k+":"+ctr);
++j;
k = k-1;
ctr--;
Stop

You may define multiple initialization statements and/or multiple update clause. But there can be only one termination condition for a for loop.

Module contents

- Control Flow Statements
  - Identifiers and Literals
  - Local variables: initialization and lifetime
  - Declaring a Variable as a Constant
  - The if-then and if-then-else statements
  - The switch statement
  - Loops: the while, do-while and for statements
  - The break and continue statements
  - The goto keyword
  - Program exit

The break and continue statements 1/2

```
public static void main(String[] args){
    int i = 0;
    while(true) {
        if (i > 10) {
            break;
        }
        System.out.println("i="+i);
        i++;
    }
    System.out.println("Program exit");
}
```

Output:

```
i=0
i=1
…
i=10
Program exit
```

break terminates a for, while, or do-while loop and switch case

The break and continue statements 2/2

```java
public static void main(String[] arg){
    int i = 0;
    while(i++<10) {
        if (i==5) {
            continue;
        }
        System.out.println(i);
    }
}
```

continue terminates a current iteration for, while, or do-while loop

The break with labels

```java
public static void main(String[] args) {
    outer: //label for outer loop
    for (int i = 0; i < 10; i++) {
        for (int j = 0; j < 10; j++) {
            if (j == 1)
                break outer;
            System.out.println(" value of j = " + j);
        }
    } //end of outer loop
} // end of main()
```

Continue with label

```
outer:
for (int i = 0; i < 5; i++) {
    for (int j = 0; j < 5; j++) {
        if (j > i) {
            System.out.println("");
            continue outer;
        }
        System.out.print(" " + (i * j));
    }
}
```

Output:

```
 0
 0 1
 0 2 4
 0 3 6 9
 0 4 8 12 16
```

You can use a labeled continue statement to skip an iteration of the outer loop.

The return statement

```
public static void main(String args[]) {
    boolean t = true;
    System.out.println(“Before return statement");
      if(t)   // if commented out it would be an unreachable
              // operator System.out.println compiler error
        return;
      System.out.println(“Program exit");
  }
```

Output:

```
Before return statement
```

return terminates a method

The return statement (2 statements)

```
public static void main(String args[]) {
    System.out.println(someMeth(2, 1));
    System.out.println(someMeth(Integer.MAX_VALUE, 1));
}
public static int someMeth(int a, int b) {
    if (a == Integer.MAX_VALUE || b == Integer.MAX_VALUE) {
        return 0;
    }
    return a + b;
}
```

Output:

```
3
0
```

return terminates a method

Module contents

- Control Flow Statements
  - Identifiers and Literals
  - Local variables: initialization and lifetime
  - Declaring a Variable as a Constant
  - The if-then and if-then-else statements
  - The switch statement
  - Loops: the while, do-while and for statements
  - The break and continue statements
  - The goto keyword
  - Program exit

The goto keyword

- Java has no goto statement .
- The Java keyword list specifies the goto keyword, but it is marked as "not used"
- Studies illustrated that goto is (mis)used more often than not simply "because it's there"
- Multi-level break and continue remove most of the need for goto statements

Studies on approximately 100,000 lines of C code determined that roughly 90 percent of the goto statements were used purely to obtain the effect of breaking out of nested loops

Module contents

- Control Flow Statements
  - Identifiers and Literals
  - Local variables: initialization and lifetime
  - Declaring a Variable as a Constant
  - The if-then and if-then-else statements
  - The switch statement
  - Loops: the while, do-while and for statements
  - The break and continue statements
  - The goto keyword
  - Program exit

Program Exit

A program terminates all its activity and exits when one of two things happens:

- All the threads that are not daemon threads terminate.

- Some thread invokes the exit method of class Runtime or class System, and the exit operation is not forbidden by the security manager.
- You can use System.exit(0) to close the program