# I. Decision Based Control Structures BJ Ch 3

week 5

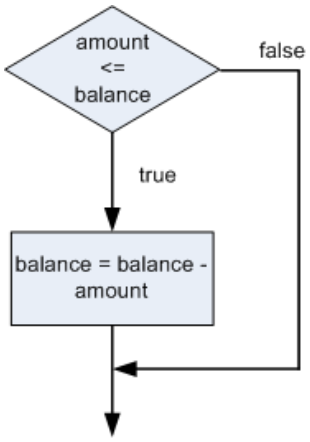
- **flow of control** refers to the order in which a programs statements are executed
- until now, all statements have executed sequentially
- decision structures allow the programmer to alter the normal sequential flow of control
- decision statements provide the ability to decide which statements, from a well-defined set, will get executed next

**simple if format:**

```
if (condition)              // note parens mandatory
    statement ;             // single-line body

if (condition)
{
    statement;              // multi-line body
    statement(s);           // must use braces
}
```

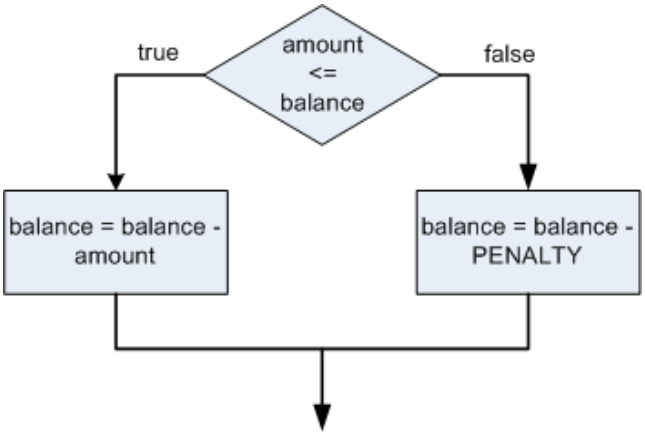
where condition has type of ? and values of ?



**if-else format:**

```
if (condition)
    statement1;                 // single-line body
else
     statement2;


if (condition)
{
    statement1;                 // multi-line body
    statement(s);               // must use braces
}
else
{
    statement2;                 // multi-line body
    statement(s);
}
```




- if >1 statement, must use { }…strongly suggest always using { }
- suggest lining up braces vertically
- indent inside { }
- note that **no statement terminator** follows the if or the else line
- empty blocks should NEVER be present – this is poor logical design
- note that compiler ignores all indentation, it is merely for the human reader

**nested if statements:**

```
if (condition1)
    if (condition2)
        statement;                    // single-line body
```

- the nested if-else (below left) will not work as planned, since the else will really match up with the closest if, no matter the indentation

```
if (condition1)          // wrong
    if (condition2)
        statement;
else
        statement;
```

```
if (condition1)          // right
{
    if (condition2)
        statement;
}
else
     statement;
```

## II. Relational Operators

- compare relationship of operands on either side of operator
- results are always ? type

| Name | Symbol | # Operands |
|---|---|---|
| 1. Equals | == | 2 |
| 2. Not Equals | != | 2 |
| 3. Less Than | < | 2 |
| 4. Less Than or Equal To | <= | 2 |
| 5. Greater Than | > | 2 |
| 6. Greater Than or Equal | >= | 2 |

- examples:

| **Expression** | **Result** |
|---|---|
| 2 < 9 | true |
| 8 < 8 | false |
| 6 <= 6 | true |
| 3 == 5 – 2 | true |
| 1.0 / 3.0 == 0.33333 | false ** |
| 3=< 4 | err |
| “a” == 9 | err – incompatible types |

- never compare strings using the == operator, use the methods that belong to the String class

- when comparing floating point numbers for equality**, take the absolute value of the difference of the two numbers, then compare to some small tolerance value

```
if (x == y)                 // wrong

if (Math.abs(x - y) < 0.0000000002)          // much better
```

## III. Logical Operators

- operators which work based on logic, return type of ?
- and operator → &&  Op Precedence # 2
- or operator → ||  Op Precedence # 3
- not operator → ! (note this is a unary operator)  Op Precedence # 1
- with an &&, if any operand is false, the entire statement is false
- with an ||, if any operand is true, the entire statement is true
- examples:

| | |
|---|---|
| 0 < 200 && 200 < 100 | false |
| 0 < 200 \|\| 200 < 100 | true |
| 0 < 200 \|\| 100 < 200 | true |
| ! ( 0 < 200 ) | false |
| 0 < 100 < 200 | error |

- rather than using (done == true), one can just use  if (done)
- rather than using (done == false), one can just use  if (!done)
- note the following is an error  if (a < 10) && (b < 20)