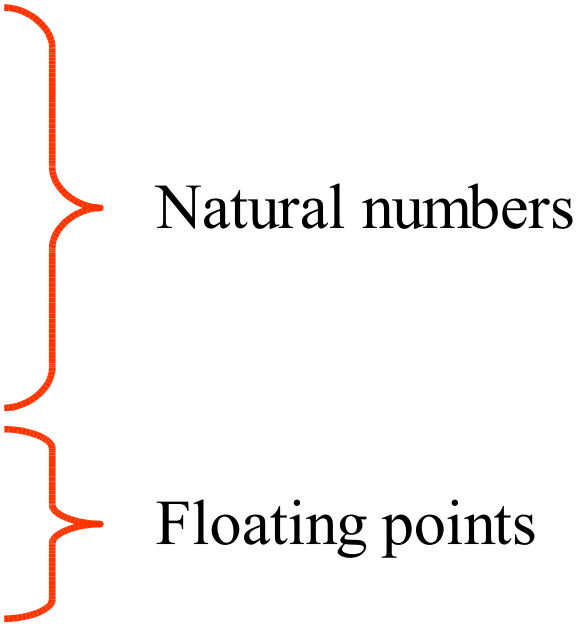
The Basic Parts of Java

- Data Types
  - Primitive
    - int, float, double, etc.
  - Composite
    - array (will also be covered in the lecture on Collections)
- Lexical Rules
- Expressions and operators
- Methods
  - Parameter list
  - Argument parsing
- Control Structures
- Branching

- Examples in http://www.cs.auc.dk/~torp/Teaching/E03/OOP/Examples/

Primitive Data Types

- **`Boolean`** {true, false}
- **`byte`** 8-bit
- **`short`** 16-bit
- **`int`** 32-bit
- **`long`** 64-bit
- **`float`** 32-bit
- **`double`** 64-bit
- **`char`** 16-bit Uni-code



- Also called *built-in types*
- Have fixed size on *all* platforms

Declarations

- A *declaration* is the introduction of a new name in a program.
- All variables must declared in advance.
- The is no dedicated variable declaration part of a Java program.
- General forms

  type variableName1, variableName2, variableName3;

  type variableName1 = value1,
  variableName2 = value2,
  variableName3 = value3;

- Constants are declared as **`final static`** variables

Primitive Data Types, Example

```
// create some integers
int x, y;
x = 1234; y = 3;

// or similar
double v = 3.14e-23,
       w = 5.5;

// create som chars
char c1 = 'a';
Character c2;

// use a wrapper class
c2 = new Character ('b'); // read only

// A well-known constant
final static double PI = 3.14;
```

Array: A Composite Data Type

- An array is an indexed sequence of values of the same type.
- Arrays are defined as classes in Java.

- Example:

```
boolean[] boolTable = new boolean[MAXSIZE]
```

  - Elements are all of type **`boolean`**
  - The index type is always integer
  - Index limits from 0 to **`MAXSIZE-1`**
- Bound-check at run-time.
- Arrays are first class objects (not pointers like in C)

- There are no record or enumaration types in Java.

Lexical Rules

- A name in Java consists of [0-9][a-z][A-Z][_$]
  - name cannot start with a number
  - national language letters can be used, e.g., æ, ø , and å.
  - no maximum length **`thisIsAVeryLongVariableName`**

- All resevered word in Java are lower case, e.g., **`if.`**
- Case matters **`myVariable, myvariable`**

Naming Conventions

- Words run together, no underscore
- Intermediate words capitalized.
  - Okay: **`noOfDays, capacity, noInSequence`**
  - Not okay **`no_of_days, noofdays`**

- Name of classes: first letter upper case
  - Okay: **`Person, Pet, Car, SiteMap`**
  - Not okay: **`vehicle, site_map, siteMap`**
- Name of method or variable: first letter lower case
- Name of constants: all upper case, separated by underscore

- Part of JavaSoft programming standard
  Java's Naming convention (link)

Commands in Java

- Assignment
  - **`variable = <expression>`**
- Method call
  - Various parameter mechanisms
- Control Structures
  - sequential
  - selective
  - iterative

Block Statement

- Several statements can be grouped together into a *block statement*.
- A block is delimited by braces `{ <statement list> }`
- Variables can be declared in a block.
- A block statement can be used wherever a statement is called for in the Java syntax.
  - For example, in an *if-else statement*, the if portion, or the else portion, or both, could be block statements

Expresions and Operators

- An *expression* is a program fragment that evaluates to a single value.
  - `double d = v + 9 * getSalary() % Math.PI;`
  - `e = e + 1;` (here `e` is used both as an *rvalue* and a *lvalue*)

- Arithmetic operators
  - Additive +, -, ++, -- `i = i + 1, i++, --i`
  - Multicative *, /, % (mod operator) `9%2 = 1, 7%4 = 3`

- Relational Operators
  - Equality == (two '=' symbols) `i = i, i == i`
  - Inequality != `i != j`
  - Greater-than >, >= `i > j, i >= j`
  - Less-than <, <= `i < j, i <= j`

Expresions and Operators, cont.

- Logical operators
  - and **&&** `bool1 && bool2`
  - or **||** `bool1 || bool2 || bool3`
  - not **!** `!(bool1)`
  - All are *short-circuit*

- Bitwise operators
  - and **&** `255 & 5 = 5` `15 & 128 = 0`
  - or **|** `255 | 5 = 255` `8 & 2 = 10`
  - xor **^** `3 ^ 8 = 11` `16 ^ 31 = 15`
  - shift left << `16 << 2 = 64` `7 << 3 = 56`
  - shift right >> `16 >> 2 = 4` `7 >> 2 = 1`

Expresions and Operators, cont.

- Assignement Operators
  - can be combined with other binary operators
  - +=, -=, *=, /=, %=, >>=, <<=, &=, ^=, !=

- Conditional Operator
  - Ternary operator
  - ?:
  - `int max = n > m ? n : m;`

- Precendence rules similar to C for Java operators
- Associtivity rules similar to C for Java operators

Methods in Java

- All procedures and functions in Java are methods on classes.
- The difference between a procedure and a function is the return type
  - **`void myProcedure()`**
  - **`int myFunction()`** or **`MyClass myFunction1()`**
- Methods cannot be nested.
- Returning
  - *Implicit*: When the last command is executed (for procedures).
  - *Explicit*: By using the **`return`** command.
    - Good design: only to have one **`return`** command each method

Methods in Java, cont.

- General format

```
ReturnType methodName (/* <argument list> */){
    // <method body>
}
```

- Examples calling methods

```
double y = getAverageSalary();   // returns double

boolean b = exists (/*args*/);   // returns boolean

Person p = getPerson (/*args*/); // returns Person
```

Class **IPAddress** Example

```
public class IPAddress{
   public static final String DOT = ".";
    private int[] n;          // example 127.0.0.1
    private String logical;   // example localhost
   /* Constructor */
    public IPAddress(){n = new int[4]; logical = null;}
   /* Sets the logical name */
    public void setName(String name){logical = name;}
    /* Gets the logical name */
    public String getName(){ return logical; }
    /* Sets numerical name */
    public void setNum(int one, int two, int three, int four){
        n[0] = one; n[1] = two; n[2] = three; n[3] = four;}
    /* Sets numerical name */
   public void setNum(int[] num){
      for (int i = 0; i < 4; i++){n[i] = num[i];} }
    /* Gets the numerical name as a string */
   public String getNum(){
      return "" + n[0] + DOT + n[1] + DOT + n[2] + DOT + n[3];    }
```

Class **IPAddress** Example, cont.

```
public static void main (String[] args){
    // create a new IPAddress
    IPAddress luke = new IPAddress();
     luke.setName("luke.cs.auc.dk");
     System.out.println(luke.getName());
     luke.setNum(130, 225, 194, 177);
     String no = luke.getNum();
     System.out.println(no);

    // create another IPAddress
    IPAddress localHost = new IPAddress();
     localHost.setName("localhost");
     int[] lNum = {127, 0, 0, 0}; // array initialization
     localHost.setNum(lNum);
     System.out.print(localHost.getName());
     System.out.print(" ");
     System.out.println(localHost.getNum());
}
```

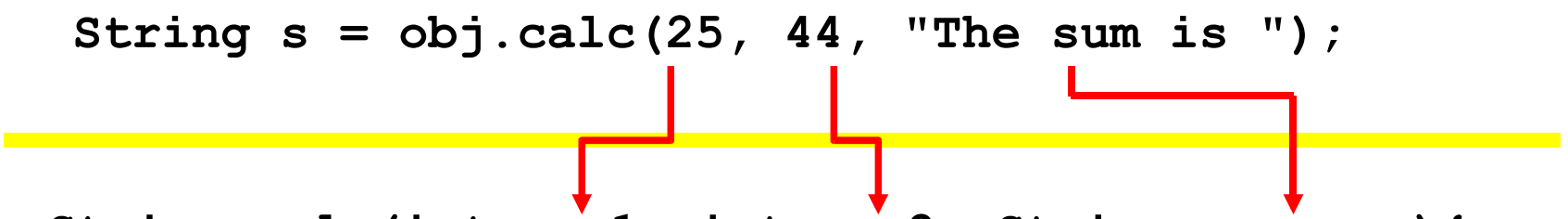
Parameter Mechanism

- All parameters in Java are pass-by-value.
  - The value of the actual parameter is copied to the formal parameter.
- A variable number of arguments in not supported
  - `public static void main (String[] args)`

- Passing Objects
  - Objects are accessed via a reference.
  - References are pass-by-value.
    - The refernce is copied
    - The object itself is not copied
  - Via a formal parameter it is possible to modify the object "directly".
  - The reference to the object can however not be modified.

Actual and Formal Parameters

- Each time a method is called, the actual parameters in the invocation are copied into the formal parameters.



```
String s = obj.calc(25, 44, "The sum is ");
```

```
String calc(int num1, int num2, String message){
   int sum = num1 + num2;
   String result = message + sum
   return result;
}
```

Class **IPAddress** Example, cont.

```
public class IPAddress{
   /* Call by value */
   public int callByValue(int i){ i += 100; return i; }
   /* Call by value */
   public String callByValue(String s){s = "modified string"; return s; }
   /* Call by ref like method */
   public int callByRefLike(int[] a){
       int sum = 0;
      for(int j = 0; j < a.length; j++){ sum += a[j]; a[j] = 255;}
      return sum;
   }
   // in main
   IPAddress random = new IPAddress()
   int dummy = 2;
   random.callByValue(dummy); // dummy unchanged
   String str = "not using new";
   random.callByValue(str); // str unchanged
   int[] ranIPNum = new int[4];
   random.setNum(ranIPNum); // ranIPNUM changed to 255.255.255.255
```

The `static` Keyword

- For data elements
  - Are shared between all the instances of a class
  - `public static int i;`
  - `public static ArrayList = new ArrayList();`
  - `public static final char DOT = '.';`

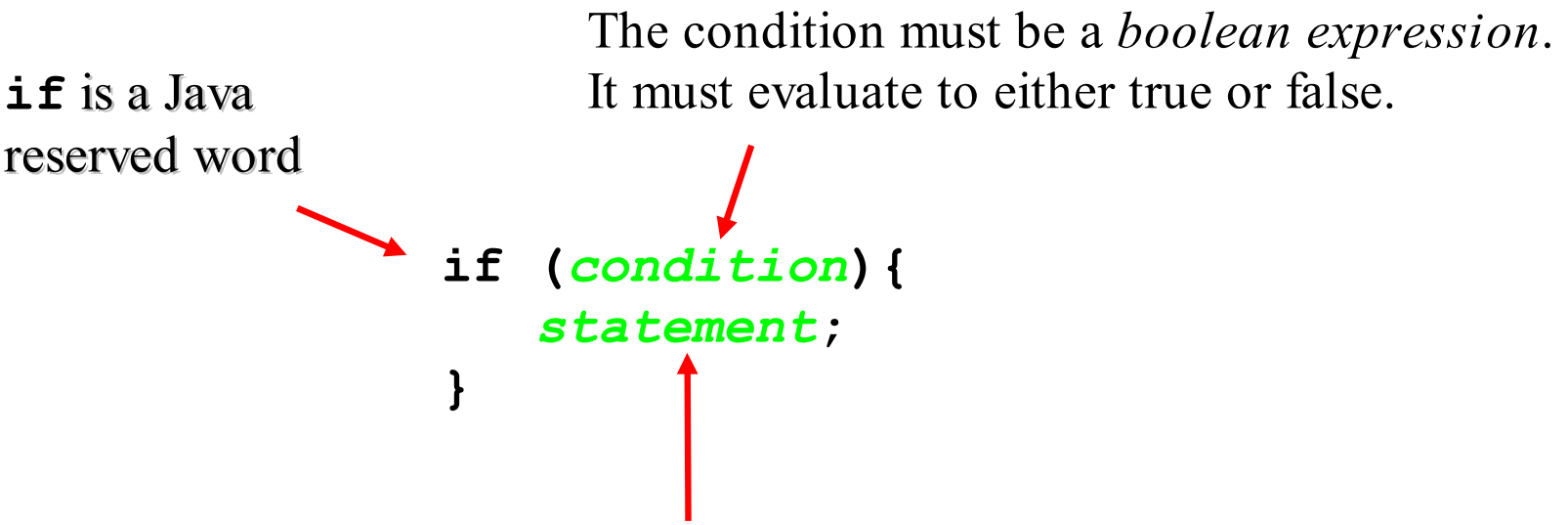
- For method
  - Can be access without using an object
  - `public static void main(String args[]){}`
  - `public static int getCount(){}`

Class **IPAddress** Example, cont.

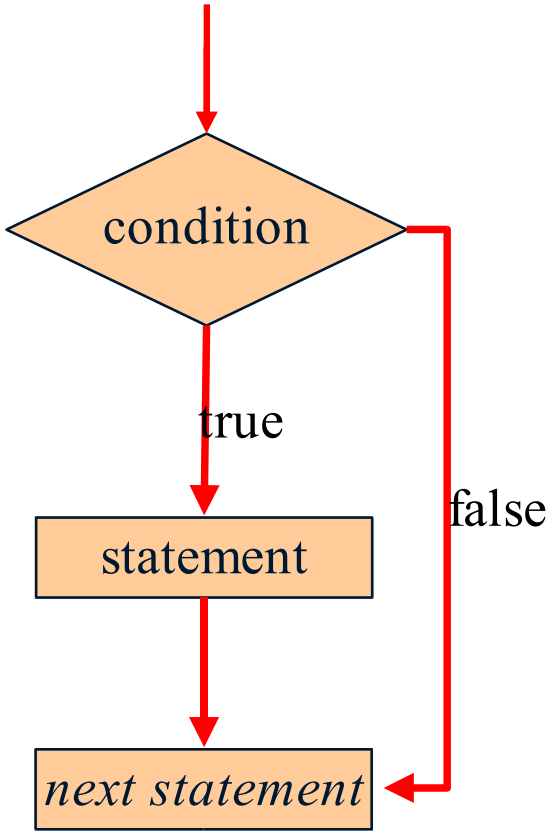
```
public static void main (String[] args){
   private static int count = 0;
   public static final String DOT = ".";
   <snip>
   /* Constructor */
   public IPAddress(){
     n = new int[4];  logical = null;
     count++;}
   /* Get the number of objects created */
   public static int getCount() { return count;}
   <snip>
   /* Handy helper method */
   public static void show(IPAddress ip){
      System.out.print(ip.getName()); System.out.print(" ");
      System.out.println(ip.getNum());
   }
}
```

The `if` Statement

- The *if statement* has the following syntax:



Logic of an `if` Statement




```
// example 1
if (weight < 20000)
  doStuffMethod();

// same thing
if (weight < 20000){
  doStuffMethod();
}
```

```
// example 2
if (weight < 20000)
  doStuffMethod();
  doMoreStuff();

// NOT the same thing
if (weight < 20000){
  doStuffMethod();
  doMoreStuff();
}
```

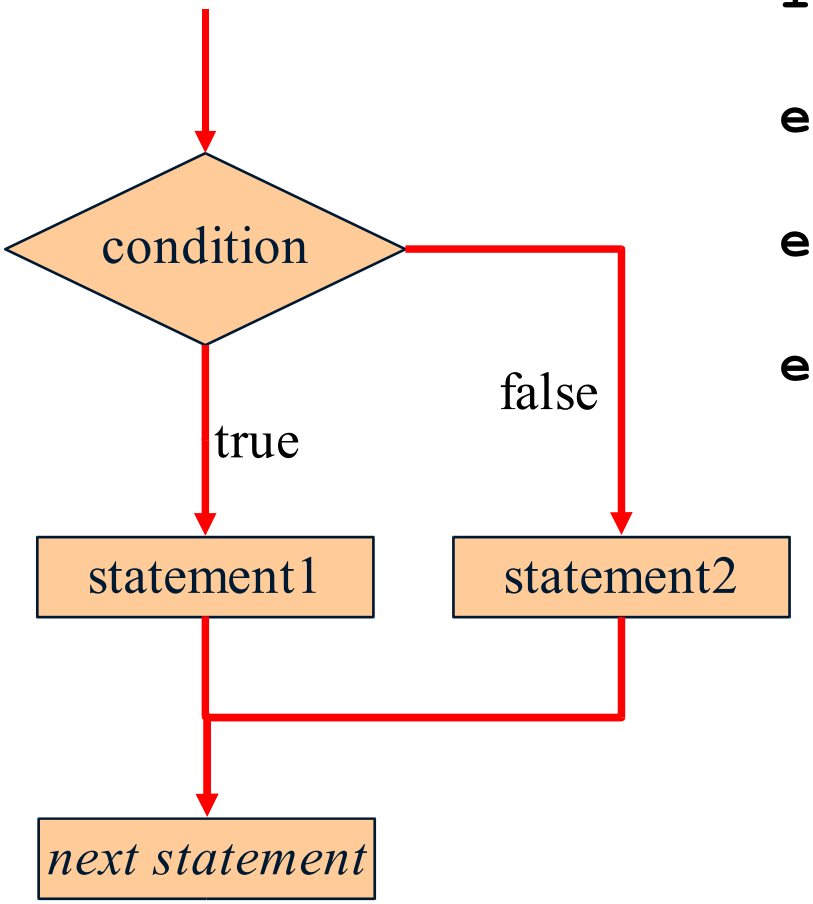
The `if-else` Statement

- An *else clause* can be added to an if statement to make it an *if-else statement*

```
if (condition){
     statement1;
}
else{
     statement2;
}
```

- If the condition is true, *statement1* is executed; if the condition is false, *statement2* is executed
- One or the other will be executed, but not both
- An else clause is matched to the last unmatched if (no matter what the indentation implies)

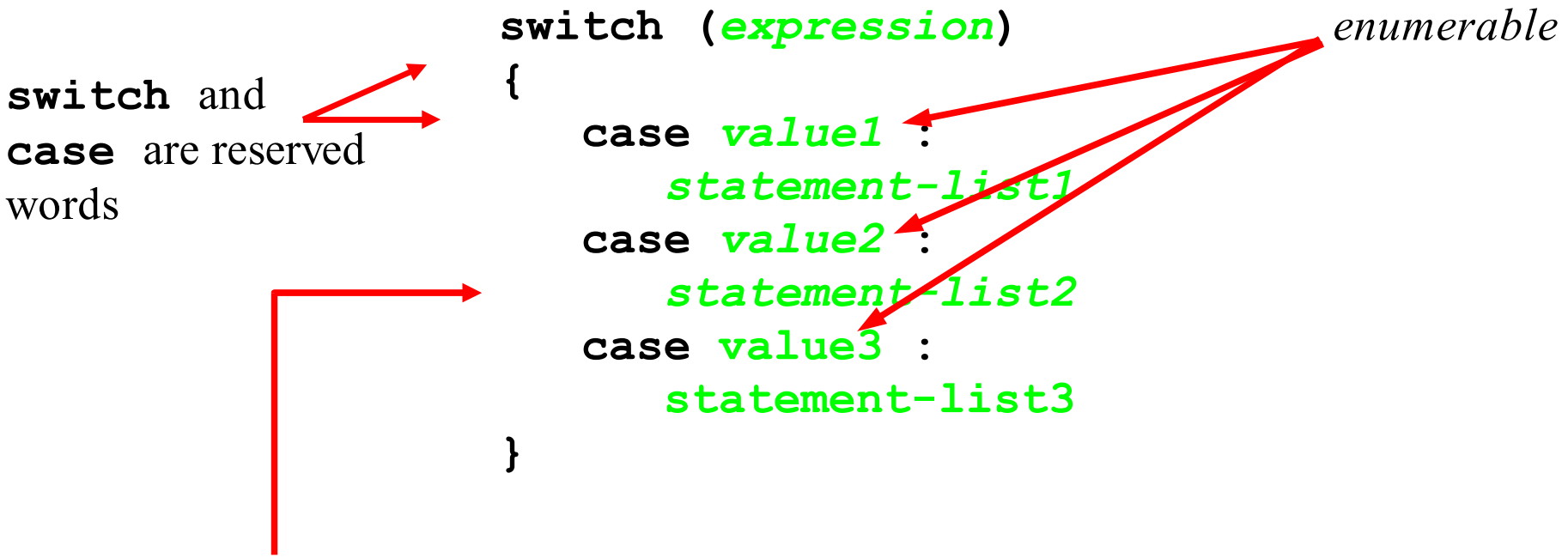
Logic of an `if-else` Statement




```
if (income < 20000)
  System.out.println ("pour");
else if (income < 40000)
  System.out.println ("not so pour")
else if (income < 60000)
  System.out.println ("rich")
else
  System.out.println ("really rich")
```

The `switch` Statement

- The general syntax of a switch statement is



- enumerables can appear in any order
- enumerables do not need to be consecutive
- several case constant may select the same substatement
- enumerables must be distinct
- enumerable cannot case 1..9

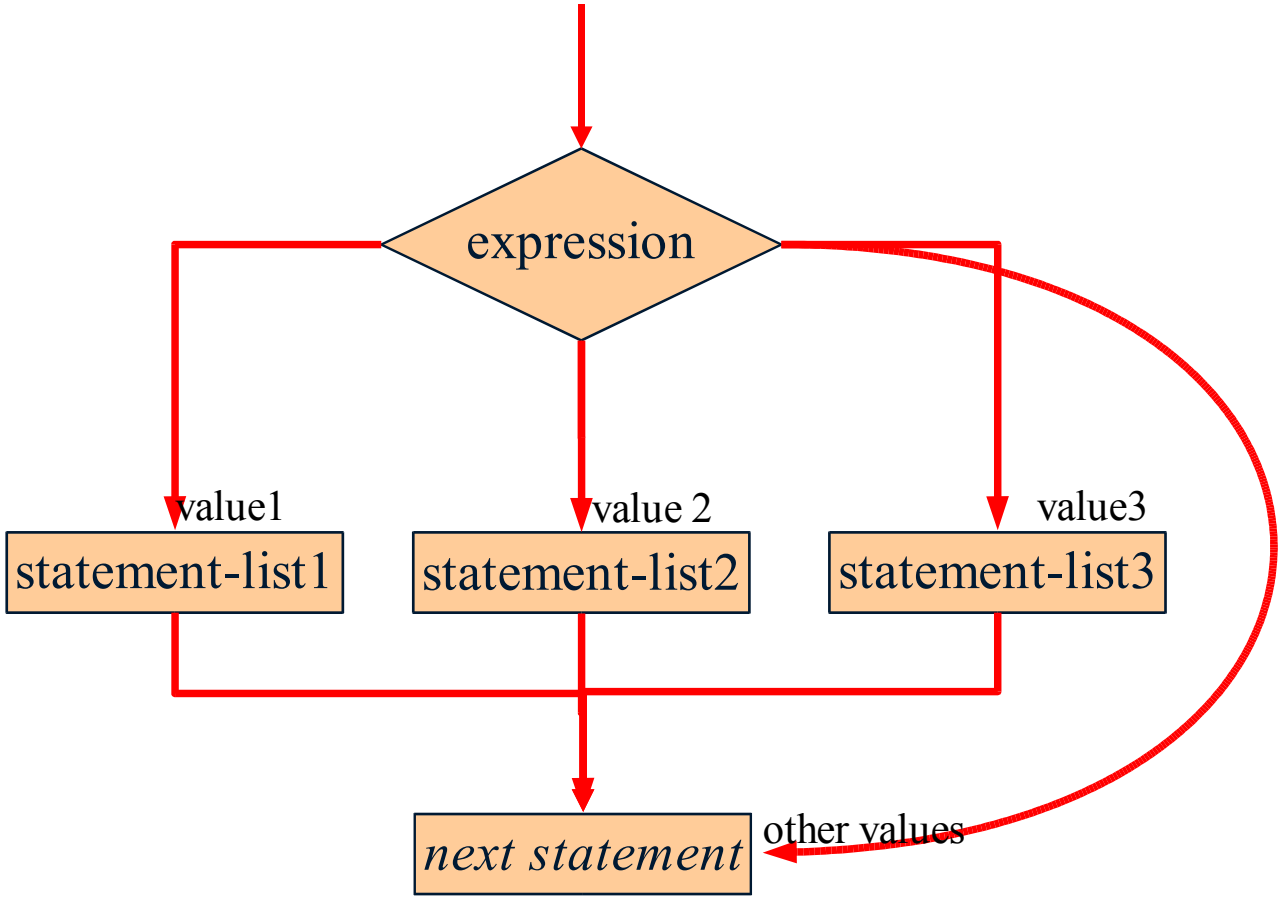
The `switch` Statement, cont.

- Often a *break statement* is used as the last statement in each case's statement list
- A break statement causes control to transfer to the end of the switch statement
- If a break statement is not used, the flow of control will continue into the next case

`break` exits the innermost enclosing loop or **`switch`**

```
switch (expression)
{
    case value1 :
        statement1
        break;
    case value2 :
        statement2
        break;
    case value3 :
        statement3
        break;
}
```

Logic of an **switch** Statement



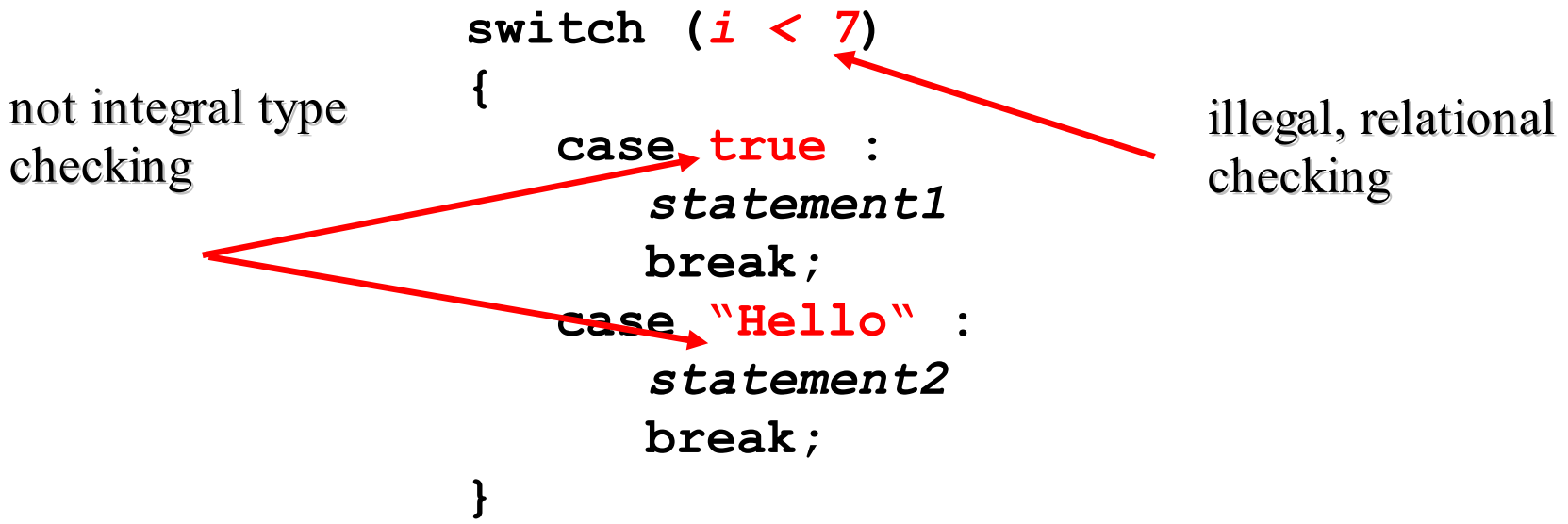

```
switch (expression){
    case value1 :
        statement-list1
        break;
    case value2 :
        statement-list2
        break;
    case value3 :
        statement-list3
        break;
}
// next statement
```

The `switch` Statement, cont.

- A switch statement can have an optional *default case.*
- The default case has no associated value and simply uses the reserved word **`default`.**
- If the default case is present, control will transfer to it if no other case value matches.
- Though the default case can be positioned anywhere in the switch, it is usually placed at the end.
- If there is no default case, and no other value matches, control falls through to the statement after the switch.

The `switch` Statement, cont.

- The expression of a switch statement must result in an *integral data type*, like an integer or character; it cannot be a floating point value.
- Note that the implicit boolean condition in a switch statement is equality - it tries to match the expression with a value.
- You cannot perform relational checks with a switch statement, e.g..



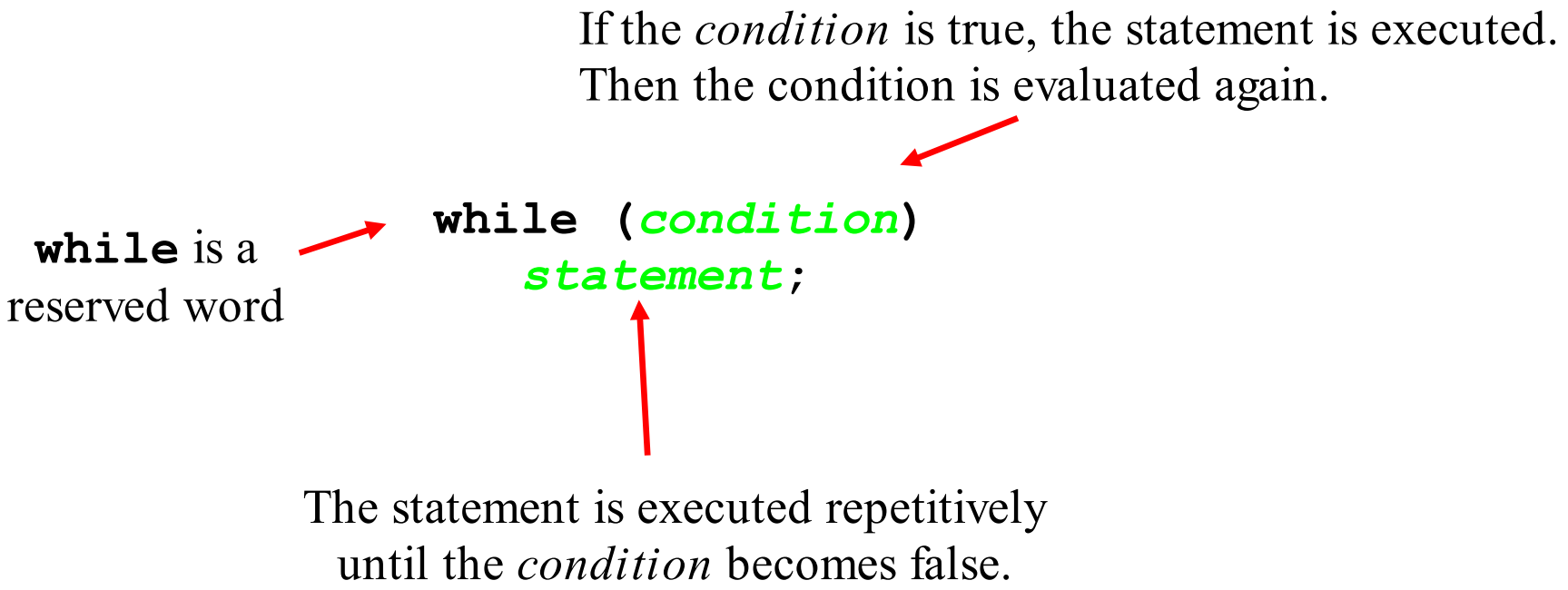
The **`switch`** Statement, Example

```
int salary = getSalary();  // gets a salary

switch(salary/20000) {
     case 0:
          System.out.println("pour");
          break;
     case 1:
          System.out.println("not so pour");
          break;
     case 2:
          System.out.println("rich");
          break;
     case 3:
          System.out.println("really rich");
          break;
     default:
          System.out.println("Hi, Bill Gates");
}
```

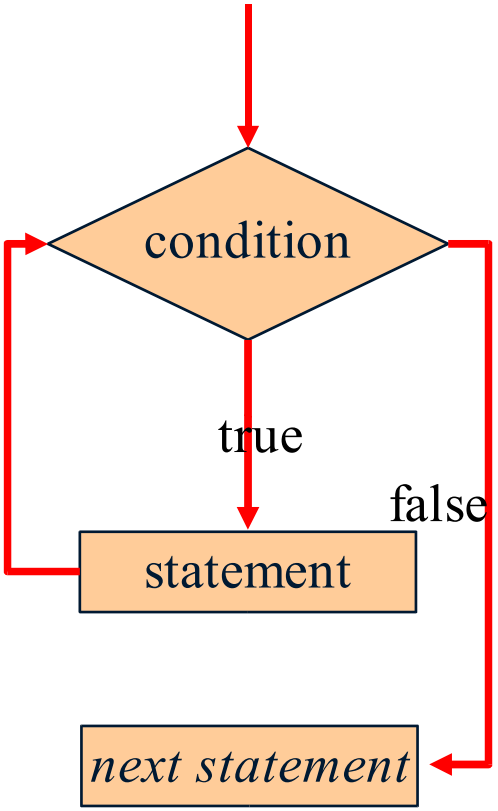
The `while` Statement

- The *while* statement has the following syntax



- Note, if the condition of a while statement is false initially, the statement is never executed
  - Therefore, the body of a while loop will execute zero or more times

Logic of the `while` Statement




```
// Count from 1 to 10
int n = 10;
int i = 1;
while (i <= n) {
     System.out.println(i);
     i = i + 1;
}
// next statement
```

```
// what is wrong here?
int i = 0;
while(i < 10){
     System.out.println(i);
     // do stuff
}
```

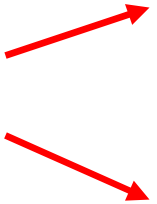
The `while` Statement, cont.

- The body of a while loop must eventually make the *condition* false.
- If not, it is an *infinite loop*, which will execute until the user interrupts the program.
  - This is a common type of logical error.
  - You should always double check to ensure that your loops will terminate normally.
- The while statement can be nested
  - That is, the body of a *while* could contain another loop
  - Each time through the outer *while*, the inner *while* will go through its entire set of iterations

The **do** Statement

- The *do statement* has the following syntax

Uses both the **do** and **while** reserved words



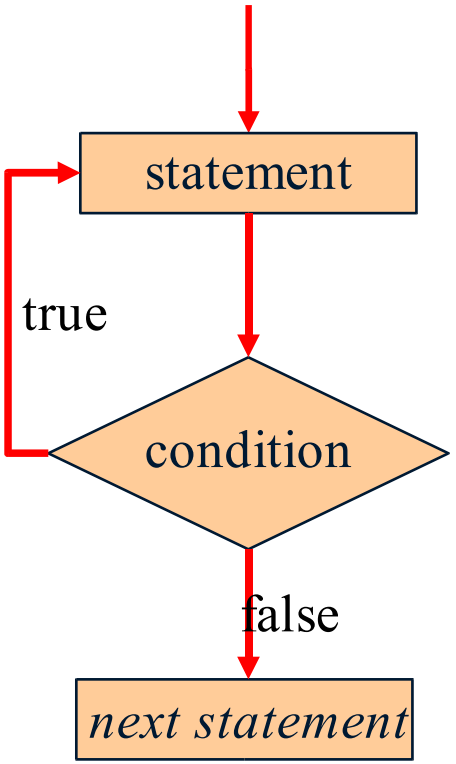
```
do
{
    statement;
}
while (condition)
```

The *statement* is executed once initially, then the condition is evaluated.

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

- A *do* loop is similar to a *while* loop, except that the condition is evaluated after the body of the loop is executed.
  - Therefore the body of a do loop will execute at least one time.

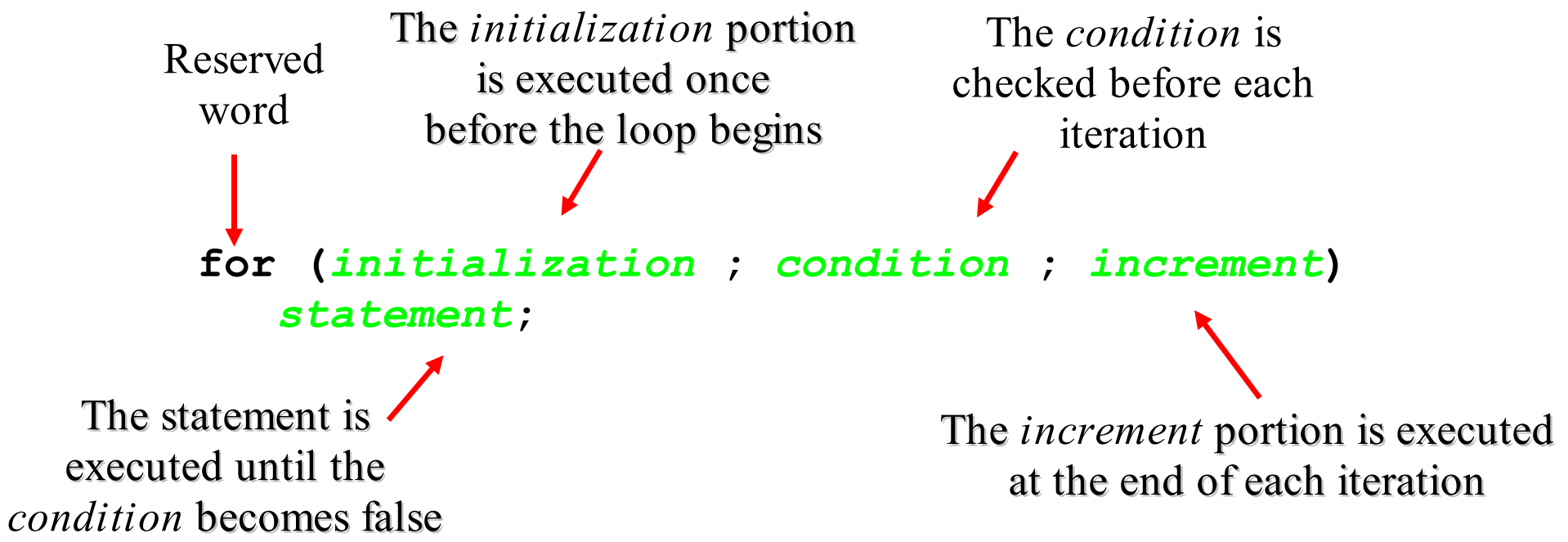
Logic of the **do** Statement




```
// Count from 1 to 10
int n = 10;
int i = 1;
do {
     System.out.println(i)
      i = i + 1;
} while (i <= 10);
// next statement
```

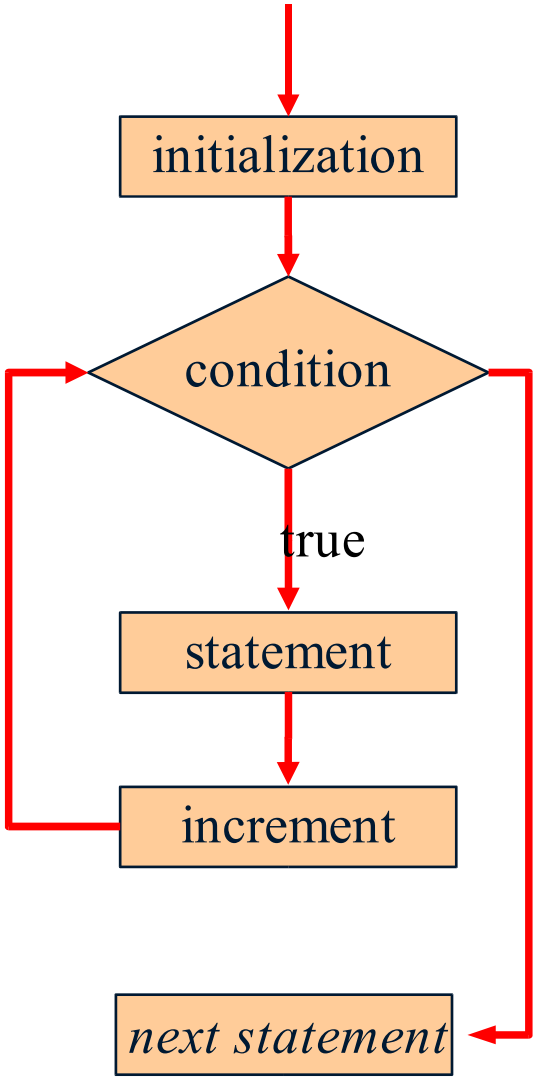
The `for` Statement

- The *for* statement has the following syntax




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

Logic of the `for` Statement



false

```
// Count from 1 to 10
int n = 10;
for (int i = 1; i <= n; i++)
  System.out.println (i);
// next statement
```

```
// what is wrong here?
for (int i=0; i < 10; i++){
    System.out.println(i);
    i--;
}
```

```
// what is wrong here?
for (int i = 0; i < 10;){
  i++;
  // do stuff
}
```

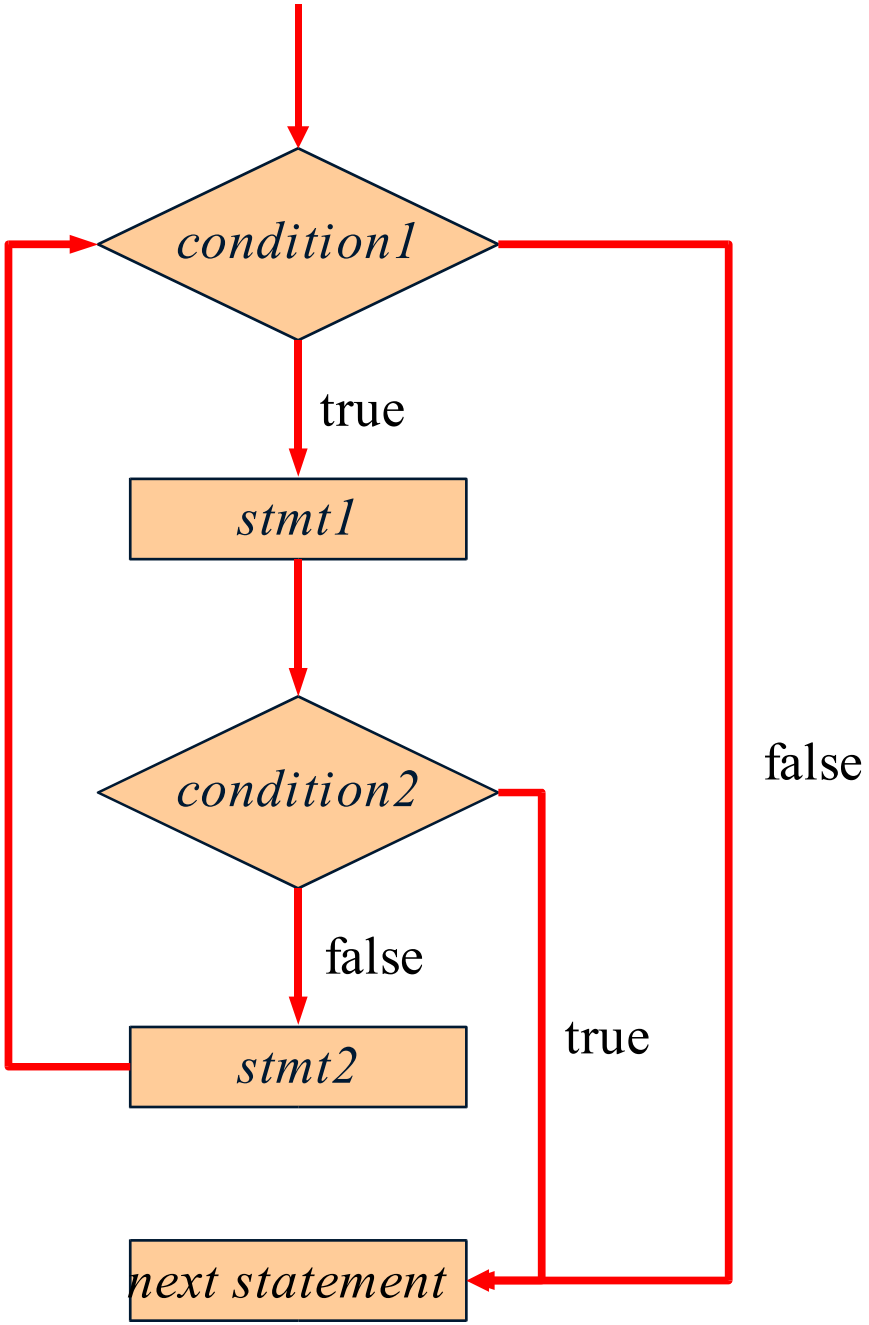
The `for` Statement, cont

- Like a *while* loop, the condition of a *for* statement is tested prior to executing the loop body.
- Therefore, the body of a for loop will execute zero or more times.
- It is well-suited for executing a specific number of times that can be determined in advance.
- Each expression in the header of a for loop is optional
  - Both semi-colons are always required in the for loop header.

Branching

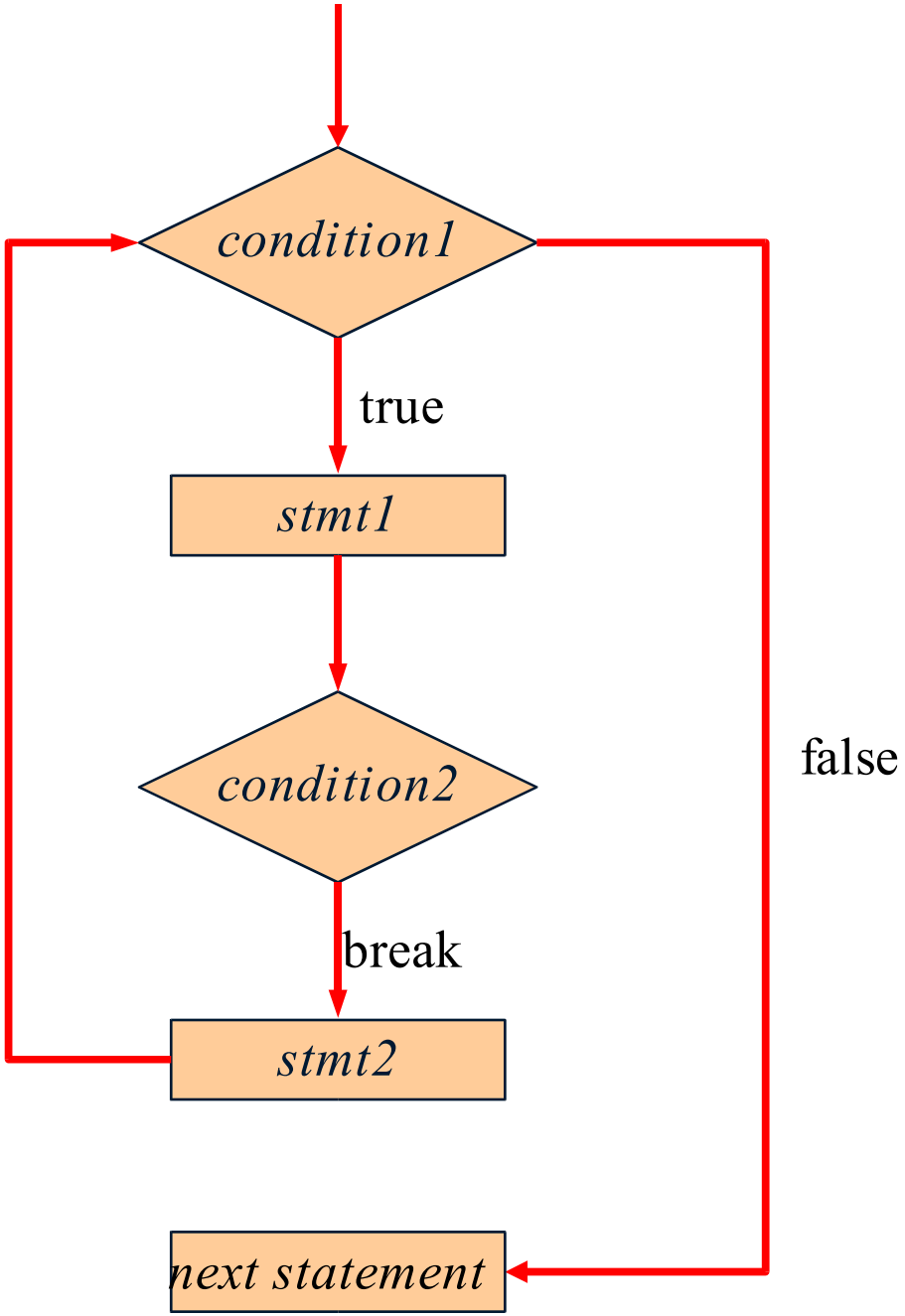
- **`break`**
  - Can be used in any control structure
  - Exits from the innermost enclosing loop
  - **`break <label>`**
- **`continue`**
  - Cycles a loop, e.g., jump to the condition checking
- **`return`**
  - Only from methods;
  - Jumps out of the current method an returns to where the method was called from.
  - **`return <expression>`**
- **`goto`**
  - Reserved word

Logic of the **`break`** Statement



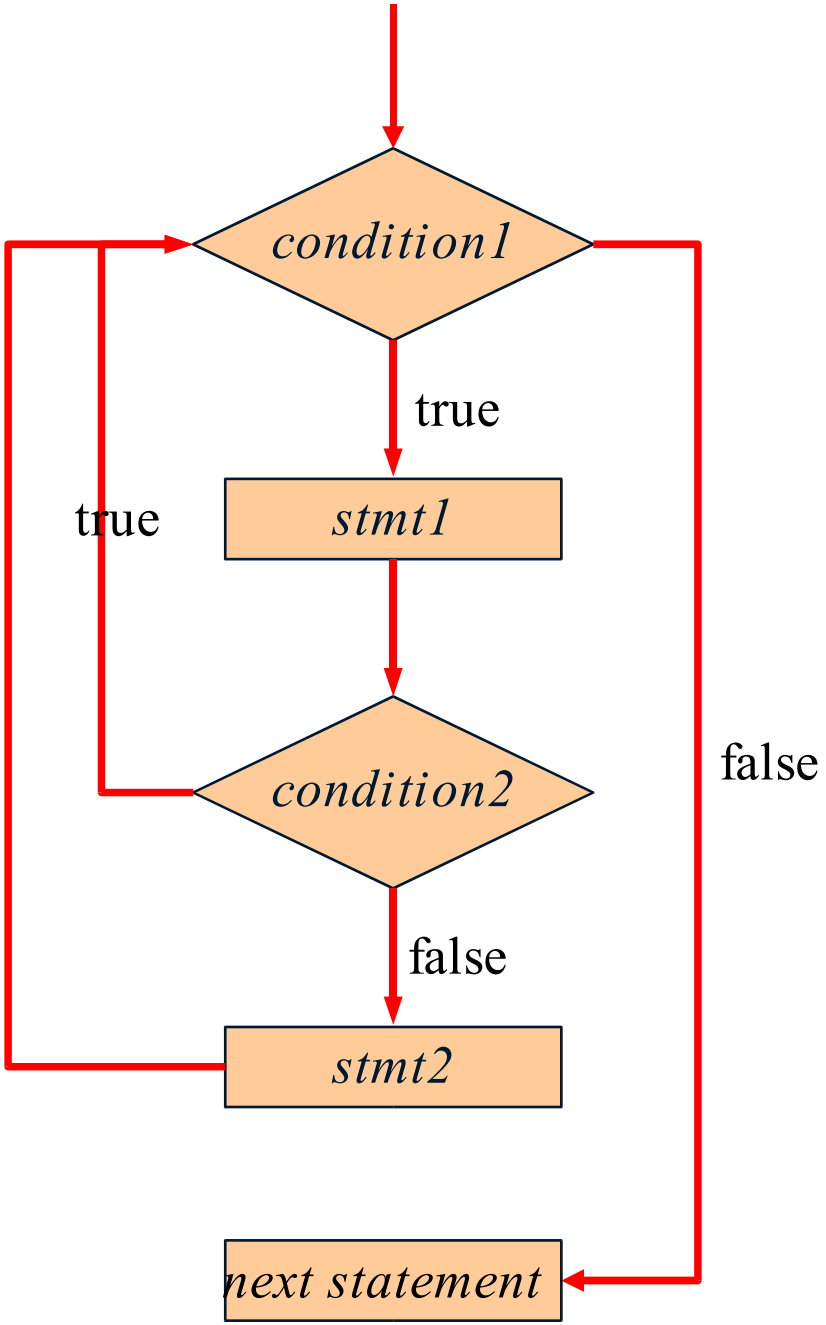

```
while (condition1) {
      stmt1;
      if (condition2)
        break;
      stmt2;
}
// next statement
```

Logic of the **`break`** Statement, cont




```
while (condition1) {
      stmt1;
      while (true){
            break;
      }
      stmt2;
}
// next statement
```

Logic of the **`continue`** Statement




```
while (condition1) {
      stmt1;
      if (condition2)
        continue;
      stmt2;
}
// next statement
```

```
// what is wrong here?
while (condition){
      // many more statements
      continue;
}
```

`continue` Example

```
public void skipPrinting(int x, int y){
    for(int num = 1; num <= 100; num++){
          if((num % x) == 0){
                continue;
          }
          if((num % y) == 0){
                continue;
          }
          // This num is not divisible by x or y
          System.out.println(num);
    }
}
```

`break` and `continue` Example

```
for (int i = 3; i <= max; i++) {
    // skip even numbers
    if (i % 2 == 0)
        continue;
    // check uneven numbers
    boolean isPrime = true;
    for (int j = 2; j < i - 1; j++) {
        // is i diviseable with any number in [2..i-i]
        // then it is not a prime number so we break
        // of efficiency reasons
        if (i % j == 0) {
            isPrime = false;
            break;
        }
    }
    if (isPrime)
        System.out.println(i + " is a prime number");
}
```

Summary

- Set of built-in data types
- Array are supported
  - no support records or enumarated type
- Methods
  - procedure
  - functions
- Argument passing
  - Always by-value in Java
  - actual and formal parameters.
- Control structures
  - if, if-else, if-else-if-else, if-else-if-else-if-else, etc.
  - while-do, do-while
  - for
  - switch