For Loop

- For loop executes group of Java statements as long as the boolean condition evaluates to true.
- For loop combines three elements which we generally use: initialization statement, boolean expression and increment or decrement statement.

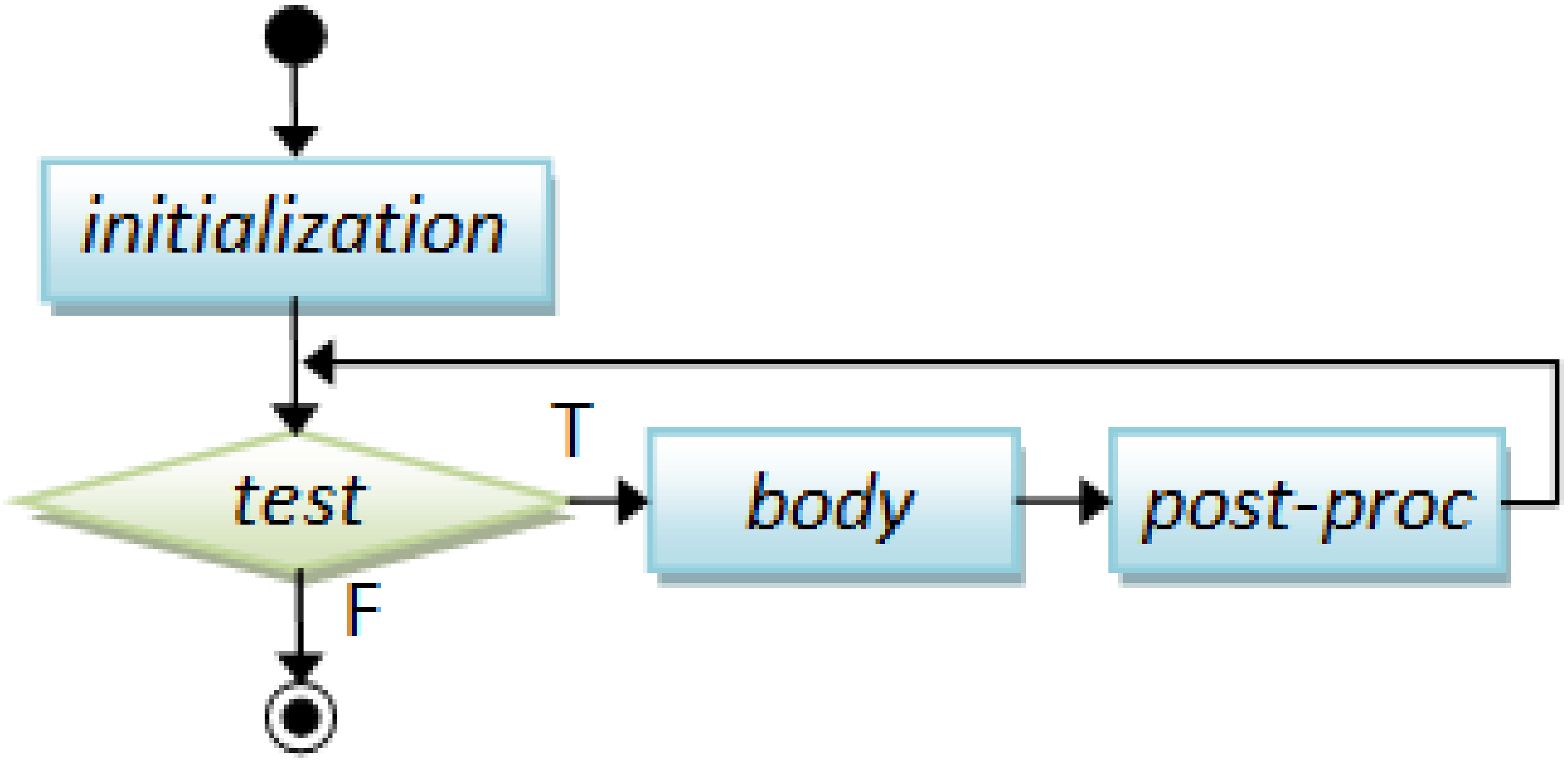
For Loop

- For loop syntax

```
for( <initialization> ; <condition> ; <statement> )
   {
                 …….
          <Block of statements>;
                 …….
```

- }

For Loop



For Loop

- The initialization statement is executed before the loop starts.
- It is generally used to initialize the loop variable.
- Condition statement is evaluated before each time the block of statements are executed.
- Block of statements are executed only if the boolean condition evaluates to true.
- Statement is executed after the loop body is done.
- Generally it is being used to increment or decrement the loop variable.

For Loop

- Following example shows use of simple for loop.
- for(int i=0 ; i < 5 ; i++)

```
{
    System.out.println(“i is : “ + i);
}
```

- It is possible to initialize multiple variable in the initialization block of the for loop by separating it by comma as given in the below example.

  ```
  For(i=0,j=5;i<5;i++)
  ```

  It is also possible to have more than one increment or decrement section as well as given below.

  ```
  for(int i=0; i < 5 ; i++, j--)
  ```

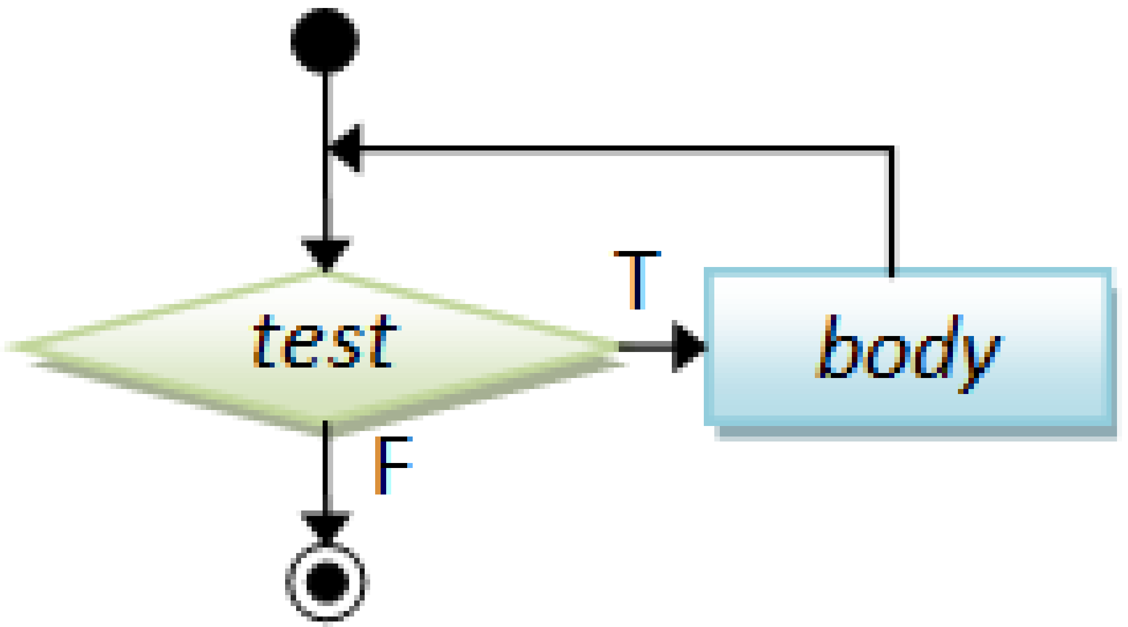
The while Loop

- The general form of the while loop is

```
while(condition) {
        statement;
        }
```

- The condition can be any Boolean expression. The body of the loop will be executed as long as the conditional expression is true.
- When condition becomes false, control passes to the next line of code immediately following the loop.
- The curly braces are unnecessary if only a single statement is being repeated.

The while Loop



The while Loop

```
• class While {
  public static void main(String args[]) {
        int n = 10;
        while(n > 0) {
              System.out.println("tick " + n);
              n--;
              }
        }
• }
```

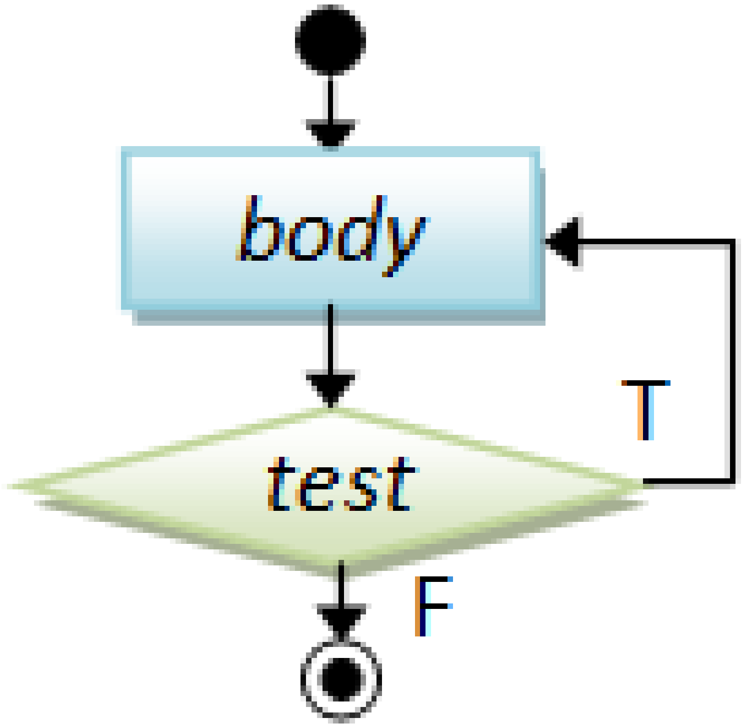
do-while Loop

- general form is

```
do {
        // body of loop
} while (condition);
```

- Each iteration of the do-while loop first executes the body of the loop and then evaluates the conditional expression.
- If this expression is true, the loop will repeat.
- Otherwise, the loop terminates. As with all of Java’s loops, condition must be a Boolean expression.

do-while Loop



do-while Loop

- The do-while loop always executes its body at least once, because its conditional expression is at the bottom of the loop.

```
class DoWhile {
    public static void main(String args[]) {
        int n = 10;
        do {
            System.out.println("tick " + n);
            n--;
        } while(n > 0);
    }
}
```

Jump Statements

- Java jump statements enable transfer of control to other parts of program.
- Java provides three jump statements:

  1) break

  2) continue

  3) return
- In addition, Java supports exception handling that can also alter the control flow of a program.

1. break Statement

- The break statement has three uses:

  1) to terminate a case inside the switch statement

  2) to exit an iterative statement

  3) to transfer control to another statement

  (1) has been described.
- We continue with (2) and (3).

Using break to Exit a Loop

- By using break, there will an immediate termination of a loop, bypassing the conditional expression and any remaining code in the body of the loop.
- When a break statement is encountered inside a loop, the loop is terminated and program control resumes at the next statement following the loop.

Loop Exit with break

- When break is used inside a loop, the loop terminates and control is transferred to the following instruction.

-
  ```
  class BreakLoop {
          public static void main(String args[]) {
                  for (int i=0; i<100; i++) {
                  if (i == 10) break;
                          System.out.println("i: " + i);
                  }
                  System.out.println("Loop complete.");
          }
  ```
- }

break in Nested Loops

- Used inside nested loops, break will only terminate the innermost loop:

```
class NestedLoopBreak {
	public static void main(String args[]) {
	for (int i=0; i<3; i++) {
		System.out.print("Pass " + i + ": ");
		for (int j=0; j<100; j++) {
			if (j == 10) break; System.out.print(j + " ");
		}
		System.out.println();
	}
	System.out.println("Loops complete.");
	}
```

- }

Control Transfer with break

- Java does not have an unrestricted "goto" statement, which tends to produce code that is hard to understand and maintain.
- However, in some places, the use of gotos is well justified. In particular, when breaking out from the deeply nested blocks of code.
- break occurs in two versions:

    1) unlabelled

    2) labeled

- The labeled break statement is a "civilized" replacement for goto.

Labeled break

- General form:

```
break label;
```

- where label is the name of a label that identifies a block of code:

```
label: { … }
```

- The effect of executing break label; is to transfer control immediately after the block of code identified by label.

Example: Labeled break

```
class Break {
        public static void main(String args[]) {
        boolean t = true;
        first: {
            second: {
                       third: {
                            System.out.println("Before the break.");
                            if (t) break second;
                            System.out.println("This won't execute");
                            }
                            System.out.println("This won't execute");
                        }
                  System.out.println(“After second block.");
                  }
        }
 }
```

Example: Nested Loop break

```
class NestedLoopBreak {
  public static void main(String args[]) {
        outer: for (int i=0; i<3; i++) {
        System.out.print("Pass " + i + ": ");
          for (int j=0; j<100; j++) {
            if (j == 10) break outer; // exit both loops
               System.out.print(j + " ");
               }
        System.out.println("This will not print");
  }
  System.out.println("Loops complete.");
  }
}
```

break Without Label

- It is not possible to break to any label which is not defined for an enclosing block.
- Trying to do so will result in a compiler error.

```
class BreakError {
        public static void main(String args[]) {
                one: for(int i=0; i<3; i++) {
                        System.out.print("Pass " + i + ": ");
                }
                for (int j=0; j<100; j++) {
                if (j == 10) break one;
                        System.out.print(j + " ");
                }
        }
```

- }

continue Statement

- The break statement terminates the block of code, in particular it terminates the execution of an iterative statement.
- The continue statement forces the early termination of the current iteration to begin immediately the next iteration.
- Like break, continue has two versions:

  1) unlabelled – continue with the next iteration of the current loop

  2) labeled – specifies which enclosing loop to continue

Example: Unlabeled continue

```
• class Continue {
        public static void main(String args[]) {
        for (int i=0; i<10; i++) {
                System.out.print(i + " ");
                if (i%2 == 0) continue;
                        System.out.println("");
                }
        }
• }
```

Example: Labeled continue

```
class LabeledContinue {
  public static void main(String args[]) {
      outer: for (int i=0; i<10; i++) {
            for (int j=0; j<10; j++) {
                  if (j > i) {
                        System.out.println();
                        continue outer;
                  }
                  System.out.print(" " + (i * j));
            }
      }
      System.out.println();
      }
```

- }

Return Statement

- The return statement is used to return from the current method: it causes program control to transfer back to the caller of the method.
- Two forms:
- 1) return without value

```
return;
```

- 2) return with value

```
return expression;
```

Example: Return

```
• class Return {
        public static void main(String args[]) {
                boolean t = true;
                System.out.println("Before the return.");
                if (t) return; // return to caller
                System.out.println("This won't execute.");
        }
• }
```