Boolean Primitive Type

❖ Like arithmetic primitive data types, except the value can be either
`true` or **`false`**

❖ Syntax

Declaration: **`boolean <variable name>;`**

Assignment: **`<variable name> = <expression>;`**

`boolean` is a Java reserved word

`<expression>` is any expression that evaluates to true or false

Examples:

```
boolean hasAllBottles;
hasAllBottles = true;
```

```
boolean isZero;
isZero = ( x == o );
```

```
boolean notNegative = ( x >= 0);
```

Boolean Expressions and Variables

❖ Relational Operators (ALL are BINARY operators)

<	less than
<=	less than or equal to
==	equal to
!=	not equal
>	greater than
>=	greater than or equal to

Examples:

```
int x = 5;
int y = 5;
int z = 6;
boolean answer;

answer = x < y;
answer = x <= y;
answer = x == y;
answer = x != z;
answer = x > z;
answer = z >= y;
```

❖ Boolean Operators (ALL are BINARY operators EXCEPT !)
(aka "Logical Operators")

&&	AND	
||	OR	
!	NOT	(a UNARY operator)

❖ **true false** are Java reserved words

❖ Can string many boolean expressions together

e.g. **(true && true || true && false)**

Boolean Expressions and Variables...continued

❖ Short Circuit Evaluation of Boolean Expressions:

- For the OR operator **||**, if the left operand is evaluated to **true**, then the right operand will not be evaluated (the result will be true regardless of the right operand)

```
( true || false )
```

- For the AND operator **&&**, if the left operand is evaluated to **false**, then the right operand will not be evaluated (the result will be false regardless of the right operand)

```
( false && true )
```

- What would happen if the short-circuit evaluation is not done for the following expression?

```
z == 0 || x / z > 20
```

❖ Ranges

In Math, ranges are expressed by: **(0 < x <= 100)**

In Java, ranges are are expressed by: **(0 < x && x <= 100)**

❖ Arithmetic Expressions can be intermixed with boolean expressions

e.g. **(x + y == y + x)**

❖ Boolean Expressions as "flags"

e.g.

```
boolean secondsToZero = false;
if (seconds == 0) {
    secondsToZero = true;
}
```

Using Logical Operators

More accurate searches on the web are done using logical operators. For each search request below, figure our which documents (A-H) would be found.

A passing a camel through the eye of a needle
B passing a camel through the hand of a goalie
C passing a ball into the eye of a goalie
D passing a ball into the hand of a needle
E kicking a camel through the eye of a goalie
F kicking a camel through the hand of a needle
G kicking a ball into the eye of a needle
H kicking a ball into the hand of a goalie

---

1 eye && ball
2 passing || kicking
3 !hand
4 passing && camel && eye && needle
5 kicking || ball || hand || goalie
6 !(!(!(!camel)))
7 kicking && (eye || ball)
8 kicking && ball || hand && goalie
9 !needle && !passing
10 !(camel || goalie)
11 camel && ball || eye && hand || passing && goalie
12 kicking && !needle || !(camel || !eye)

If Statements

❖ Syntax:

```
if (condition) {
    < then block >
}
else {
    < else block >
}
```

if & **else** are Java reserved words

(condition): some boolean expression (i.e., an expression that evaluates to either true or false)

< then block >: 0, 1, or more Java statements
< else block >: 0, 1, or more Java statements

The **else** and **< else block >** are optional!

```
if (condition) {
    < then block >
}
```

Using curly braces {} is optional if only one statement is contained within the block. However, one should ALWAYS use curly braces regardless of the number of statements contained within the block.

To "do nothing":

```
if (condition) {
    /* DO NOTHING */
}
else {
    // DO NOTHING
}
```

If Statements...continued

❖ Nested if...else

```
if (condition) {
    if (condition) {
        < then block >
    }
    else {
        if (condition) {
            < then block >
        }
        < other else block statements >
    }
else {
    < then block >
}
```

NOTE:

- For every **`else`**, there must be a corresponding **`if`**
- Every **`else`** is matched to the closest previously unmatched **`if`** at the same level of nesting.

❖ Indenting "else if"

Instead of:

```
if ( condition )
else
    if (condition)
    else
        if (condition)
        else
            if (condition)
            else
```

Use:

```
if ( condition )
else if (condition)
else if (condition)
else if (condition)
else
```

```
import javabook.*;

class NestedIfElse {
```

```
    public static void main ( String args [] ) {

        MainWindow mw = new MainWindow ();
        InputBox inBox = new InputBox (mw);
        OutputBox outBox = new OutputBox (mw);
        mw.show ();
        outBox.show ();

        int temp = inBox.getInteger (“Enter the
        temperature in Fahrenheit”);

        if (temp > 32) {
            if (temp > 212) {
                outBox.printLine (“It’s Boiling!”);
            }
        }
        else {
            outBox.printLine (“It’s Freezing”);
        }

    }

}
```

Using if...else

Given the following:

```
int year;       // an integer > 0
int month;      // an integer between 1 and 12
int day;        // an integer between 1 and 31
```

Write a code fragment for each of the following:

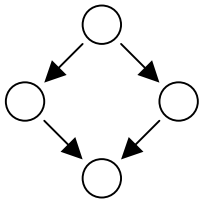
1. Determine if it is Halloween (October 31)
2. Determine if it is a summer month (June, July, August, or September)

Write a method named daysInMonth which is passed an integer value representing the month (i.e., 1-12) and returns the number of days in that month. You may assume that February has 28 days.

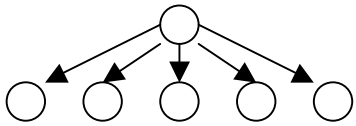
Write a method named rightTriangle that determines whether a Triangle represents a right triangle. Have this method use the Pythagorean Theorem ($a^2 + b^2 = c^2$ where a and b are sides and c is hypotenuse). The method should be an instance method of the Triangle class. Triangle objects have three data members, **`side1`**, **`side2`**, and **`side3`**, all of data type double. The method should return a boolean value of **`true`** or **`false`**.

Switch Statements

if...else



switch



Syntax:

```
switch ( < expression > ) {
    case < expression value >:    < case body >
    case < expression value >:    < case body >
    case < expression value >:    < case body >
    default:                      < default body >
}
```

switch, **case**, **default**, and **break** are all Java reserved words

<expression>: must evaluate to a **byte**, **short**, **int** or **char** (**char** is a primitive data type referring to one character (e.g., **char letter = 'a';**))

< expression value >: must match the type of the evaluation of the **< expression >**

< case body >: 0, 1 or more Java statements
curly braces { } are not used

< default body >: 0, 1 or more Java statements (error messages)
curly braces { } are not used
optional: executed only if there is no matching case

Switch Statements...continued

- Can have ANY number of cases, but can only have 0 or 1 default

- Each case is skipped until a case matches. Then ALL remaining cases, including the case "jumped to" are executed.—Fall Through

- To prevent Fall Through, use the Java reserved word '**break**'.
  When a break is encountered in a case body, the flow of control is immediately moved to the end of the switch statement

Example:

Menu

A) Do Homework
B) Eat A Snack
C) Go To Bed

```
char choice = Class.getChoice ();
switch (choice) {
    case 'A':
    case 'a':  doHomework ();
               break;

    case 'B':
    case 'b':  eatASnack ();
               break;

    case 'C':
    case 'c':  goToBed ();
               break;

    default:   outBox.printLine ("Wrong Choice!");
               System.exit (1);
}
```

class ListBox

- Defined as an instantiable class in the javabook package
- Objects of the class provide a list of items that the user can select which gives control of input to programmer, not user. This is considered a better interface.
- Requires an owner frame window (like a MainWindow object)
- Can be used effectively with switch statements
- Example Code:

```
MainWindow mw = new MainWindow ();
ListBox list = new ListBox (mw);

list.addItem ("Do Homework");
list.addItem ("Eat A Snack");
list.addItem ("Go To Bed");

int choice = list.getSelectedIndex ();

switch (choice) {

  case CANCEL:
  case NO_SELECTION:    /* DO NOTHING */ break;

  case 0:               doHomework ();
                        break;

  case 1:               eatASnack ();
                        break;

  case 2:               goToBed ();
                        break;

  default:              /* ERROR if default reached */
                        System.exit (1);
}
```

Additional Operators

❖ Increment **++** /Decrement **--**

Adds or Subtracts 1 from a variable

Unary operators

Syntax: PREFIX: **++< variable >**
increments variable FIRST

POSTFIX: **< variable >++**
variable retains value (for the expression) and then is incremented

NOTES: Once incremented, the variable retains the new value
Rather than use **x = x + 1;** use **x++;**

Example:

```
int x = 0;
outBox.printLine (x++);
outBox.printLine (++x);
outBox.printLine (x);
```

OUTPUT

Other "increment/decrement" operators (these are binary):

Operator	Example Usage	Equivalent To:
+=	x += y;	x = x + y;
-+	x -= y;	x = x – y;
*=	x *= y;	x = x * y;
/=	x /= y;	x = x / y;
%=	x %= y;	x = x % y;