

## Decisions

### I. Relational Operators, Boolean Operators, Precedence of Operators

#### A) Relational Operators:

<u>Operation</u>	<u>Algebra</u>	<u>Java</u>
equal to	=	==
not equal to	≠	!=
greater than	>	>
less than	<	<
less than or equal to	≤	<=
greater than or equal to	≥	>=

Note 1: You can not have a space between !=, ==, >=, or <=

Note 2: You can not have =!, =>, or =<

ex.1a)  $5 < 6$  evaluates to true

ex.1b)  $2 > 3$  evaluates to false

\*Note: The above true and false are values that are not Strings, numbers, nor objects. They are values from a data type called **boolean**.

Note: Characters are compared by alphabetical order:

ex.2) 'b' < 'c' evaluates to true

ex.3) 'R' < 'W' evaluates to true

**but**

ex.4) 'a' < 'N' FALSE Why?

Answer: Because all uppercase letters come before lower case letters.

Note: You should only compare items of the same type:

'3' < '5' evaluates to true

3 < 5 evaluates to true

**but**

'3' < 5 evaluates to false

## II. B) Boolean Operators:

<u>Mathematics</u>	<u>Java</u>
AND	&& (binary operator)
OR	(binary operator)
NOT	! (unary operator)

Truth Tables: (page 212)

A	B	A && B
true	true	true
true	false	false
false	true	false
false	false	false

A	B	A    B
true	true	true
false	true	true
true	false	true
false	false	false

A	! A
true	false
false	true

Note: Java uses *short-circuit* (lazy) evaluation. That means in an or evaluation if the first part is true the evaluation stops and the result is true; likewise with an and evaluation with false as the first part the evaluation stops and the result is false.

Note: Java does have an & and | operator that causes complete evaluation.

### II. B) Boolean Operators continued

ex.1) `answer = (x > 7 && x == 9)` if a) `x = 0` b) `x = 9` c) `x = 10`

ex.2) `answer = (x > 7 || x == 9)` if a) `x = 0` b) `x = 9` c) `x = 10`

Note: What would happen if we took away the parenthesis in the following?  
`answer = x > 7 || x == 9`

or What would happen if we put parenthesis around each Boolean expression? `answer = (x > 7) || (x == 9)`

ex.3) In algebra we can logically write:  $2 < x$  and  $x < 12$  as  $2 < x < 12$   
but in Java can we logically write `2 < x && x < 12` as `2 < x < 12` ?

Answer: No, you would get an error message!

\*ex.4) convert from English to Java: `x = 3 or 4`

\*Answer: `x == 3 || x == 4`

## Computer Programming I - Unit 4 Lecture

C) Precedence of Operators: (Highest - done first -, to Lowest - done last)

<u>Operator</u>	<u>Associatively</u>
!	right to left
* / %	left to right
+ -	left to right
< <= > >=	left to right
= = !=	left to right
&&	left to right
	left to right
= *= /= += -=	right to left

ex.) evaluate:

**boolean** x;

x = 6 \* 3 == 36 / 2 || 13 < 3 \* 3 + 4 && !(6 - 2 < 5)

x = 6 \* 3 == 36 / 2 || 13 < 3 \* 3 + 4 && !(4 < 5)

x = 6 \* 3 == 36 / 2 || 13 < 3 \* 3 + 4 && !true

x = 6 \* 3 == 36 / 2 || 13 < 3 \* 3 + 4 && false

x = 18 == 18 || 13 < 13 && false

x = true || false && false

x = true || false

x = true

## III. *if* Statement - selection (decision) control structure

### A) *if* statement:

1) Syntax: **if** (logical, boolean expression)  
statement;

or

```
if (logical, boolean expression)
{
    statement1;
    statement2;
    etc.;
}
```

### 2) Semantic explanation

If *true* do the following statement or block of statements and continue at the statement following the semicolon or brace.

If *false* skip the following statement or block of statements and go right to the statement following the semicolon or brace.

ex.1) **if** (x >= 90)  
System.out.println("A");  
System.out.println("continue");

- a) x = 90
- b) x = 80

ex.2) **if** (x == 4)  
{  
System.out.println("Bob");  
System.out.println("Carol");  
System.out.println("Ted");  
System.out.println("Alice");  
}  
System.out.println("continue");

- a) x = 4
- b) x = 2

## III. *if* Statement - continued

### B) **if-else** statement:

1) Syntax: **if** (logical, Boolean expression)  
    statement1;  
    **else**  
        statement2;

or

```
if (logical, Boolean expression)
{
    statement1a;
    statement1b;
    etc.;
}
else
{
    statement2a;
    statement2b;
    etc.;
}
```

### 2) Semantic explanation

If TRUE do the following statement or block of statements, skip the statement or block of statements following the **else** and continue at the statement following the semicolon after the **else** or continue at the statement following the brace coming after the **else**.

If FALSE skip the following statement or block of statements and go right to the statement or block of statements following the **else** and then continue at the statement following the semicolon after the **else** or continue at the statement following the brace coming after the **else**.

ex.1) **if** (x >= 90)  
    System.out.println("A");  
    **else**  
        System.out.println("Not an A");  
        System.out.println("Bye");

- a) x = 90
- b) x = 80

### III. B) **if-else** Statement - continued

```
ex.2) if (x == 4)
    {
        System.out.println("Bob");
        System.out.println("Carol");
        System.out.println("Ted");
        System.out.println("Alice");
    }
else
    {
        System.out.print("The ");
        System.out.print("else ");
        System.out.print("block");
    }
System.out.println("Bye");
```

- a) with x = 4
- b) with x = 2

### C) **if** Statement Syntax Errors

```
ex.1) if (x = 100)
    System.out.print("Good ");
else
    System.out.println("OK ");
    System.out.println("Bye");
```

Error message    Why?

Answer: == needed

```
ex.2) if (x == 'a')
    System.out.println("Hi");
else ;
    System.out.println("Hello");
    System.out.println("Bye");
```

### III. C) **if** statement logical and syntax errors continued

```
ex.3) if (x == 4)
    {
        System.out.println("Bob");
        System.out.println("Carol");
        System.out.println("Ted");
        System.out.println("Alice");
    } ;
else
    {
        System.out.print("The ");
        System.out.print("else ");
        System.out.print("block");
    }
System.out.println("Bye");
```

```
ex.4a) if (x == 100);
        System.out.print("Good ");
    else
        System.out.print("OK ");
        System.out.print("Bye");
```

```
ex.4b) if (x == 100);
        System.out.println("Hi");
        System.out.println("Bye");
```

Note: Watch out, especially Pascal programmers, for:

- 1) the parenthesis around the logical expression after the **if**
- 2) the semicolon immediately before the **else** is OK in ex.1) but not in ex.3)
- 3) use == not = when making an equality comparison
- 4a) **if** (x == 1 || 2 || 3) *etc.* is an error Why?  
use: **if** (x == 1 || x == 2 || x == 3)
- 4b) **if** (x != 1          x != 2          x != 3)

### De Morgan's Law:

!(a && b) is the same as !a || !b

!(a || b) is the same as !a && !b

### III. *if* Statement - continued

#### D) Nested *if* statements - an *if* statement inside of another one

```
ex.1) if (x < 9)
    if (x > 6)
        System.out.println("snap");
    else
        System.out.println("crackle");
System.out.println("pop");
```

Test with:

- a) x = 8
- b) x = 5
- c) x = 10

```
ex.2) if (x < 9)
    {
        if (x > 6)
            System.out.println("snap");
    }
else
    System.out.println("crackle");
System.out.println("pop");
```

test with:

- a) x = 8
- b) x = 5
- c) x = 10

### III. D) Nested **if** statements continued

ex.3) If the code fragment

```
if (a == 5)
if (a != 5)
  a = a + 2;
else
  a = a + 1;
```

is indented according to the manner in which it is executed, the correct indentation would be

```
if (a == 5)
  if (a != 5)
    a = a + 2;
  else
    a = a + 1;
```

NOT

```
if (a == 5)
  if (a != 5)
    a = a + 2;
else
  a = a + 1;
```

UNLESS we do:

```
if (a == 5)
{
  if (a != 5)
    a = a + 2;
}
else
  a = a + 1;
```

## III. Nested **if** statements continued

### E) **if else** chain:

ex.1)	<u>Place</u>	<u>Points</u>
	First	> 1500 pts
	Second	1000 < pts ≤ 1500
	Third	750 < pts ≤ 1000
	Honorable Mention	≤ 750 pts

### OK algorithm:

```
If pts > 1500 then
    First
If pts ≤ 1500 and pts > 1000 then
    Second
If pts ≤ 1000 and pts > 750 then
    Third
If pts ≤ 750 then
    Honorable Mention
```

### Better algorithm:

```
If pts > 1500 then
    First
Else
    If pts > 1000 then
        Second
    Else
        If pts > 750 then
            Third
        Else
            Honorable Mention
```

### Java code:

```
if (pts > 1500)
    System.out.println("First");
else
    if (pts > 1000)
        System.out.println("Second");
    else
        if (pts > 750)
            System.out.println("Third");
        else
            System.out.println("Honorable Mention");
```

## IV. Comparing Strings

### A) Case insensitive test ("Y" or "y")

```
if (input.equalsIgnoreCase("Y"))
```

s.compareTo(t) < 0 means: s comes before t in the dictionary

Note: "car" comes before "cargo"

Note: All uppercase letters come before lowercase:

ex.1) "Hello" comes before "car"

### B) == vs. equals( )

Don't use == for strings!

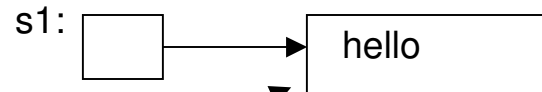
```
if (input == "Y") // WRONG!!!
```

Use equals method:

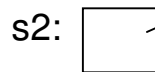
```
if (input.equals("Y"))
```

== tests identity, equals tests equal contents

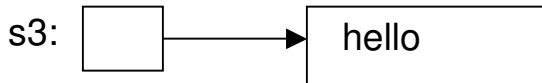
ex.1) String s1 = "hello";



String s2 = s1;



String s3 = **new** String("hello");



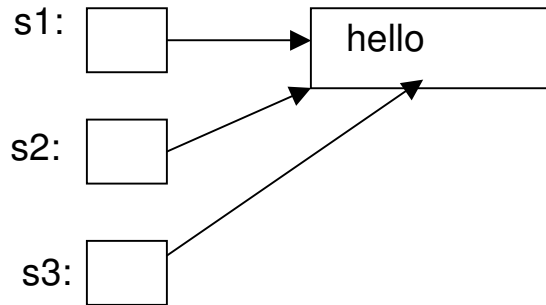
<u>Expression</u>	<u>Result</u>
a) s1 == s2	true
b) s1.equals(s2)	true
c) s1 == s3	false
d) s1.equals(s3)	true
e) s2 == s3	false
f) s2.equals(s3)	true

## IV. Comparing Strings continued

ex.2) String s1 = "hello";

String s2 = s1;

String s3 = "hello";



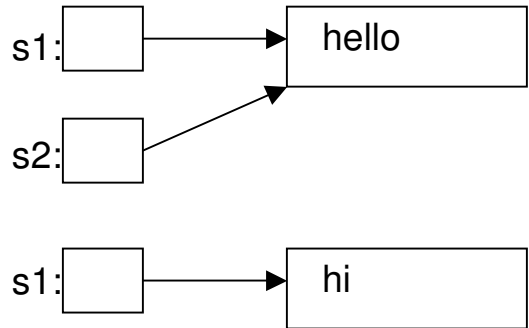
<u>Expression</u>	<u>Result</u>
a) s1 == s2	true
b) s1.equals(s2)	true
c) s1 == s3	true
d) s1.equals(s3)	true
e) s2 == s3	true
f) s2.equals(s3)	true

## IV. Comparing Strings continued

ex.3)

```
String s1 = "hello";  
String s2 = s1;  
s1 = new String("hi");
```

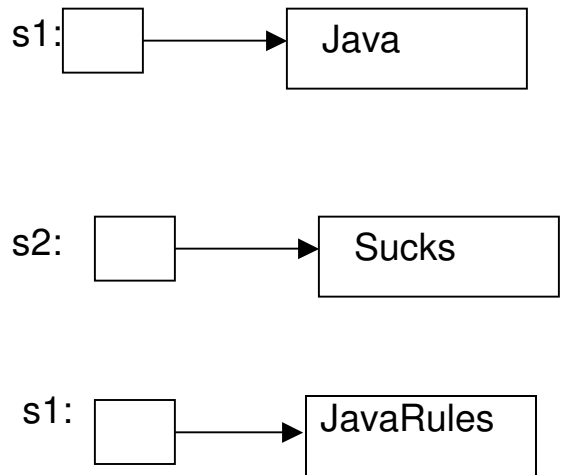
<u>Expression</u>	<u>Result</u>
a) s1 == s2	false
b) s1.equals(s2)	false



ex.4)

```
String s1 = "Java";  
String s2 = "Sucks";  
s2 = s1;  
s1 += "Rules";  
System.out.println ("s1: " + s1);  
System.out.println ("s2: " + s2);
```

Output:  
s1: JavaRules  
s2: Java



\*This last example shows why Strings are called *immutable* objects i.e. they can not be changed to a new string, instead a new object (string) is automatically created.

### IV. Comparing Objects

recall: == tests for identity, equals( ) for identical content

ex.1)

```
Rectangle box1 = new Rectangle(5, 10, 20, 30);
```

```
Rectangle box2 = box1;
```

```
Rectangle box3 = new Rectangle(5, 10, 20, 30);
```

box1 == box2 is true also

box1.equals(box2) is true

box1 == box3 is false but

box1.equals(box3) is true!

Note: The equals( ) method must be defined - i.e. code written for it - in the class. The String class has a built-in, already written, equals( ) method but other classes do not.

### V. Testing for null

Note: null reference refers to no object

```
String middleInitial = null; // Not set
```

```
if ( . . . )
```

```
    middleInitial = middleName.substring(0, 1);
```

```
    .
```

```
    .
```

```
    .
```

```
if (middleInitial == null)
```

```
    System.out.println(firstName + " " + lastName);
```

```
else
```

```
    System.out.println(firstName + " " + middleInitial + ". " + lastName);
```

Note: null is not the same as the empty string ""

\*Note: Use ==, not equals( ), to test for null

### VII. Using Boolean Variables

ex.1)

```
private boolean married;  
married = input.equals("M");  
if (married) . . .
```

or . . .

```
if (!married) . . .
```

It is considered gauche (bad form - laughed at by real programmers) to write a test such as:

```
if (married == true)
```

Just use the simpler test:: **if** (married) . . .

Note: This is also called *flag*