Lecture 11

- Covers
  - Branching statements
  - The Java if…else statement
  - Boolean expressions
- Reading: Savitch 3.1

Lecture overview

1. The if Statement
2. The if…else Statement
3. Relational Operators
4. Logical Operators
5. String Comparison
6. Bitwise Operators (Optional)

- The if statement

if statement

- The Java if statement allows us to conditionally execute a statement
- The condition in brackets is evaluated and if it is true, the next statement is executed
- Note there is no semicolon after the condition as that is not the end of the if statement

```
if (<condition>)
    <statement>;
```

if statement

- We can use braces {} to group statements together and conditionally execute a group of them
- By using braces, we create a compound statement

```
if (<condition>)
{
    <statements>
}
```

if statement

- Even when we only have one statement to execute in the if statement, it is good style to enclose it in braces

```
if (time < 1400)
{
        System.out.println("Time to go to lunch");
}
```

Example

```
if (myScore > yourScore)
{
    System.out.println("I win! ");
    winnings = winnings + bet;
}
```

Example

- Problem
  - Display the maximum of two numbers entered by the user
- Algorithm

```
Get the two numbers (n1 and n2)
IF n1 >= n2 THEN
        display n1
ENDIF
IF n2 > n1 THEN
        display n2
ENDIF
```

Java Solution

```
// get n1 and n2

if ( n1 >= n2 )
{
    System.out.println("max = " + n1 );
}

if ( n2 > n1 )
{
    System.out.println("max = " + n2 );
}
```

Example

- Problem
  - Read two numbers, then display the smaller number followed by the larger

Solution

- The if…else statement

if…else statement

- We can extend the if statement to execute a different statement or set of statements if the condition is false

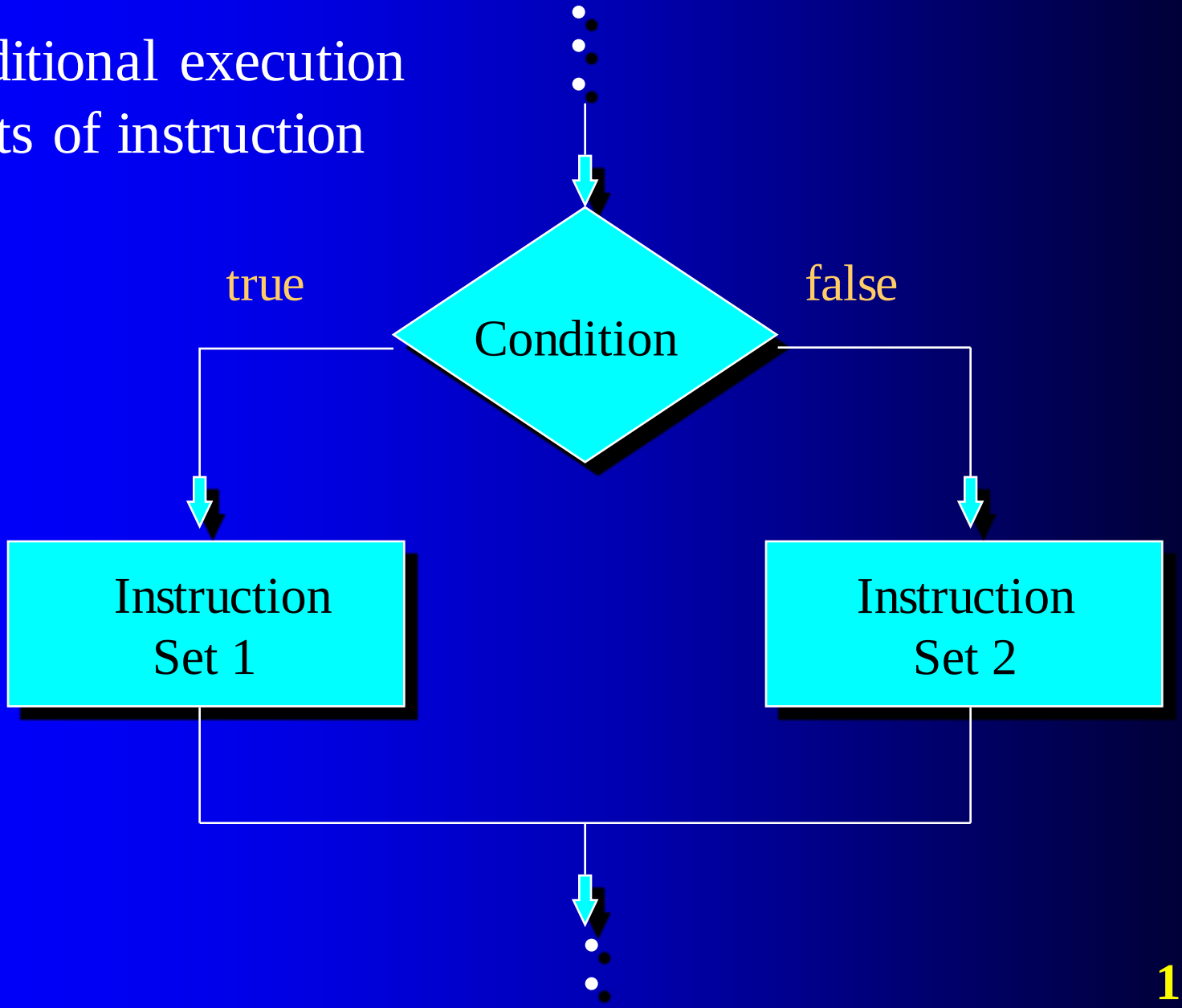
```
if (<condition>)
{
    <statements>
}
else
{
    <statements>
}
```

this group of statements is executed if the condition is true

this group of statements is executed if the condition is false

Selection

Conditional execution of sets of instruction



Example

- Revisit the last problem (using if…else)
  - Read two numbers, then display the smaller number followed by the larger
- Solution

```
if ( n1 <= n2 )
{
        System.out.println( n1 );
        System.out.println( n2 );
}
else
{
        System.out.println( n2 );
        System.out.println( n1 );
}
```

Question

- What are the advantages of using an if...else statement as opposed to using two if statements?

Is this code correct?

```
if (mark >= 50)
    System.out.println("Congratulations, you passed.");
    totalPassed++;
else
    System.out.println("Sorry, you didn't pass.");
    totalFailed++;
```

- Problems?

Compound statement blocks

- Solution

```
if (mark >= 50)
{
    System.out.println("Congratulations, you passed.");
    totalPassed++;
}
else
{
    System.out.println("Sorry, you didn't pass.");
    totalFailed++;
}
```

if…else statement

- The condition of an if or if…else statement must be a boolean expression
- That is, it must evaluate to a boolean value

- Relational operators

Relational operators

- Relational operators allow us to compare two values of primitive type
- The value of a logical expression is either true or false

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

Pitfalls

```
if (x = 10)
{
    // Requires boolean expression
}

if (8 < x < 12)
{
    // Tries to compare the
    // boolean result with 12
}
```

Pitfalls

- What happens if we want to check that the value of some variable falls between two specified values?
- E.g. Check x is greater than 1 but less than 10

  1 < x < 10 is not allowable in Java

  (1 < x) && (x < 10) is the solution

- Logical operators

Logical connectives

- We can combine boolean values with the and, or and not symbols

&&	and
||	or
!	not

Be careful when using ! in a boolean expression; a rewritten expression without ! may be easier to understand

Logical operators

- Relational operators can have arithmetic expressions as operands
- Logical operators have logical expressions as operands
- Precedence (from highest to lowest)

  !

  &&

  ||

Syntax revisited

```
if ( <condition>)
    <if-block>
else
    <else-block>
```

- The if-block and else-block contain any kinds of statements
- When they contain other if or if…else statements, we get a structure often referred to as nested if

Class exercise

- Write a Java code segment that outputs to screen the message "Phone home" if the value of the boolean variable isET is true. Otherwise, the message "Sorry, wrong number" is displayed.

Solution

- (More on) String Comparison

== with Strings

- == with Strings (as with any other objects) does not evaluate whether the content of two Strings is the same
- It evaluates whether the two strings are stored at the same memory location

Equality with Strings

- To test if two strings are equal, use methods equals or equalsIgnoreCase

  E.g.

```
String s1 = "abc";
String s2 = "ABC";
s1.equals(s2)                 // false
s1.equalsIgnoreCase(s2)       // true
```

Class exercise

Given these four declarations:

```
String family1 = "Who";
String family2 = "who";
String name = "Dr Who";
String title = "Dr";
```

Evaluate the expressions below:

```
family1.equals(family2)
family2.equalsIgnoreCase(family1)
name.equalsIgnoreCase(title + family2)
name.equals(title + " " + family1)
```

Ordering of Strings

- To test the lexical order of two strings, use methods compareTo or compareToIgnoreCase

  E.g.

```
String s1 = "abc";
String s2 = "abc";
String s3 = "abe";
s1.compareTo(s2)        // returns 0
s1.compareTo(s3)        // returns -2
```

Class exercise

- Write a Java code segment that outputs to screen the contents of two String objects, author1 and author2, in lexical order, one per line.

- Bitwise operators

Bitwise operators

- Used for low-level bit manipulation of data (which are represented as integers)
- Examples are

&	bitwise AND
|	bitwise OR
<<	left shift
>>	right shift

e.g. 0xF0F0 & 0x0F0F gives 0x0000 (i.e. 0)

- Programming with bitwise operations is not required in this course

Examples

- Example 1 - consider an integer variable n

```
int rightMostBit = n & 1;
int thirdBitFromRight = (n & (1 << 2)) >> 2;
```

- Example 2

```
final int clearanceLevelA = 1;
final int clearanceLevelB = 2;
final int clearanceLevelC = 4;

int agent007Clearance = 0;
// set clearance level flag at A and C for agent007
agent007Clearance = agent007Clearance | clearanceLevelA;
agent007Clearance = agent007Clearance | clearanceLevelC;
```

* Remember, programming with bitwise operations is *not* required in this course

Examples

- For example, to test for clearance at level C say:

```
if ( (agent007Clearance & clearanceLevelC) == clearanceLevelC )
{
    …
}
```

Next lecture

- Branching statements
  - Nested if…else statements
  - Multiway branching statements