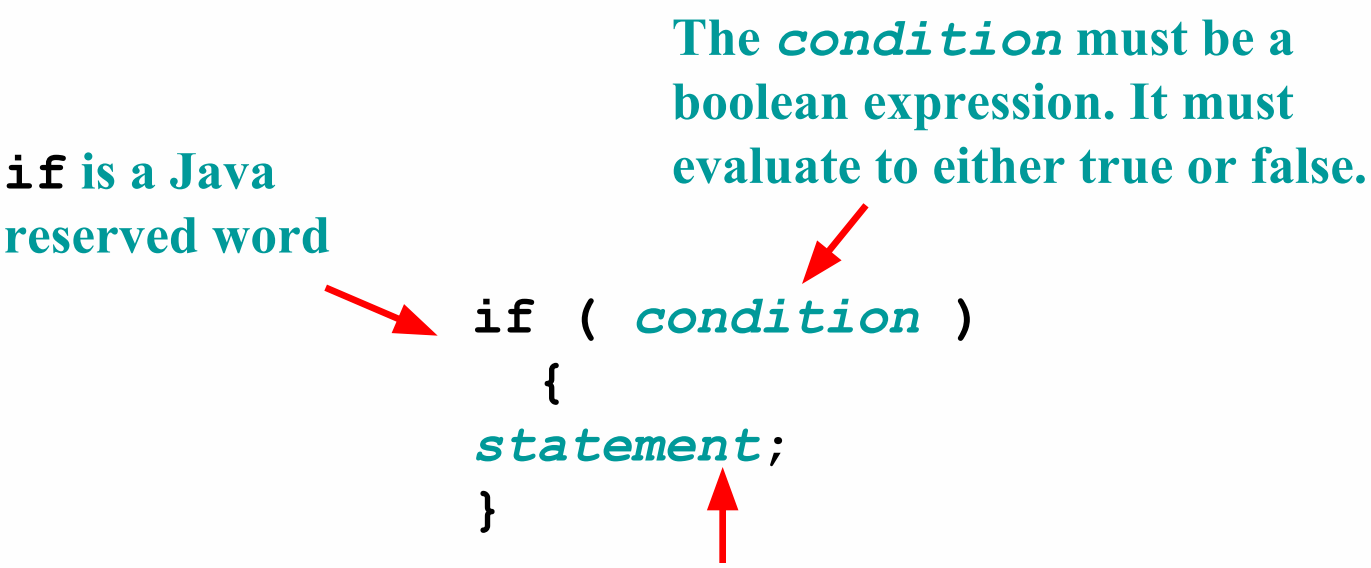
Conditional Statement

Conditional Statements

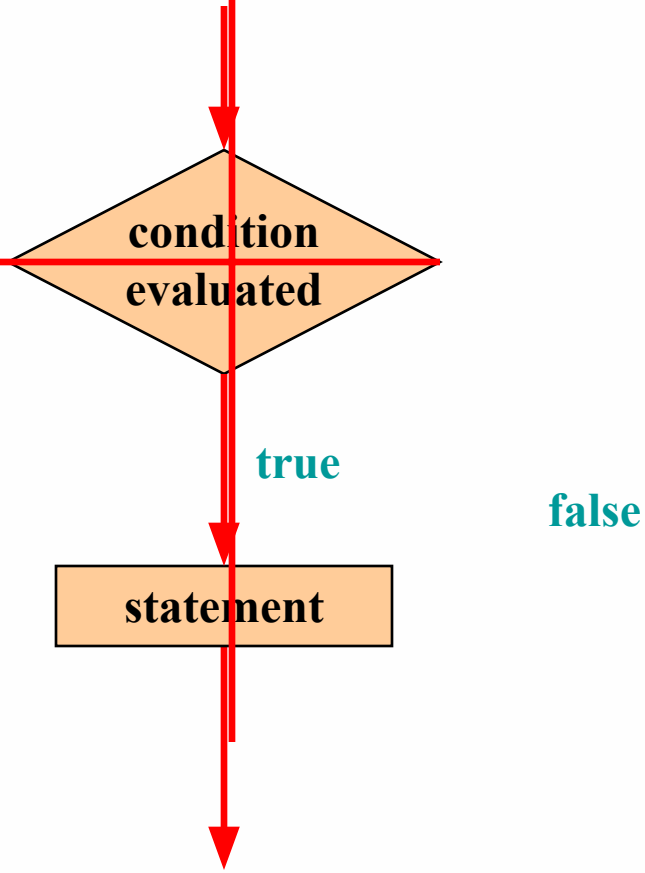
- A *conditional statement* lets us **choose which statement will be executed next**.
- Therefore they are sometimes called ***selection statements***
- Conditional statements give us the power to **make basic decisions**
- **The Java conditional statements are the:**
  - *if statement*
  - *if-else statement*
  - *Nested if-else statement*
  - *switch statement*

The if Statement

- The *if statement* has the following syntax:



Logic of an if statement



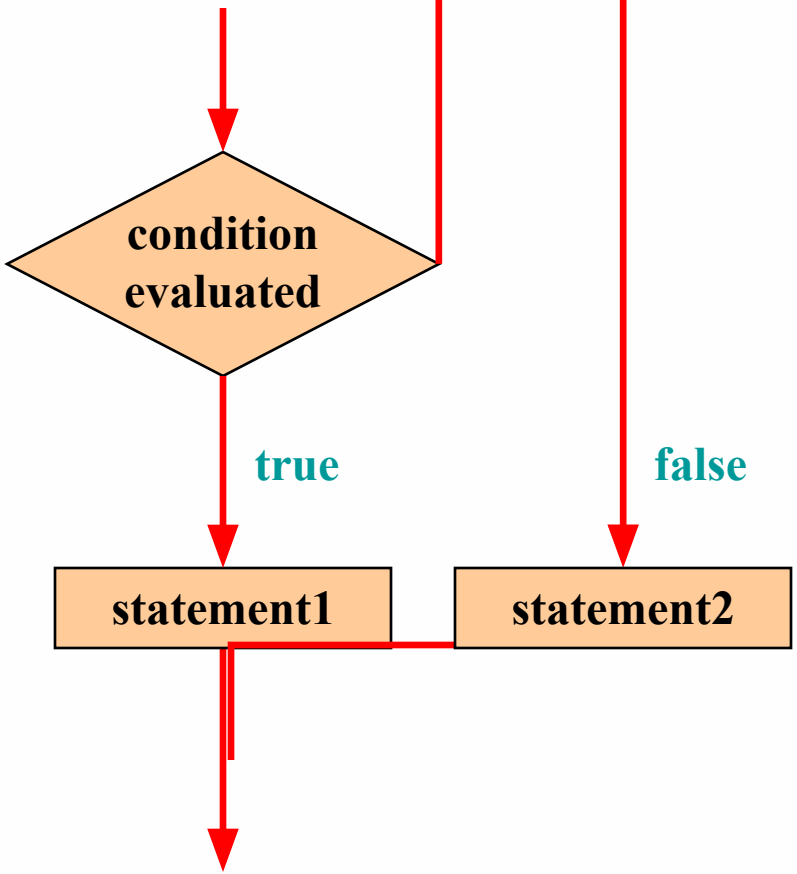
The if-else Statement

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

Logic of an if-else statement



Nested if Statements

- The statement executed as a result of an `if` statement or `else` clause could be another `if` statement
- These are called *nested if statements*
- An `else` clause is matched to the last unmatched `if` (no matter what the indentation implies)
- Braces can be used to specify the `if` statement to which an `else` clause belongs

The switch Statement

- The *switch statement* provides another way to decide which statement to execute next
- The `switch` statement evaluates an expression, then attempts to match the result to one of several possible *cases*
- Each case contains a value and a list of statements
- The flow of control transfers to statement associated with the first case value that matches

The switch Statement

- The general syntax of a `switch` statement is:

switch and case are reserved words

```
switch ( expression )
{
    case value1 :
       statement-list1
    case value2 :
       statement-list2
    case value3 :
       statement-list3
    case  ...

}
```

If *expression* matches *value2*, control jumps to here

Comparing Characters

- In Unicode, the digit characters (0-9) are contiguous and in order
- Likewise, the uppercase letters (A-Z) and lowercase letters (a-z) are contiguous and in order

Characters	Unicode Values
0 – 9	48 through 57
A – Z	65 through 90
a – z	97 through 122

Comparing Strings

- Remember that in Java a character string is an object
- The `equals` method can be called with strings to determine if two strings contain exactly the same characters in the same order
- The `equals` method returns a boolean result

```
if (name1.equals(name2))
   System.out.println ("Same name");
```

Comparing Strings

- We cannot use the relational operators to compare strings
- The `String` class contains a method called `compareTo` to determine if one string comes before another
- A call to `name1.compareTo(name2)`
  - returns zero if `name1` and `name2` are equal (contain the same characters)
  - returns a negative value if `name1` is less than `name2`
  - returns a positive value if `name1` is greater than `name2`

Comparing Strings

```
if (name1.compareTo(name2) < 0)
   System.out.println (name1 + "comes first");
else
   if (name1.compareTo(name2) == 0)
      System.out.println ("Same name");
   else
      System.out.println (name2 + "comes first");
```

- **Because comparing characters and strings is based on a character set, it is called a *lexicographic ordering***

THANK YOU