COMP 202

Conditional Programming

CONTENTS:

- The IF statement
- The SWITCH statement

Flow of Control

- **Default Flow**: the order of statement execution through a method is linear: one after the other in the order they are written (top of page, downwards to end of page)
- Some programming statements modify that order, allowing us to:
  - decide whether or not to execute a particular statement, or
  - perform a statement over and over repetitively (while)
- The order of statement execution is called the *flow of control*

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
- Java's conditional statements are the *if statement*, the *if-else statement*, and the *switch statement*

Part 1

The IF Statement

The if Statement

- The *if statement* has the following syntax:

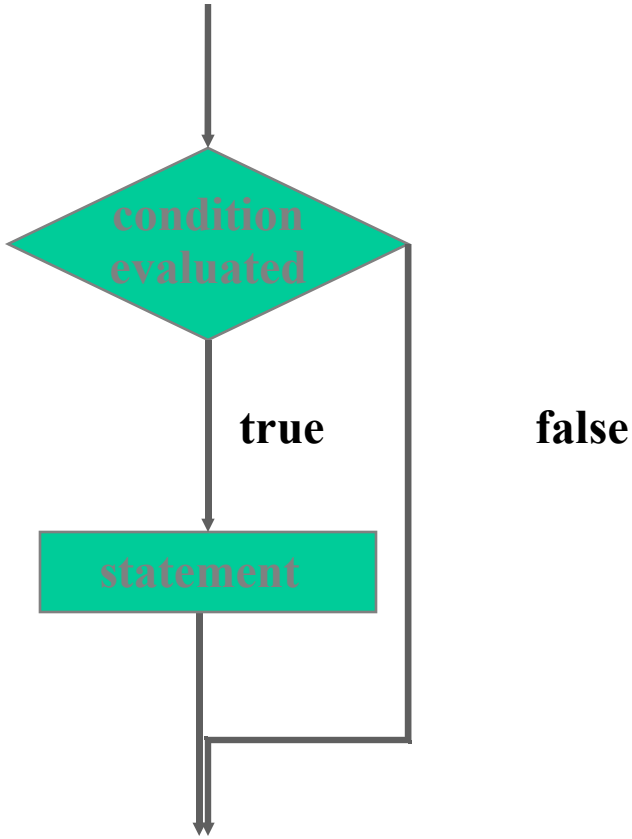
The condition must be a *boolean expression*.
It must evaluate to either true or false.

`if` is a Java reserved word

```
if ( condition )
    statement;
```

If the condition is true, the statement is executed.
If it is false, the statement is skipped.

Logic of an if statement



Boolean Expressions

- A condition often uses one of Java's *equality operators* or *relational operators*, which all return boolean results:

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

- Note the difference between the equality operator (==) and the assignment operator (=)

The if-else Statement

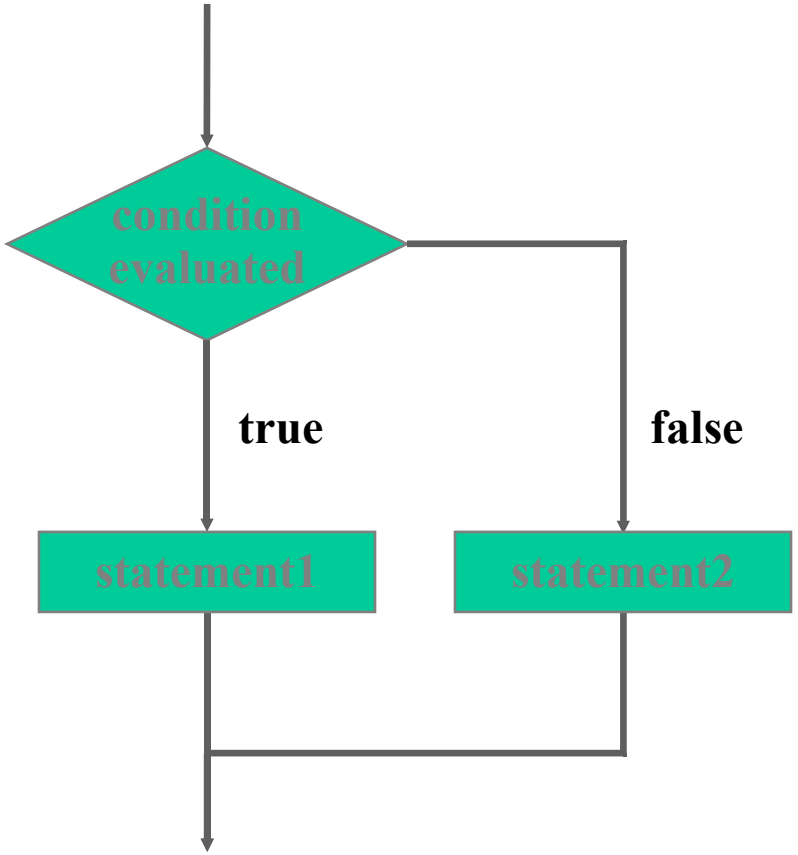
- An *else clause* can be added to an if statement to make it an *if-else statement*:

```
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



Block Statements

- Several statements can be grouped together into a *block statement*
- A block is delimited by braces ({ . . . })
- A block statement can be used wherever a statement is called for in the Java syntax
- For example, in an if-else statement, the if portion, or the else portion, or both, could be block statements

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*
- Indentation does not determine which IF and ELSE matches with. It is determined by syntax (ie. Order or {})
- Note: DrJava might not perform proper indentation for nested statements
  - solution: use {}

MinOfThree.java

```
int num1, num2, num3, min = 0;
Scanner scan = new Scanner(System.in);

System.out.println ("Enter three integers: ");
num1 = scan.nextInt();
num2 = scan.nextInt();
num3 = scan.nextInt();

if (num1 < num2)
   if (num1 < num3)
      min = num1;
   else
      min = num3;
else
   if (num2 < num3)
      min = num2;
   else
      min = num3;

System.out.println ("Minimum value: " + min);
```

MinOfThree.java

```
int num1, num2, num3, min = 0;
Scanner scan = new Scanner(System.in);

System.out.println ("Enter three integers: ");
num1 = scan.nextInt();
num2 = scan.nextInt();
num3 = scan.nextInt();

if (num1 < num2) {
   if (num1 < num3)
      min = num1;
   else
      min = num3;
} else {
   if (num2 < num3)
      min = num2;
   else
      min = num3;
}
System.out.println ("Minimum value: " + min);
```

More than two execution branches

- Nested statements are needed when there are not only two branches.
- An if-(else-if)-else statement allows several execution branches.

```
if ( condition )
    statement1;
else if (condition)
    statement2;
else
    statement3;
```

same as

```
if ( condition )
    statement1;
else
    if (condition)
        statement2;
    else
        statement3;
```

Comparing Characters

- We can use the logical operators on character data
- The results are based on the Unicode character set
- The following condition is true because the character '+' comes before the character 'J' in Unicode:

```
if ('+' < 'J')
   System.out.println ("+ is less than J");
```

- The uppercase alphabet (A-Z) and the lowercase alphabet (a-z) both appear in alphabetical order in Unicode

Comparing Strings

- Remember that a character string in Java is an object
- We cannot use the logical operators to compare objects
- The `equals` method can be called on a String to determine if two strings contain exactly the same characters in the same order (even constants)
- The String class also contains a method called `compareTo` to determine if one string comes before another alphabetically (as determined by the Unicode character set)

Comparing Floating Point Values

- We also have to be careful when comparing two floating point values (`float` or `double`) for equality
- You should rarely use the equality operator (==) when comparing two floats
- In many situations, you might consider two floating point numbers to be "close enough" even if they aren't exactly equal
- Therefore, to determine the equality of two floats, you may want to use the following technique:

```
if (Math.abs (f1 - f2) < 0.00001)
   System.out.println ("Essentially equal.");
```

Try These Out

- Write a program called BuyStuff.java that asks the user for two amounts, adds them and calculates tax at 15%, shows this to user and asks for money. It then compares if the person gave enough money. If so, it displays the amount of change to return otherwise it displays a message asking for more money.

Part 2

The SWITCH Statement

The switch Statement

- The *switch statement* provides another means 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 the case associated with the first value that it matches with (first come first serve)

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

The switch Statement

- Often a *break statement* is used as the last statement in each case's statement list
- A `break` statement causes control to transfer to the end of the switch statement
- If a break statement is not used, the flow of control will continue into the next case
- Sometimes this can be helpful, but usually we only want to execute the statements associated with one case

Example

```
int age;
age = scan.nextInt();

switch(age)
{
  case 5:
      System.out.println("Five years old");
      break;
  case 10:
      age++;
  case 20:
      age--;
}
```

What happens when:

- AGE is 5, 10 or 20?
- AGE is 3, or any other number?

The switch Statement

- A switch statement can have an optional *default case*
- The default case has no associated value and simply uses the reserved word `default`
- If the default case is present, control will transfer to it if no other case value matches
- Though the default case can be positioned anywhere in the switch, it is usually placed at the end
- If there is no default case, and no other value matches, control falls through to the statement after the switch

Example

```
char grade;
String input = scan.next(); // Input A, B, C,  F
grade = input.charAt(0); // Input A, B, C,  F

switch(grade)
{
  case 'A':
  case 'B':
  case 'C':
       System.out.println("pass");
       break;
  case 'F':
       System.out.println("fail");
       break;
  default:
       System.out.println("Sorry, no other choices!");
}
```

The switch Statement

- The expression of a switch statement must result in an *integral data type*, like an integer or character; it cannot be a floating point value, nor a String
- Note that the implicit boolean condition in a switch statement is equality - it tries to match the expression with a value (it is never <, <=, >, nor >=)
- You cannot perform relational checks with a switch statement

Drinks.java


```
System.out.println ("Here is the drinks menu : ");
System.out.println ("1.\tOrange juice");
System.out.println ("2.\tMilk");
System.out.println ("3.\tWater");
System.out.println ("4.\tWine");
System.out.println ("5.\tBeer");
System.out.print ("What will it be ? ");
int choice = scan.nextInt();

switch (choice)
{
   case 1:
      System.out.println ("Vitamin C!");
   case 2:
      System.out.println ("Your bones will thank you.");
      break;
```

Drinks.java


```
        case 3:
            System.out.println ("The classics never die.");
            break;
        case 4:
            System.out.print ("Red or white ? ");
            String type = scan.next();
            boolean isRed = (type.toLowerCase()).equals("red");
            if (isRed)
                System.out.println ("Good for your heart.");
            else
                System.out.println ("Good for your lungs.");
            break;
        case 5:
            System.out.println ("Watch that belly!");
            break;
        default:
            System.out.println ("That's not going to quench your
thirst...");
    }
```

Part 3

About Logical Operators

Logical Operators

- Boolean expressions can also use the following *logical operators*:

!	Logical NOT
&&	Logical AND
||	Logical OR

- They all take boolean operands and produce boolean results
- Logical NOT is a unary operator (it has one operand), but logical AND and logical OR are binary operators (they each have two operands)

Examples

Unary

```
boolean choice = false;
if (!choice) System.out.println("Go");
else System.out.println("Stop");
```

Unary with expression

```
if (!(x>5)) …
```

Binary

```
if ( (x>5) && (y<10) )
    choice = true;
else
    choice = false;
```

Logical NOT

- The *logical NOT* operation is also called *logical negation* or *logical complement*
- If some boolean condition `a` is true, then `!a` is false; if `a` is false, then `!a` is true
- Logical expressions can be shown using *truth tables*

a	**!a**
true	**false**
false	**true**

Logical AND and Logical OR

- The *logical and* expression

```
a && b
```

is true if both `a` and `b` are true, and false otherwise

- The *logical or* expression

```
a || b
```

is true if a or b or both are true, and false otherwise

Truth Tables (revisited)

- A truth table shows the possible true/false combinations of the terms
- Since `&&` and `||` each have two operands, there are four possible combinations of true and false

`a`	`b`	`a && b`	`a || b`
true	**true**	**true**	**true**
true	**false**	**false**	**true**
false	**true**	**false**	**true**
false	**false**	**false**	**false**

Logical Operators

- Conditions in selection statements and loops can use logical operators to form complex expressions

```
if (total < MAX && !found)
   System.out.println ("Processing…");
```

- Logical operators have precedence relationships between themselves and other operators

Truth Tables

- Specific expressions can be evaluated using truth tables

`total < MAX`	`found`	`!found`	`total < MAX && !found`
false	false	true	false
false	true	false	false
true	false	true	true
true	true	false	false

Part 4

The ?: Operator

The Conditional Operator

- Java has a *conditional operator* that evaluates a boolean condition that determines which of two other expressions to evaluate
- The result of the chosen expression is the result of the entire conditional operator
- Its syntax is:

```
condition ? expression1 : expression2
```

- If the *condition* is true, *expression1* is evaluated; if it is false, *expression2* is evaluated

The Conditional Operator

- The conditional operator is similar to an if-else statement, except that it is an expression that returns a value
- For example:

```
larger = (num1 > num2) ? num1 : num2;
```

- If `num1` is greater that `num2`, then `num1` is assigned to `larger`; otherwise, `num2` is assigned to `larger`
- The conditional operator is *ternary*, meaning that it requires three operands

The Conditional Operator

- Another example:

```
System.out.println ("Your change is " +
  count +(count == 1) ? "Dime" : "Dimes");
```

- If `count` equals 1, then `"Dime"` is printed
- If `count` is anything other than 1, then `"Dime**s**"` is printed

Wages2.java

```
final double RATE = 8.25;  // regular pay rate
final int STANDARD = 40;   // standard hours in a work week
boolean isProf;            // is the worker a professor or not?
double pay = 0.0;
Scanner scan = new Scanner(System.in);

System.out.print ("Enter the number of hours worked: ");
int hours = scan.nextInt();
System.out.print ("Are you a professor (Y/N)? ");
String answer = scan.next();

if ( answer.equalsIgnoreCase("Y") ) {
   isProf = true;
   System.out.println("Sorry...Overtime does not apply to YOUR kind.");
}
else
   isProf = false;
pay = (hours > STANDARD && !isProf) ?
   STANDARD*RATE+(hours-STANDARD)*(RATE*1.5) : hours*RATE;

System.out.println ("Gross weekly earnings: " + pay);
```