Last time

We covered:

- primitive data types
- declaration, initialization, assignment of variables
- expressions and operator precedence
- data conversions
- accepting input from the user
Review: Primitive Data Types

Of the following types, which one cannot store a numeric value?

A) int
B) byte
C) float
D) char
E) all of these can store numeric values
Review: Primitive Data Types

Of the following types, which one cannot store a numeric value?

A) int
B) byte
C) float
D) char
E) all of these can store numeric values
Correct Mistakes

• // The following program has several errors
• Fix these errors

public class CorrectMe
    public static main(String[] args) {
        System.out.println(Hello world);
        system.out.Pritnln("Do you like this program"?);
        System.out. println()

        System.println("I wrote it myself.";
    }

See CorrectMe.java
Review: What is the result of these?

```c
int z = 5 / 2;
float z = 5 / 2;
double z = 5 / 2;
```
Review: Remainder

• The remainder operator (%) returns the remainder after dividing the first operand by the second

\[ 14 \% 3 \quad \text{equals} \quad 2 \]
\[ 8 \% 12 \quad \text{equals} \quad 8 \]
What do the following expressions evaluate to?

3.0 / 2.0 + 4.1

"hi" + (1 + 1) + "u"

12 / 5 + 8 / 4

42 % 5 + 16 % 3

"cs" + 2 + 6

2 + 6 + "cs"
Review of Type Casting

• See Char.java
Conditionals and Loops

• Now we will examine programming statements that allow us to:
  – make decisions
  – repeat processing steps in a loop
Outline

Boolean Expressions
The if Statement
The Conditional Operator ( ? : )
The switch Statement
Flow of Control

• The order of statement execution is called the *flow of control*

• Unless specified otherwise, the order of statement execution through a method is linear: one after another

• Some programming statements allow us to make decisions and perform repetitions

• These decisions are based on *boolean expressions* (also called *conditions*) that evaluate to true or false
Conditional Statements

• A *conditional statement* lets us choose *which statement will be executed next*

• The Java conditional statements are the:

  – if and if-else statement
  – switch statement
Boolean expression

• Boolean expression is just a test for a condition
  – Eventually, evaluates to true or false
Value comparisons

• 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* (=)
Relational Operators

• Note that these relational operators are for comparing primitive data types only.
• char values are compared according to their positions in the UNICODE table
• You can only use == or != for boolean data type
• Since computations may generate a round-off error in 15th decimal place in a double value, use
  \[ \text{Math.abs(calculated-expected)} \leq 1E-15 \]
  Instead of
  \[ \text{calculated == expected} \]
• See **BoolTest.java**
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

- 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*:

<table>
<thead>
<tr>
<th>( a )</th>
<th>( !a )</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>
Logical AND and Logical OR

- 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`

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>a &amp;&amp; b</td>
<td>a</td>
</tr>
<tr>
<td>----</td>
<td>----</td>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>
Logical Operators

• Expressions that use logical operators can form complex conditions

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

• All logical operators have lower precedence than the relational operators

• The `!` operator has higher precedence than `&&` and `||`
Boolean Expressions

• Specific expressions can be evaluated using truth tables

<table>
<thead>
<tr>
<th>total &lt; MAX</th>
<th>found</th>
<th>!found</th>
<th>total &lt; MAX &amp;&amp; !found</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>
Short-Circuit Evaluations

• The processing of && and || is “short-circuited”

• Stop evaluating the boolean expression as soon as we know the answer

• Consider:

```java
boolean flag = true, p;
p = 5 > 3 || flag;
```

The second test, flag, is not evaluated at all.
A useful example

• If the left operand is sufficient to determine the result, the right operand is not evaluated

\[ p = (\text{count} \neq 0) \land (\text{total}/\text{count} > \text{MAX}) \]
Outline

Boolean Expressions

The if Statement

The Conditional Operator ( ? : )

The switch Statement
The if Statement

```java
if ( condition ){
    statements;
}

if ( condition )  // can omit braces
    statement;     // if there is one statement
```
The if Statement

• Let's now look at the `if` statement in more detail
• The *if statement* has the following syntax:

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

*if* is a Java reserved word

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

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

```java
if((num % 2) == 0 ){
    System.out.println ( "num is even" );
}
```
Logic of an if statement
Indentation

• The statement controlled by the `if` statement is indented to indicate that relationship

• The use of a consistent indentation style makes a program easier to read and understand

"Always code as if the person who ends up maintaining your code will be a violent psychopath who knows where you live."

-- Martin Golding
Quick Check
What do the following statements do?

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

```java
if (found || !done)
    System.out.println("Ok");
```
Quick Check
What do the following statements do?

```java
if (total != (stock + warehouse))
    inventoryError = true;
```
Sets the boolean variable to true if the value of `total` is not equal to the sum of `stock` and `warehouse`

```java
if (found || !done)
    System.out.println("Ok");
```
Prints "Ok" if `found` is true or `done` is false
If Statement

• See Age.java
//********************************************************************
// Age.java       Author: Lewis/Loftus
//
// Demonstrates the use of an if statement.
//*********************************************************************

import java.util.Scanner;

public class Age
{
    //---------------------------------------------------------------
    // Reads the user's age and prints comments accordingly.
    //---------------------------------------------------------------
    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();

        continue
continue

    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.");
    }
}
```java
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.");
```

**Sample Run**
Enter your age: 47
You entered: 47
Age is a state of mind.

**Another Sample Run**
Enter your age: 12
You entered: 12
Youth is a wonderful thing. Enjoy.
Age is a state of mind.
The if-else Statement

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

```java
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

• See [Wages.java](#)
Logic of an if-else statement

condition evaluated

true

false

statement1

statement2
import java.util.Scanner;

public class Wages {

  //-----------------------------------------------------------------
  //  Reads the number of hours worked and calculates wages.
  //-----------------------------------------------------------------
  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;

    continue
continue

    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;

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

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

Sample Run

Enter the number of hours worked: 46
Gross earnings: $404.25
If statement

```java
if ((num % 2) == 0 )
{
    System.out.println ( "num is even");
}
else
{
    System.out.println ( "num is odd");
}
```
NOTICE

• Remember that indentation is for the human reader, and is ignored by the compiler

```java
if (depth >= UPPER_LIMIT)
    delta = 100;
else
    System.out.println("Reseting Delta");
delta = 0;
```

• Despite what the indentation implies, `delta` will be set to 0 no matter what
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

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

• The if clause, or the else clause, or both, could govern block statements

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

• See Guessing.java
import java.util.*;

public class Guessing
{
    //----------------------------------------------------------------------------
    //  Plays a simple guessing game with the user.
    //----------------------------------------------------------------------------
    public static void main (String[] args)
    {
        final int MAX = 10;
        int answer, guess;

        answer = 9;
        Scanner scan = new Scanner (System.in);
        continue
continue

    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);
    }
}
Sample Run

I'm thinking of a number between 1 and 10. Guess what it is: 6
That is not correct, sorry.
The number was 9

```java
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);
}
```
Nested if Statements

• The statement executed as a result of an if or else clause could be another if statement.

• These are called *nested if statements*.

• An else clause is matched to the last unmatched if (no matter what the indentation implies).

• Braces can be used to specify the if statement to which an else clause belongs.

• See MinOfThree.java
import java.util.Scanner;

public class MinOfThree
{
    //------------------------------------------------------------------------------
    //    Reads three integers from the user and determines the smallest
    //    value.
    //------------------------------------------------------------------------------
    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();
    
    continue
}
continue

    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);
}
continue

```java
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);
```
Finding the minimum of 3 integers

• Do we really need a nested if statement to find the minimum of 3 integer numbers?

• The answer is no, see MinOfThree2.java

```java
// Assume num1 is the minimum
min = num1;

// Test if num2 is less than min, and update min if necessary
if (num2 < min)
    min = num2;

// Test if num3 is less than min, and update min if necessary
if (num3 < min)
    min = num3;
```
Question

Write a Java program to input the overall grade of a student and output his/her letter grade according to the criteria below:

<table>
<thead>
<tr>
<th>Grade</th>
<th>Letter</th>
</tr>
</thead>
<tbody>
<tr>
<td>90-100</td>
<td>A</td>
</tr>
<tr>
<td>80-89</td>
<td>B</td>
</tr>
<tr>
<td>70-79</td>
<td>C</td>
</tr>
<tr>
<td>60-69</td>
<td>D</td>
</tr>
<tr>
<td>0-59</td>
<td>F</td>
</tr>
</tbody>
</table>
Solution

- **ComputeLetterGrade1.java** uses if-statement only
- **ComputeLetterGrade2.java** uses nested-if
- **ComputeLetterGrade3.java** uses switch (NEXT TOPIC!)
Outline

Boolean Expressions

The if Statement

The Conditional Operator ( ? : )

The switch Statement
Conditional Operator ( ? : )

• Conditional operator is also known as the ternary operator. Another name is arithmetic if. This operator consists of three operands and is used to evaluate boolean expressions. The goal of the operator is to decide which value should be assigned to the variable. The operator is written as:

• (expression) ? value if true : value if false
Conditional Operator (?:) Example

- Test.java
```java
public class Test {

    public static void main(String args[]) {
        int a, b;
        a = 10;
        b = (a == 1) ? 20 : 30;
        System.out.println( "Value of b is : " + b );

        b = (a == 10) ? 20 : 30;
        System.out.println( "Value of b is : " + b );

        System.out.println ( (a>5) ? a%2 : -a);
    }
}
```
public class Test {
    public static void main(String args[]){
        int a , b;
        a = 10;
        b = (a == 1) ? 20: 30;
        System.out.println( "Value of b is : " + b );

        b = (a == 10) ? 20: 30;
        System.out.println( "Value of b is : " + b );

        System.out.println ( (a>5) ? a%2 : -a);
    }
}
Outline

Boolean Expressions

The `if` Statement

The Conditional Operator `( ? : )`

The `switch` Statement
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 (expression)
{
    case value1 :
        statement-list1
    case value2 :
        statement-list2
    case value3 :
        statement-list3
    case ...  

}
```

switch and case are reserved words

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

- An example of a switch statement:

```java
switch (option)
{
    case 'A':
        aCount++;
        break;
    case 'B':
        bCount++;
        break;
    case 'C':
        cCount++;
        break;
}
```
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 type of a switch expression must be integers, characters, or enumerated types

• As of Java 7, a switch can also be used with strings

• You cannot use a switch with floating point values

• The implicit boolean condition in a switch statement is equality

• You cannot perform relational checks with a switch statement

• See GradeReport.java
import java.util.Scanner;

public class GradeReport {
    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.println ("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:
        System.out.println("not passing.");
}
In the `switch` statement, the `case` clause 9 is executed because the grade is 91. The corresponding message is printed to the console:

```
System.out.println("a perfect score. Well done.");
bike;
```

The message printed is:

```
That grade is well above average. Excellent.
```

The output will be:

```
Sample Run
Enter a numeric grade (0 to 100): 91
That grade is well above average. Excellent.
```