03 – Decision Structures

ICSI 201
Introduction to Computer Science

Qi Wang
Department of Computer Science
University at Albany
State University of New York

Course Materials

- All course materials are **copyrighted** and can be used by the students who are enrolled in the class only.
- Posting/sharing the course materials for other uses without permission is prohibited.

Outline

- Conditional statements
  - Comparing Data
  - The if, if-else, and switch Statements

Flow of Control

- Unless specified otherwise, the order of statement execution through a method is linear: one statement after another in sequence.
- Some programming statements allow us to:
  - decide whether or not to execute a particular statement.
  - execute a statement over and over, repetitively.

Flow of Control

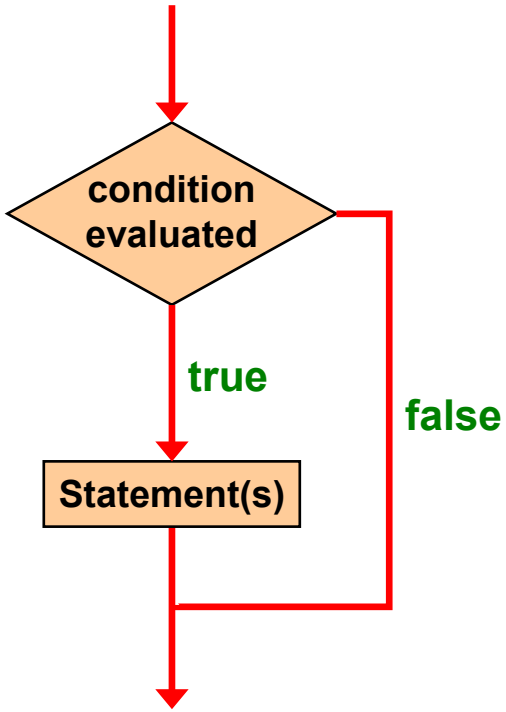
- These decisions are based on *boolean expressions* (or *conditions*) that evaluate to true or false.
- The order of statement execution is called the *flow of control.*

Conditional Statements

- A *conditional statement* lets us choose which statement will be executed next.
- Conditional statements give us the power to make basic decisions.
- The Java conditional statements are the:
  - *if statement*
  - *if-else statement*
  - *switch 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(s);
}
```

If the *condition* is true, the *statement(s)* are executed. If it is false, the *statement(s)* are skipped.

Block Statements

- Several statements can be grouped together into a *block statement* delimited by braces.
- A block statement can be used wherever a statement is called for in the Java syntax rules.

```
if (total > MAX){
   System.out.println ("Error!!");
   errorCount++;
}
```

Block Statements

- In an `if-else` statement, the `if` portion, or the `else` portion, or both, could be block statements.

```
if (total > MAX){
   System.out.println ("Error!!");
   errorCount++;
}else{
   System.out.println ("Total: " + total);
   current = total*2;
}
```

Comparing Data using Boolean Expressions

- When comparing data using boolean expressions, it's important to understand the nuances of certain data types.

- Let's examine some key situations:
  - Comparing integral values
  - Comparing floating point values
  - Comparing characters
  - Comparing strings
  - Comparing object vs. comparing object references

Comparing integral values

Comparing integral values

- 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 Statement

- Example:

```
if (sum > MAX)
   delta = sum - MAX;
System.out.println ("The sum is " + sum);
```

OR

```
if (sum > MAX){
   delta = sum - MAX;
}
System.out.println ("The sum is " + sum);
```

- First the condition is evaluated -- the value of sum is either greater than the value of MAX, or it is not.

The if Statement

- Example cont.:

```
   if (sum > MAX)
      delta = sum - MAX;
   System.out.println ("The sum is " + sum);
OR
   if (sum > MAX){
      delta = sum - MAX;
   }
   System.out.println ("The sum is " + sum);
```

- If the condition is true, the assignment statement is executed -- if it isn't, it is skipped.

- Either way, the call to println is executed next.

```java
package chapter03;

import java.util.Scanner;

/**
 * Checks for minor.
 * @author Lewis Loftus
 * @version 1.0
 */
public class Age{
    /**
     * Asks for an age, comments for a minor.
     * @param args A string array that can hold the command-line arguments
     */
   public static void main (String[] args){
      final int MINOR = 21;
      Scanner scan = new Scanner (System.in);

      System.out.print ("Enter your age: ");
      int age = scan.nextInt();

      System.out.println ("You entered: " + age);

      if (age < MINOR){
         System.out.println ("Youth is a wonderful thing. Enjoy.");
      }
      System.out.println ("Age is a state of mind.");

      scan.close();
   }
}
```

The if Statement

- What do the following statements do?

```
if (total != stock + warehouse){
    inventoryError = true;
}
```

Sets a flag to true if the value of `total` is not equal to the sum of `stock` and `warehouse`.

- The precedence of the arithmetic operators is higher than the precedence of the equality and relational operators.

Comparing floating point values

Comparing floating point values

- You can use one of Java's *equality operators* or *relational operators to compare* two floating point values.

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

- You should rarely use the equality operator (`==`) when comparing two floating point values for equality (`float` or `double`).

Comparing Float Values

- Two floating point values are equal only if their underlying binary representations match exactly.
- Computations often result in slight differences that may be irrelevant.
- In many situations, you might consider two floating point numbers to be "close enough" even if they aren't exactly equal.

Comparing Float Values

- To determine the equality of two floats, you may want to use the following technique:

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

- If the difference between the two floating point values is less than the tolerance, they are considered to be equal.
- The tolerance could be set to any appropriate level, such as 0.000001.

Comparing characters

Comparing Characters

- Java character data is based on the Unicode character set.
- Unicode establishes a particular numeric value for each character, and therefore an ordering.
- We can use relational operators on character data based on this ordering.
  - For example, the character '+' is less than the character 'J' because it comes before it in the Unicode character set.
- ASCII Character

Comparing Characters

- In Unicode, the digit characters (0-9) are contiguous and in order.
- Likewise, the uppercase letters (A-Z) and lowercase letters (a-z) are contiguous and in order.

Characters	Unicode Values
0 – 9	48 through 57
A – Z	65 through 90
a – z	97 through 122

Comparing strings (Lexicographical Order)

Comparing Strings for equality

- In Java, a character string is an object.
- Don't use == or != to compare Objects such as strings for equality.

```
if (name1 == name2){
 System.out.println ("Same name");
}

if (name1 != name2){
 System.out.println ("Not same name");
}
```

- == and != compare addresses not contents.

Comparing Strings for equality

- The `equals` method can be called with strings to determine if two strings contain exactly the same characters in the same order.
- The `equals` method returns a boolean result.

```
if (name1.equals(name2)){
      System.out.println ("Same name");
}


if (!(name1.equals(name2))){
  System.out.println ("Not same name");
}
```

Compare strings

- We cannot use the following operators to sort strings.

`<`	less than
`>`	greater than
`<=`	less than or equal to
`>=`	greater than or equal to

- The `String` class contains a method called `compareTo` to determine if one string comes before another.

Compare strings

- A call to `name1.compareTo(name2)`
  - returns zero if `name1` and `name2` are equal (contain the same characters).
  - returns a negative value if `name1` is less than `name2`.
  - returns a positive value if `name1` is greater than `name2`.

Compare strings

```
if (name1.compareTo(name2) < 0){
   System.out.println (name1 + "comes first");
}else{
   if (name1.compareTo(name2) == 0){
      System.out.println ("Same name");
   }else{
      System.out.println (name2 + "comes first");
   }
}
```

- Because comparing characters and strings is based on a character set, it is called a *lexicographic ordering.*

Lexicographic Ordering

- Lexicographic ordering is not strictly alphabetical when uppercase and lowercase characters are mixed.
  - For example, the string `"Great"` comes before the string `"fantastic"` because all of the uppercase letters come before all of the lowercase letters in Unicode.
- Short strings come before longer strings with the same prefix (lexicographically).

  `"book"` comes before `"bookcase"`

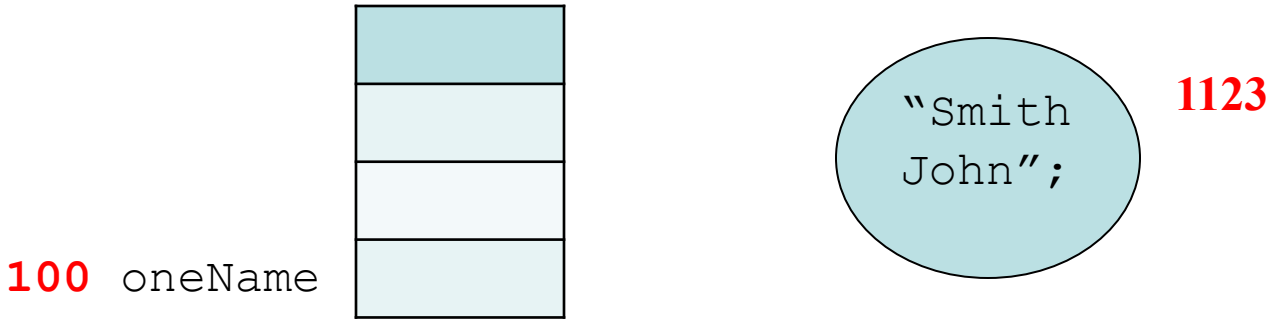
Comparing object vs. Comparing object references

Comparing Objects

- A String is an object. An object variable contains the address of the object.

```
String oneName = "Smith John";
```

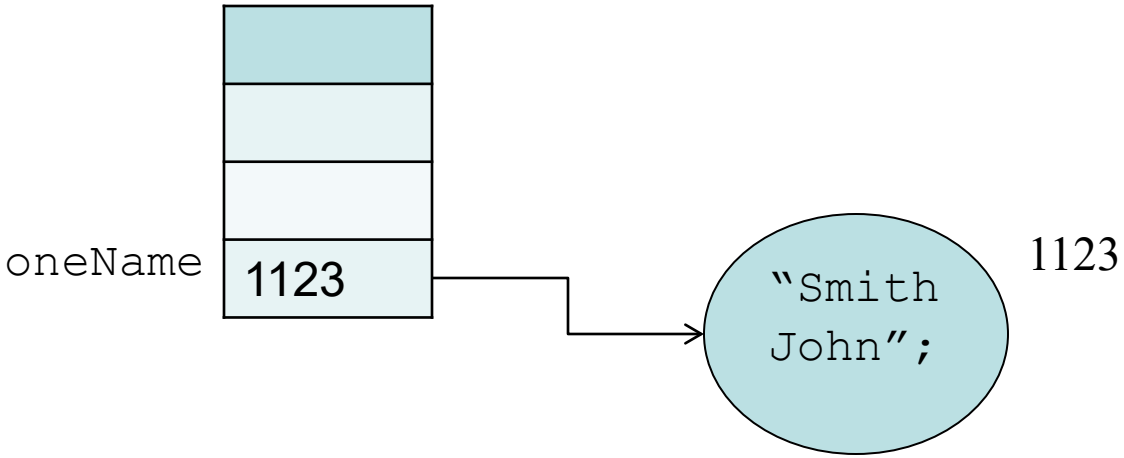
- Variable `oneName` is a named location in stack memory.



- `"Smith John"` is an object stored in Heap memory.
- Each has its own address.

Comparing Objects

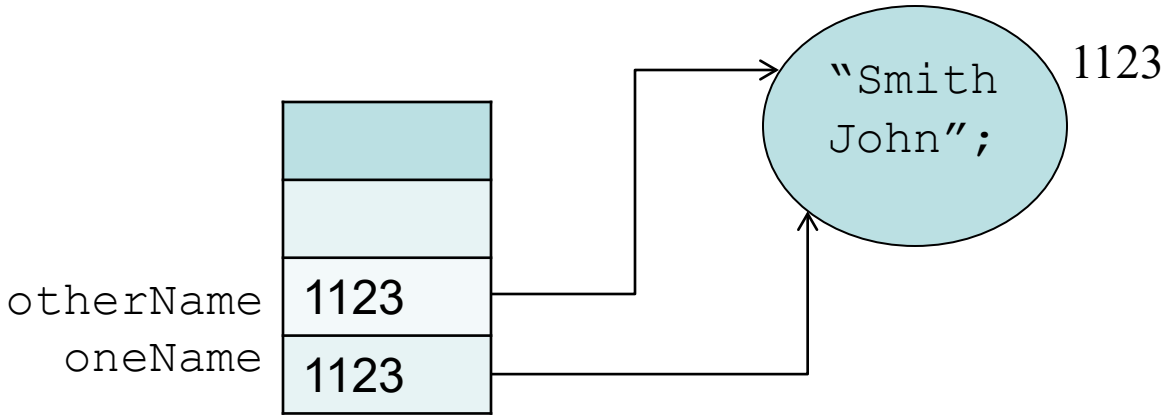
- Variable `oneName` contains the address of object `"Smith John"`.



Comparing Objects

- The == operator can be applied to objects – it returns true if the two references are aliases of each other.

```
String oneName = "Smith John";
String otherName = oneName;
```

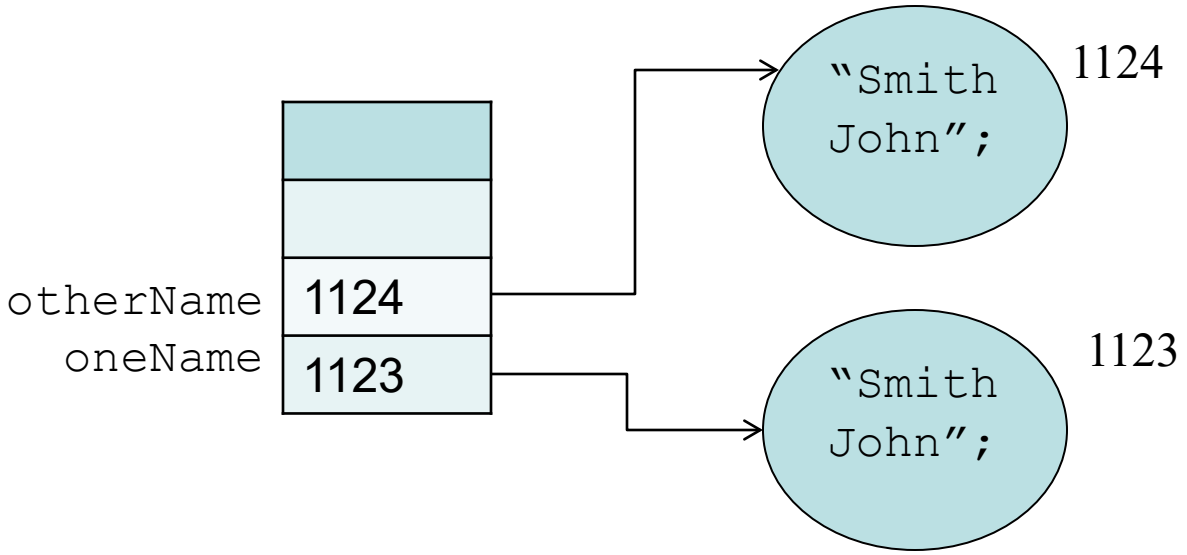


`oneName == otherName` **returns** `true`

Comparing Objects

- It returns false if the two references refer to two different objects.

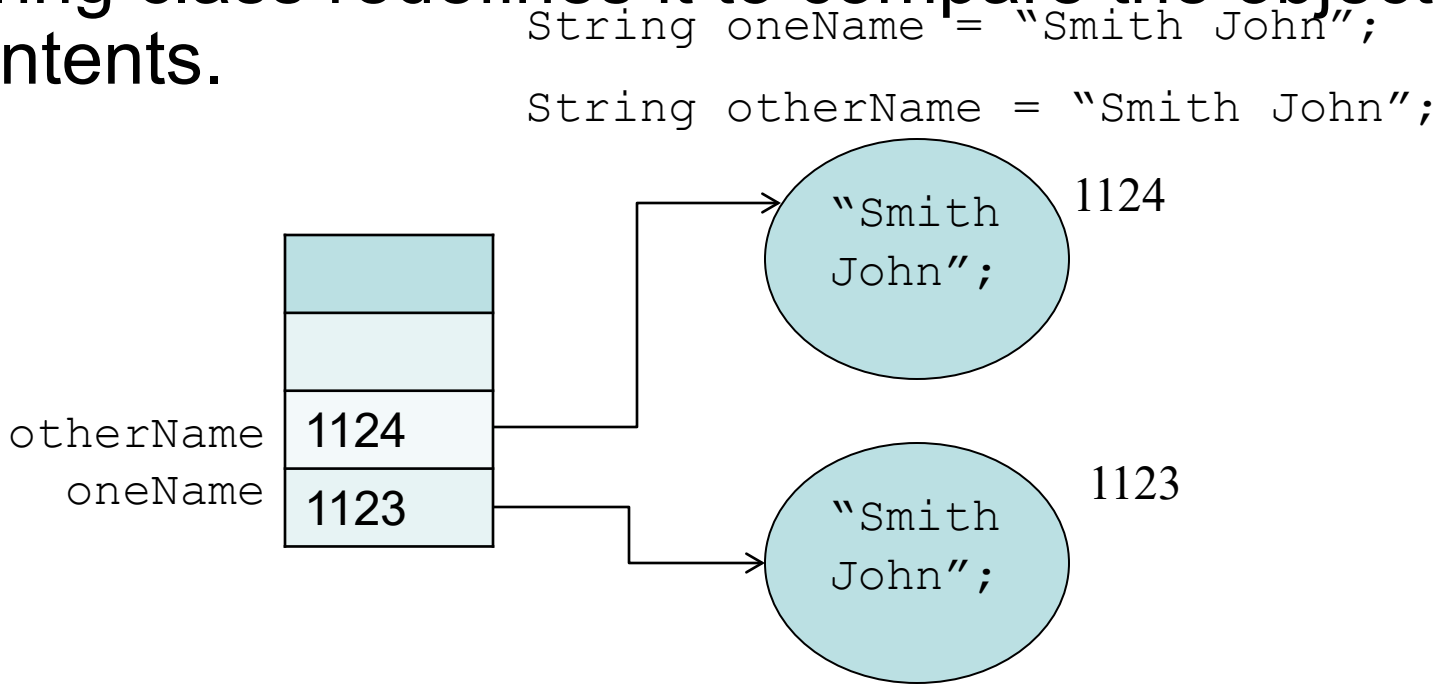
```
String oneName = "Smith John";
String otherName = "Smith John";
```



`oneName == otherName` **returns** `false`

Comparing Objects

- The `equals` method is defined for all objects, but unless we redefine it when we write a class, it has the same semantics as the == operator.
- String class redefines it to compare the object contents.



`oneName == otherName` **returns** `true`

Compound Boolean Expressions

- How do we model a complex situation as follows in one condition?
  - For example, if a student grade is greater than 70 and less than 80, the grade is B.

```
if (a student grade is greater than 70 and
    less than 80){
     Display grade B
}
```

- We have to use logical operators to combine two boolean expressions to create a compound boolean expression.

Compound Boolean Expressions

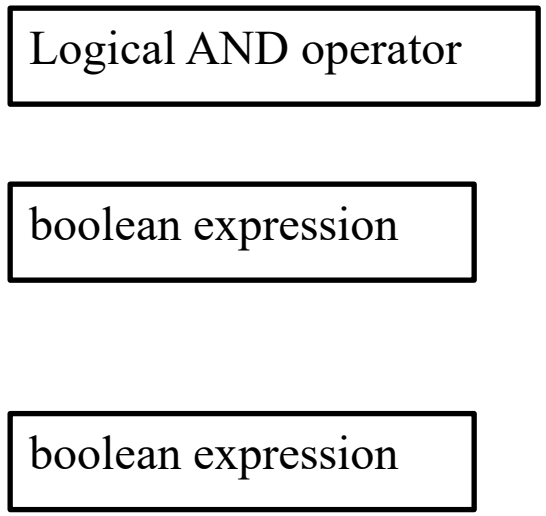
- The following condition

```
if (a student grade is greater than 70 and
    less than 80){
     Display grade B
}
```

Is represented as

```
if ( grade> 70 && grade < 80){

    System.out.println('B');
}
```



Logical Operators

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

`!`	Logical NOT	//unary operator
`&&`	Logical AND	// binary operator
`||`	Logical OR	// binary operators

- They all take boolean operands and produce boolean results.

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 a *truth table.*

`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.

Logical Operators

- Expressions that use logical operators can form complex conditions.

```
if ((total < MAX+5) && (!found)){
   System.out.println ("Processing…");
}
```

- All logical operators have lower precedence than the relational operators.
- Logical NOT has higher precedence than logical AND and logical OR.

Logical Operators

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

a	b	a && b	a || b
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

Boolean Expressions

- Specific expressions can be evaluated using truth tables.

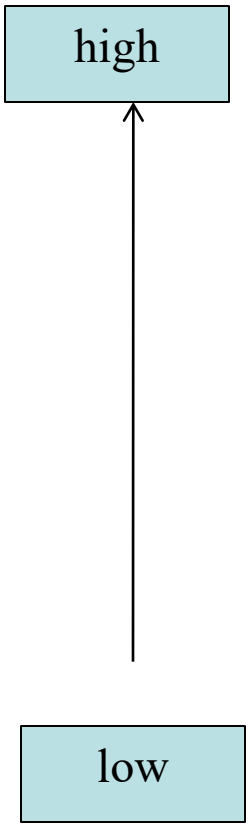
`total < MAX`	`found`	`!found`	`total < MAX && !found`
false	false	true	false
false	true	false	false
true	false	true	true
true	true	false	false

Precedence

Arithmetic operators

Equality and Relational operators

Logical operators



Short-Circuited Operators

- The processing of logical AND and logical OR is “short-circuited”.
- If the left operand is sufficient to determine the result, the right operand is not evaluated.

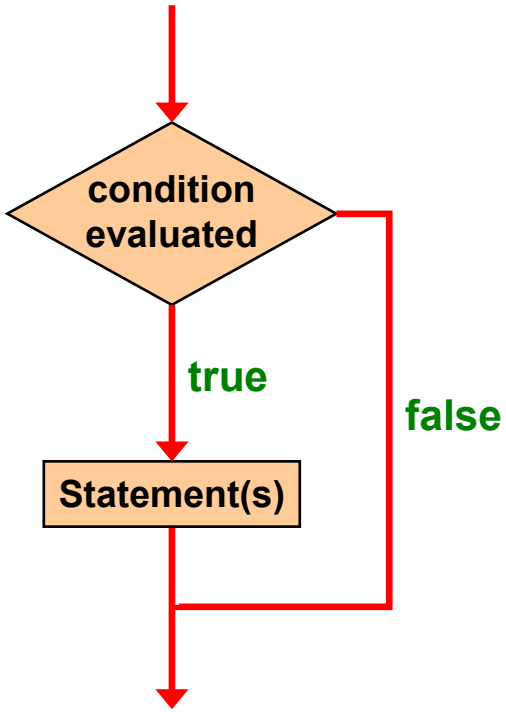
```
if ((count != 0) && (total/count > MAX)){
   System.out.println ("Testing…");
}
```

- This type of processing must be used carefully.

If, If-else statements

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(s);
}
```

If the *condition* is true, the *statement(s)* are executed. If it is false, the *statement(s)* are skipped.

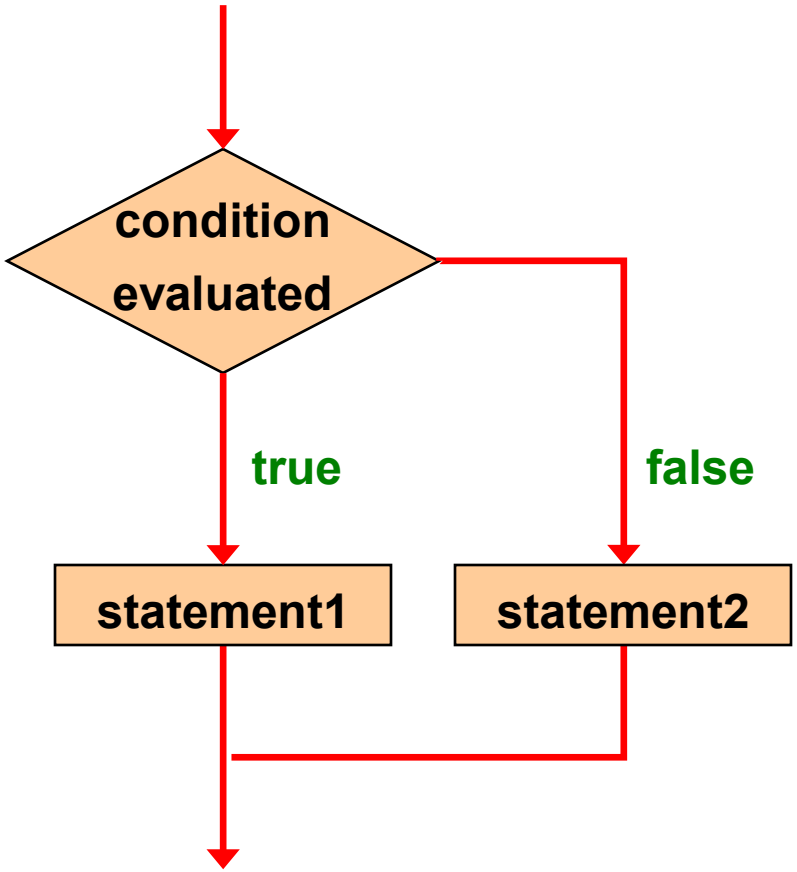
The if-else Statement

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



```java
package chapter03;

import java.text.NumberFormat;
import java.util.Scanner;

/**
 * Calculates wages.
 * @author Lewis Loftus
 * @version 1.0
 */
public class Wages{
    /**
     * Calculates wages.
     * @param args A reference to a string array containing command-line arguments
     */
    public static void main (String[] args){
      final double RATE = 8.25;  // regular pay rate
      final int STANDARD = 40;   // standard hours in a work week

      Scanner scan = new Scanner (System.in);
      double pay = 0.0;

      System.out.print ("Enter the number of hours worked: ");
      int hours = scan.nextInt();
      System.out.println ();

      // Pay overtime at "time and a half"
      if (hours > STANDARD){
         pay = STANDARD * RATE + (hours-STANDARD) * (RATE * 1.5);
      }else{
         pay = hours * RATE;
      }
      NumberFormat fmt = NumberFormat.getCurrencyInstance();
      System.out.println ("Gross earnings: " + fmt.format(pay));

      scan.close();
   }
}
```

```java
package chapter03;

import java.util.Scanner;
import java.util.Random;

/**
 * A simple guessing game.
 * @author Lewis Loftus
 * @version 1.0
 */
public class Guessing{
   /**
    * Plays a simple guessing game with the user.
    * @param args A reference to a string array containing command-line arguments
    */
   public static void main (String[] args){
      final int MAX = 10;
      int answer, guess;

      Scanner scan = new Scanner (System.in);
      Random generator = new Random();
      answer = generator.nextInt(MAX) + 1;
      System.out.print ("I'm thinking of a number between 1 and "
                        + MAX + ". Guess what it is: ");

      guess = scan.nextInt();

      if (guess == answer){
         System.out.println ("You got it! Good guessing!");
      }else{
         System.out.println ("That is not correct, sorry.");
         System.out.println ("The number was " + answer);
      }

      scan.close();
   }
}
```

if else if … else Statements

```
System.out.print("Enter a day of a week: ");
dayOfWeek = input.next();
if(dayOfWeek.equalsIgnoreCase("Monday")){
    System.out.print("It is Monday");
}else if(dayOfWeek.equalsIgnoreCase("Tuesday")){
    System.out.print("It is Tuesday");
}else if(dayOfWeek.equalsIgnoreCase("Wednesday")){
    System.out.print("It is Wednesday");
}else if(dayOfWeek.equalsIgnoreCase("Thursday")){
    System.out.print("It is Thursday");
}else if(dayOfWeek.equalsIgnoreCase("Friday")){
    System.out.print("It is Friday");
}else if(dayOfWeek.equalsIgnoreCase("Saturday")){
    System.out.print("It is Saturday");
}else{
    // SUNDAY
    System.out.print("It is Sunday");
}
```

Nested if Statements (More Options)

- The statement executed as a result of an `if` statement or `else` clause could be another `if` statement.
- These are called *nested if statements.*

```java
package chapter03;

import java.util.Scanner;

/**
 * Determine the smallest values amoung a list of values.
 * @author Lewis Loftus
 * @version 1.0
 */
public class MinOfThrees{
    /**
     * Reads three integers from the user and determines the smallest value.
     * @param args A string array that contains command-line arguments
     */
    public static void main (String[] args){
        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);
        scan.close();
    }
}
```

The switch statements

The switch Statement

- The *switch statement* provides another way 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 statement associated with the first case value that matches.

The switch Statement

- The general syntax of a `switch` statement is:

switch and case are reserved Words.

```
switch ( expression ){
    case value1 :
        statements1;
        break;
    case value2 :
        statements2;
        break;
    case value3 :
        statements3;
        break;
    case  ...
    default:
        statements;
}
```

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 may be appropriate, but often we want to execute only the statements associated with one case.

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.
- If there is no default case, and no other value matches, control falls through to the statement after the switch.

The switch Statement

- The expression of a `switch` statement must result in an *integral type*, meaning an integer (`byte`, `short`, `int`,),`char` , or a `String`(Java 7 or newer).
- It cannot be a `boolean` value or a floating point value (`float` or `double`).
- The implicit boolean condition in a `switch` statement is equality.
- You cannot perform relational checks with a `switch` statement.

```java
package chapter03;

import java.util.Scanner;

/**
 * Creates a grade report for a student.
 * @author  Lewis Loftus
 * @version 1.0
 */
public class GradeReport{
    /**
     * Reads a grade from the user and prints comments accordingly.
     * @param args A string array that contains command-line arguments
     */
    public static void main (String[] args){
        int grade, category;
        Scanner scan = new Scanner (System.in);

        System.out.print ("Enter a numeric grade (0 to 100): ");
        grade = scan.nextInt();
        category = grade / 10;

        System.out.print ("That grade is ");
        switch (category){
            case 10:
                System.out.println ("a perfect score. Well done.");
                break;
            case 9:
                System.out.println ("well above average. Excellent.");
                break;
            case 8:
                System.out.println ("above average. Nice job.");
                break;
            case 7:
                System.out.println ("average.");
                break;
            case 6:
                System.out.println ("below average. You should see the");
                System.out.println ("instructor to clarify the material "
                        + "presented in class.");
                break;
            default:
                scan.close();
        }
    }
}
```

The Conditional Operator

- Java has a *conditional operator* that uses a boolean condition to determine which of two expressions is evaluated.
- Its syntax is:

```
condition ? expression1 : expression2
```

- If the `condition` is true, `expression1` is evaluated; if it is false, `expression2` is evaluated.
- The value of the entire conditional operator is the value of the selected expression.

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 than `num2`, then `num1` is assigned to `larger`; otherwise, `num2` is assigned to `larger`.
- The conditional operator is *ternary* because 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 "Dimes" is printed.

Summary

- Conditional statements
  - Comparing Data
  - The if, if-else, and switch Statements