LOOPING STATEMENTS

REPETITION STATEMENTS

- *REPETITION STATEMENTS* ALLOW US TO EXECUTE A STATEMENT MULTIPLE TIMES
- OFTEN THEY ARE REFERRED TO AS *LOOPS*
- LIKE CONDITIONAL STATEMENTS, THEY ARE CONTROLLED BY BOOLEAN EXPRESSIONS
- JAVA HAS THREE KINDS OF REPETITION STATEMENTS:
  - THE *WHILE LOOP*
  - THE *DO LOOP*
  - THE *FOR LOOP*

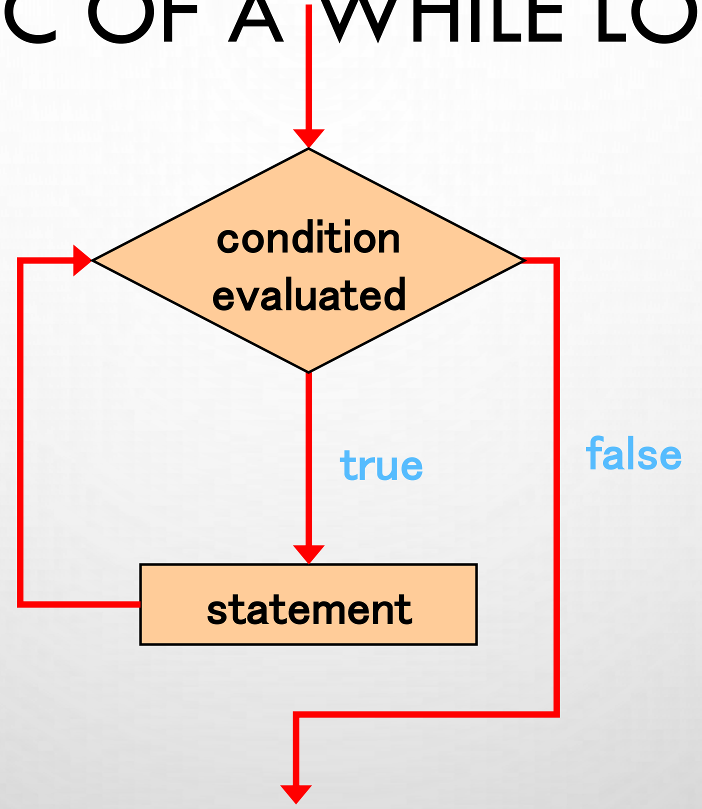
THE WHILE STATEMENT

- A *WHILE STATEMENT* HAS THE FOLLOWING SYNTAX:

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

- **If the `condition` is true, the `statement` is executed**
- **Then the condition is evaluated again, and if it is still true, the statement is executed again**
- **The statement is executed repeatedly until the condition becomes false**

LOGIC OF A WHILE LOOP



THE WHILE STATEMENT

- AN EXAMPLE OF A WHILE STATEMENT:

```
int count = 1;
while (count <= 5){
   System.out.println (count);
   count++;
}
```

- **If the condition of a `while` loop is false initially, the statement is <u>never executed</u>**
- **Therefore, the body of a `while` loop will execute zero or more times**

THE WHILE STATEMENT

- LET'S LOOK AT SOME EXAMPLES OF LOOP PROCESSING
- A LOOP CAN BE USED TO MAINTAIN A *RUNNING SUM*
- A *SENTINEL VALUE* IS A SPECIAL INPUT VALUE THAT REPRESENTS THE END OF INPUT
- A LOOP CAN ALSO BE USED FOR *INPUT VALIDATION*, MAKING A PROGRAM MORE *ROBUST*

INFINITE LOOPS

- THE BODY OF A `WHILE` LOOP EVENTUALLY MUST MAKE THE CONDITION FALSE
- IF NOT, IT IS CALLED AN *INFINITE LOOP*, WHICH WILL EXECUTE UNTIL THE USER INTERRUPTS THE PROGRAM
- THIS IS A COMMON LOGICAL (SEMANTIC) ERROR
- YOU SHOULD ALWAYS DOUBLE CHECK THE LOGIC OF A PROGRAM TO ENSURE THAT YOUR LOOPS WILL TERMINATE NORMALLY

INFINITE LOOPS

- AN EXAMPLE OF AN INFINITE LOOP:

```
int count = 1;
while (count <= 25){
   System.out.println (count);
   count = count - 1;
}
```

- **This loop will continue executing until interrupted (Control-C) or until an underflow error occurs**

NESTED LOOPS

- SIMILAR TO NESTED `IF` STATEMENTS, LOOPS CAN BE NESTED AS WELL
- THAT IS, THE BODY OF A LOOP CAN CONTAIN ANOTHER LOOP
- FOR EACH ITERATION OF THE OUTER LOOP, THE INNER LOOP ITERATES COMPLETELY
- YOUR SECOND COURSE PROJECT INVOLVES A `WHILE` LOOP NESTED INSIDE OF A `FOR` LOOP

NESTED LOOPS

- HOW MANY TIMES WILL THE STRING "HERE" BE PRINTED?

```
count1 = 1;
while (count1 <= 10){
   count2 = 1;
   while (count2 <= 20)    {
      System.out.println ("Here");
      count2++;
   }
   count1++;
}
```

10 * 20 = 200

OUTLINE

➡ **The `while` Statement**

Other Repetition Statements

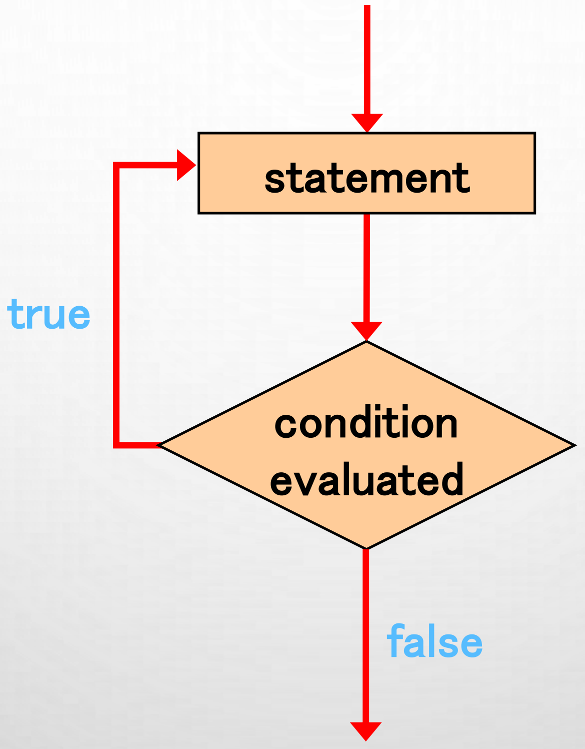
THE DO-WHILE STATEMENT

- A *DO-WHILE STATEMENT* (ALSO CALLED A DO LOOP) HAS THE FOLLOWING SYNTAX:

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

- **The `statement` is executed once initially, and then the `condition` is evaluated**
- **The statement is executed repeatedly until the condition becomes false**

LOGIC OF A DO-WHILE LOOP



THE DO STATEMENT

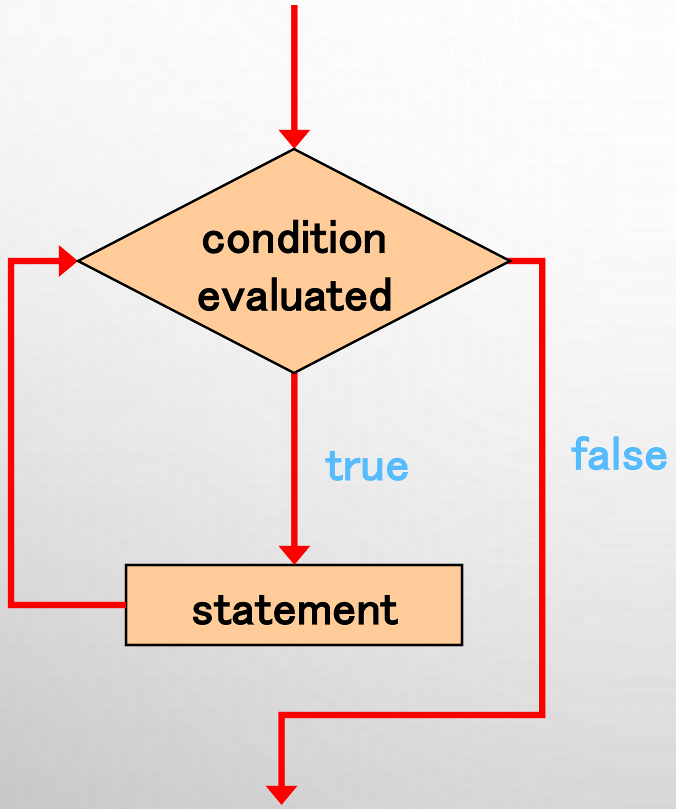
- AN EXAMPLE OF A `DO` LOOP:

```
int count = 0;
do{
   count++;
   System.out.println (count);
} while (count < 5);
```

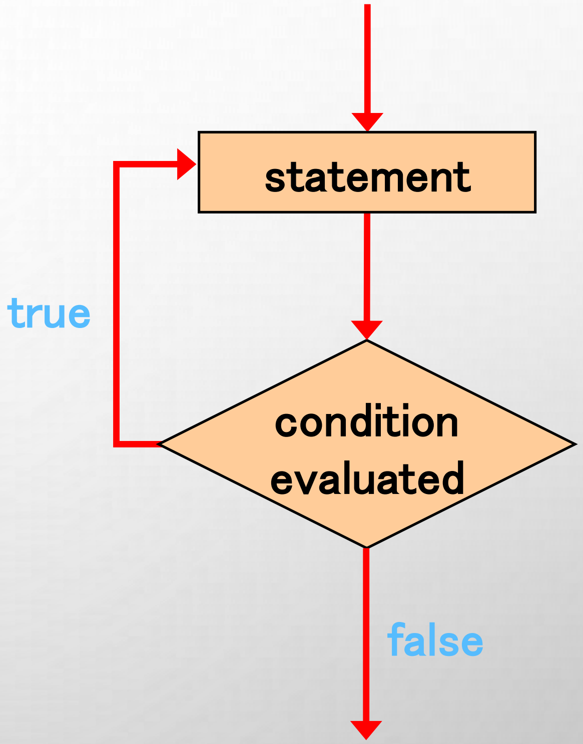
- **The body of a `do` loop executes at least once**

COMPARING WHILE AND DO

The while Loop



The do Loop



THE FOR STATEMENT

- A *FOR STATEMENT* HAS THE FOLLOWING SYNTAX:

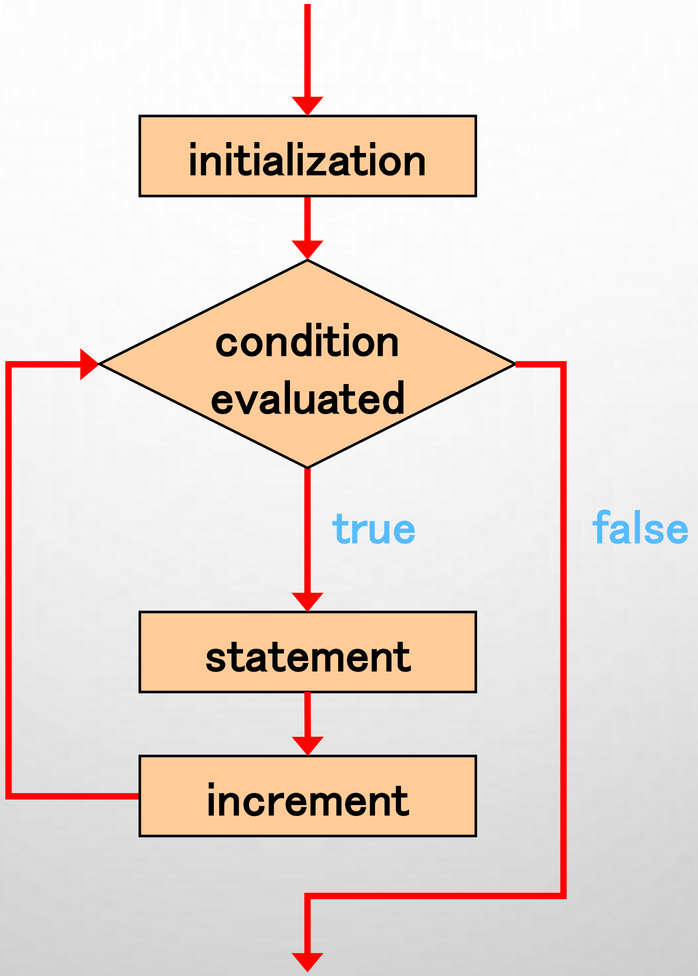
The *initialization* is executed once before the loop begins

The *statement* is executed until the *condition* becomes false

```
for ( initialization ; condition ; increment ){
    statement;
}
```

The *increment* portion is executed at the end of each iteration

LOGIC OF A FOR LOOP



THE FOR STATEMENT

- A `FOR` LOOP IS FUNCTIONALLY EQUIVALENT TO THE FOLLOWING `WHILE` LOOP STRUCTURE:

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

THE FOR STATEMENT

- AN EXAMPLE OF A `FOR` LOOP:

```
for (int count=1; count <= 5; count++){
   System.out.println (count);
}
```

- **The initialization section can be used to declare a variable**
- **Like a `while` loop, the condition of a `for` loop is tested <u>prior</u> to executing the loop body**
- **Therefore, the body of a `for` loop will execute zero or more times**

THE FOR STATEMENT

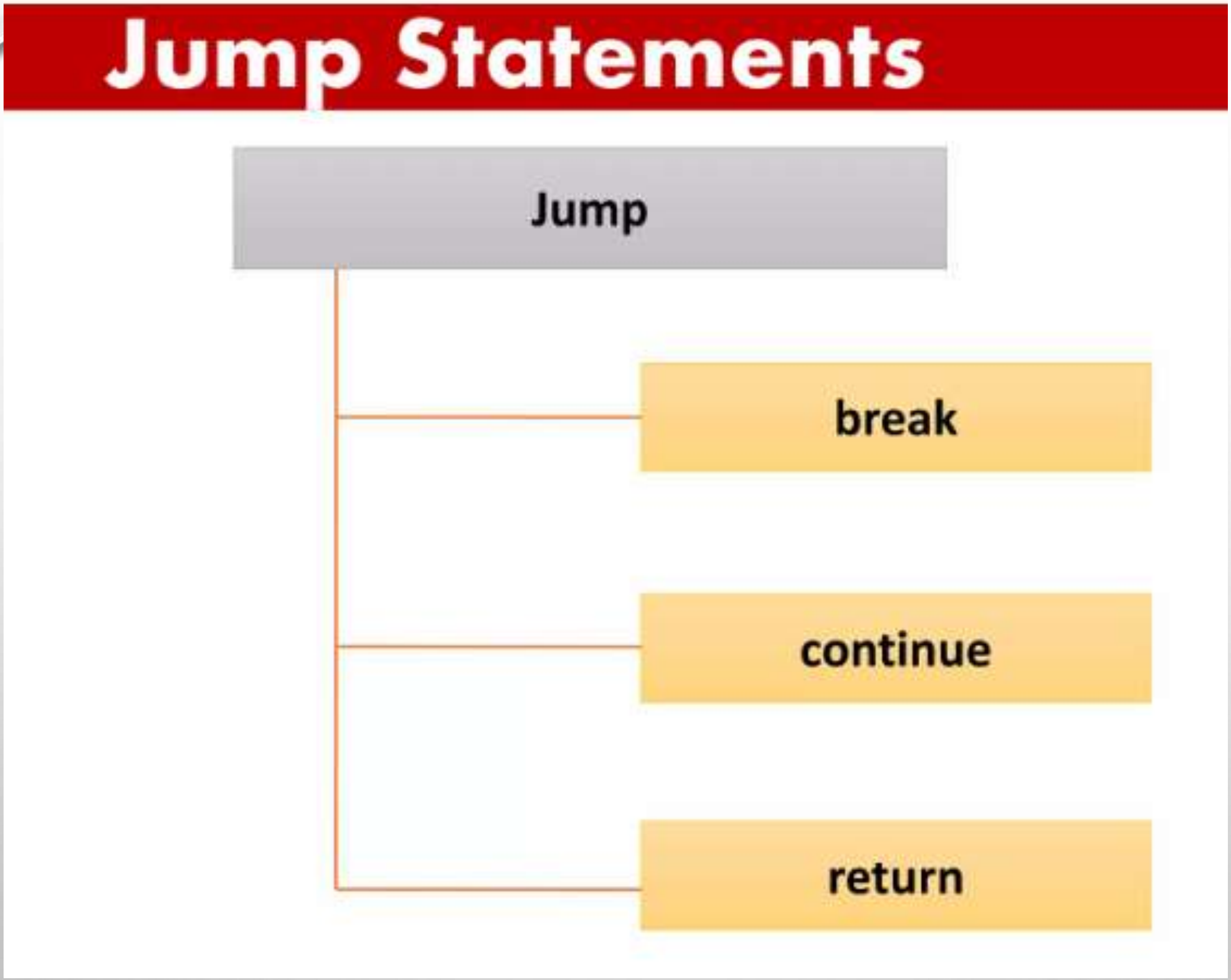
- THE INCREMENT SECTION CAN PERFORM ANY CALCULATION

```
for (int num=100; num > 0; num -= 5){
    System.out.println (num);
}
```

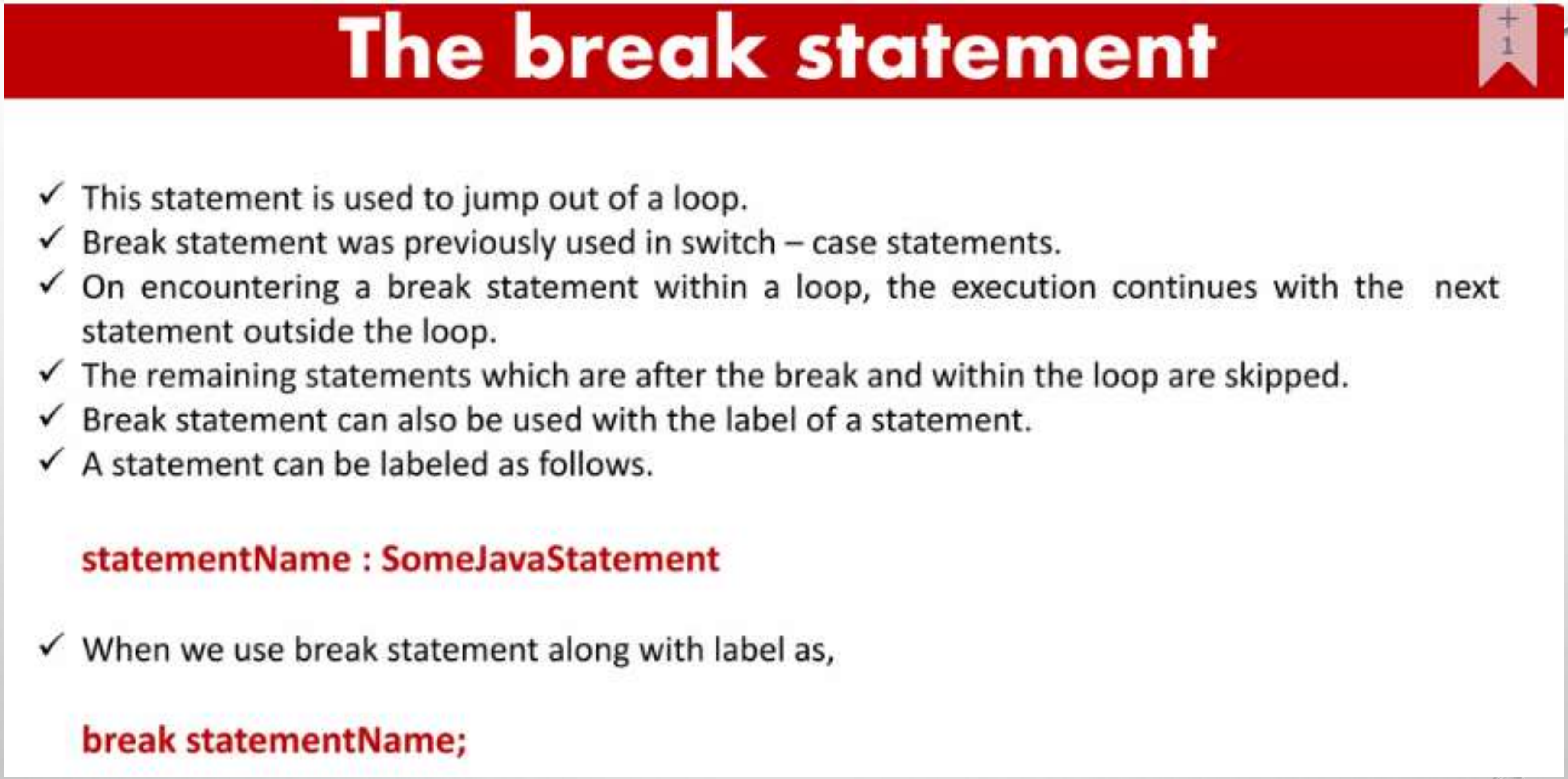
- **A `for` loop is well suited for executing statements a *specific number of times* that can be calculated or determined *in advance***

THE FOR STATEMENT

- EACH EXPRESSION IN THE HEADER OF A `FOR` LOOP IS <u>OPTIONAL</u>
- IF THE INITIALIZATION IS LEFT OUT, NO INITIALIZATION IS PERFORMED
- IF THE CONDITION IS LEFT OUT, IT IS ALWAYS CONSIDERED TO BE TRUE, AND THEREFORE CREATES AN <u>INFINITE LOOP</u>
    - WE USUALLY CALL THIS A “FOREVER LOOP”
- IF THE INCREMENT IS LEFT OUT, NO INCREMENT OPERATION IS PERFORMED



Jump Statements



The break statement

- ✓ This statement is used to jump out of a loop.
- ✓ Break statement was previously used in switch – case statements.
- ✓ On encountering a break statement within a loop, the execution continues with the next statement outside the loop.
- ✓ The remaining statements which are after the break and within the loop are skipped.
- ✓ Break statement can also be used with the label of a statement.
- ✓ A statement can be labeled as follows.

statementName : SomeJavaStatement

- ✓ When we use break statement along with label as,

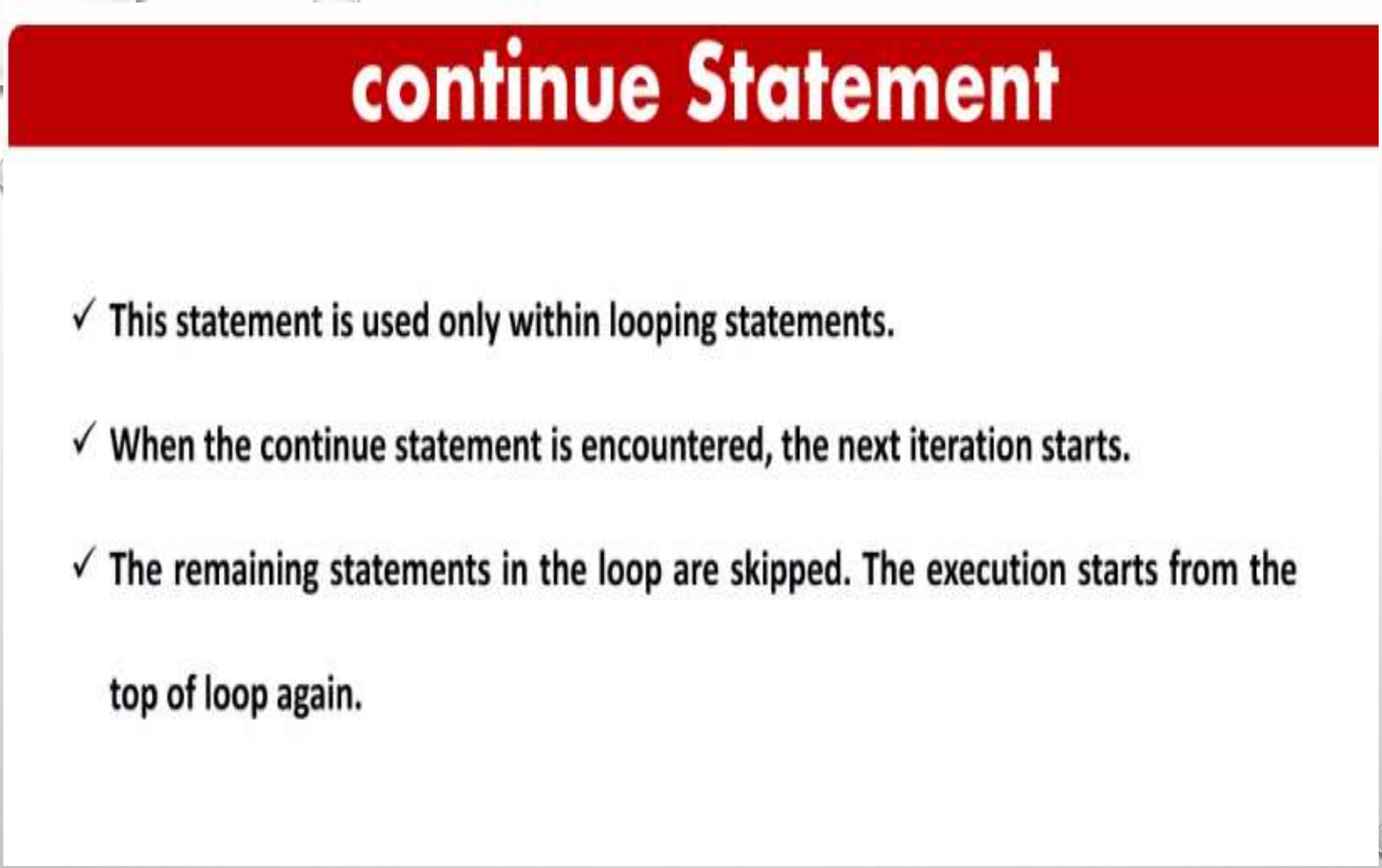
break statementName;

Example

```
class break1
{
    public static void main(String args[])
    {
        int i = 1;
        while (i<=10)
        {
            System.out.println("\n" + i);
            i++;
            if (i==5)
            {
                break;
            }
        }
    }
}
```

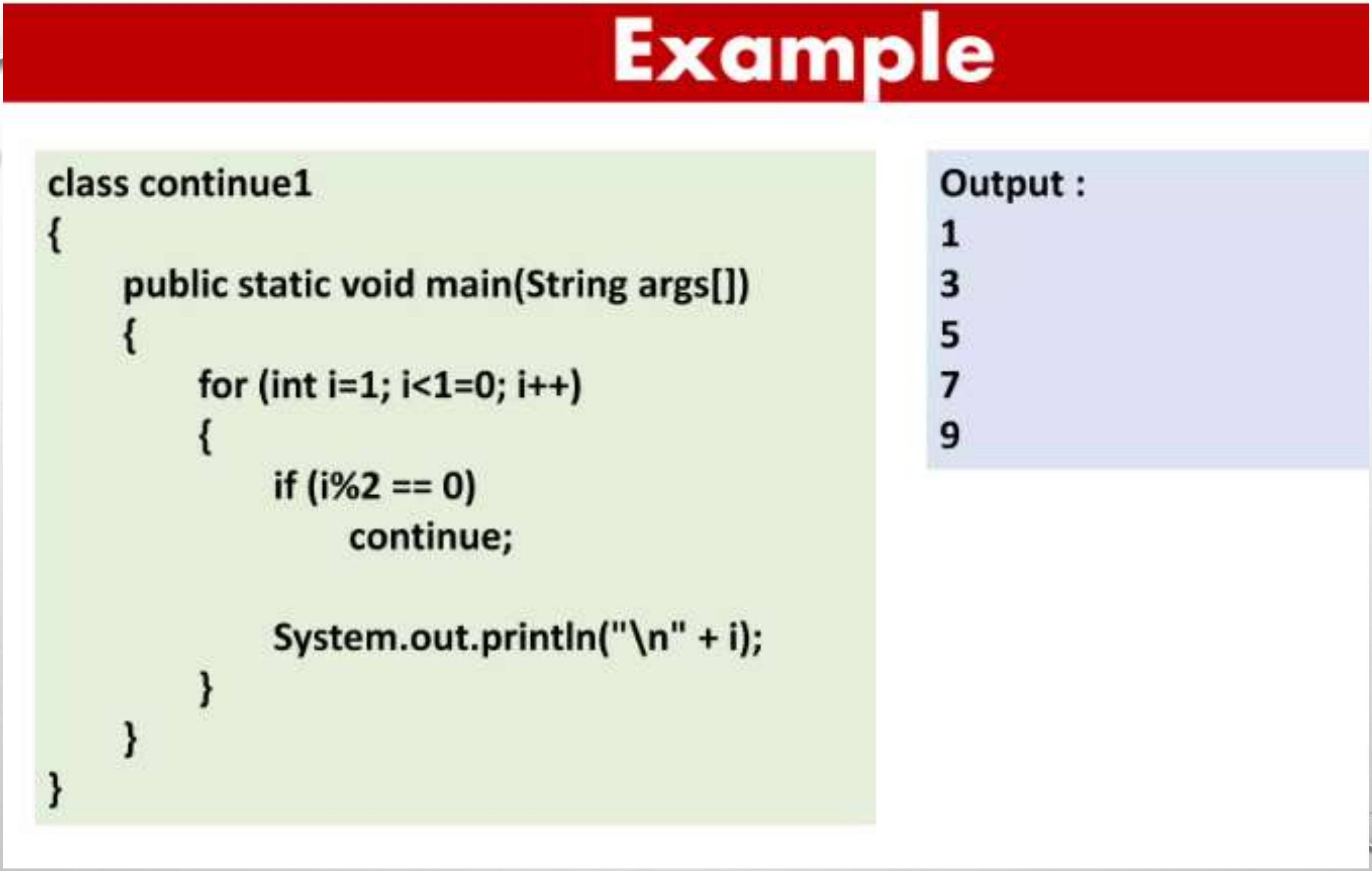
Output :

```
1
2
3
4
```



continue Statement

- ✓ This statement is used only within looping statements.
- ✓ When the continue statement is encountered, the next iteration starts.
- ✓ The remaining statements in the loop are skipped. The execution starts from the top of loop again.



Example

```
class continue1
{
        public static void main(String args[])
        {
                for (int i=1; i<1=0; i++)
                {
                        if (i%2 == 0)
                                continue;

                        System.out.println("\n" + i);
                }
        }
}
```

```
Output :
1
3
5
7
9
```

The return Statement

- ✓ The last control statement is return. The return statement is used to explicitly return from a method.
- ✓ That is, it causes program control to transfer back to the caller of the method.
- ✓ The return statement immediately terminates the method in which it is executed.

Example

```
class Return1
{
      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.");
     }
}
```

Output :

```
Before the return.
```