CS101
Algorithms and Programming I

Week 2

17/06/2013

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?

```
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   equals   2
8 % 12   equals   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

The `while` Statement

The `for` Statement

The `do` 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

```
Math.abs(calculated-expected) <=1E-15
```

Instead of

```
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*:

`a`	`!a`
true	false
false	true

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`

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

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
- The `!` operator has higher precedence than `&&` and `||`

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

Short-Circuit Evaluations

- The processing of `&&` and `||` is "short-circuited"
- Stop evaluating the boolean expression as soon as we know the answer
- Consider:

```
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 = (count != 0) && (total/count > MAX)
```

Outline

Boolean Expressions

The `if` Statement

The Conditional Operator (`?` `:`)

The `switch` Statement

The `while` Statement

The `for` Statement

The `do` Statement

The if Statement

```
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:

`if` is a Java reserved word

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

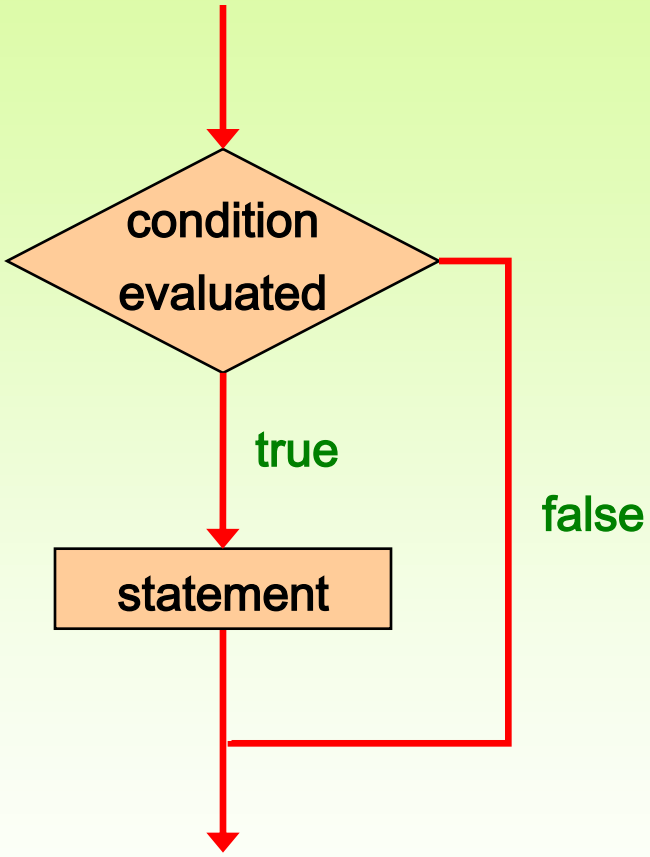
```
if ( condition )
   statement;
```

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

If statement

```
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?

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

```
if (found || !done)
   System.out.println("Ok");
```

Quick Check

What do the following statements do?

```
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`

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

Sample Run

```
Enter your age: 47
You entered: 47
Age is a state of mind.
```

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.");
    }
}
```

Another Sample Run

```
Enter your age: 12
You entered: 12
Youth is a wonderful thing. Enjoy.
Age is a state of mind.
```

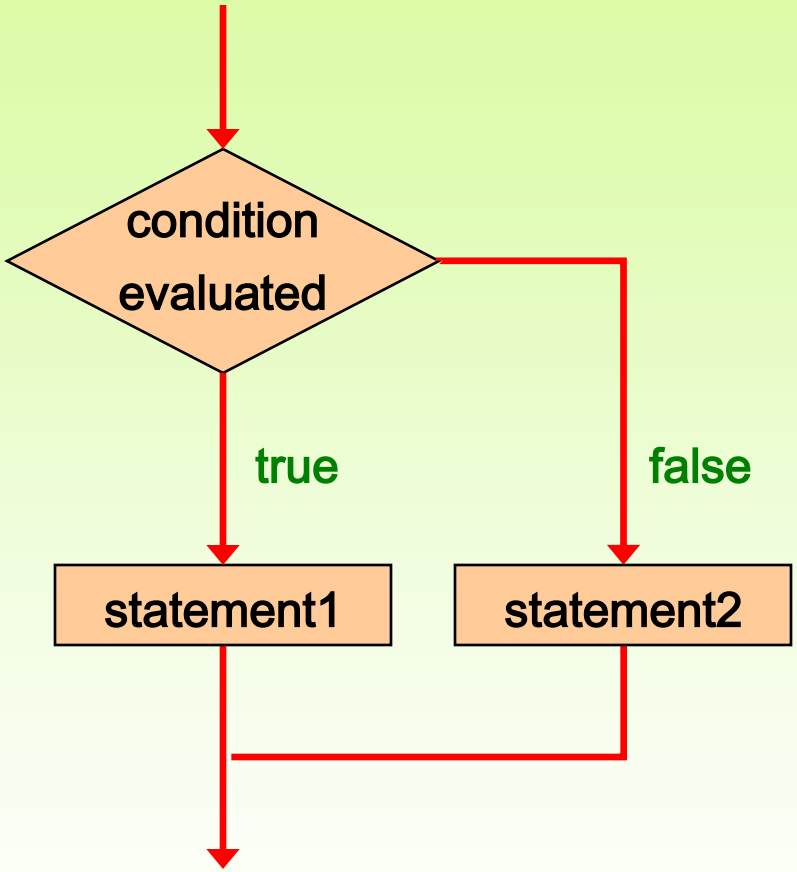
The if-else Statement

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

```
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



```
//********************************************************************
//  Wages.java       Author: Lewis/Loftus
//
//  Demonstrates the use of an if-else statement.
//********************************************************************

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);
   }
}
```

continue

```
        System.
        int hou

        System.

        // 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
```

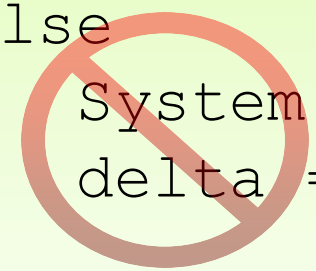
If statement

```
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

```
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

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

Block Statements

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

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

- See Guessing.java

```
//********************************************************************
//  Guessing.java       Author: Lewis/Loftus
//  Modified by Oznur Tastan
//  Demonstrates the use of a block statement in an if-else.
//********************************************************************

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
```

```
      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`

```
//********************************************************************
//  MinOfThree.java       Author: Lewis/Loftus
//
//  Demonstrates the use of nested if statements.
//********************************************************************

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

```
        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);
    }
}
```

Sample Run

```
Enter three integers:
84   69   90
Minimum value: 69
```

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

```
// 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

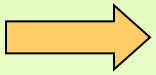
Write a Java program to input the overall grade of a student and output his/her letter grade according to the criteria below:

90-100	A
80-89	B
70-79	C
60-69	D
0-59	F

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

The `while` Statement

The `for` Statement

The `do` 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

```
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(Str
      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);
   }
}
```

Sample Run

```
Value of b is : 30

Value of b is : 20

0
```

Outline

Boolean Expressions

The `if` Statement

The Conditional Operator (`?` `:`)

The `switch` Statement

The `while` Statement

The `for` Statement

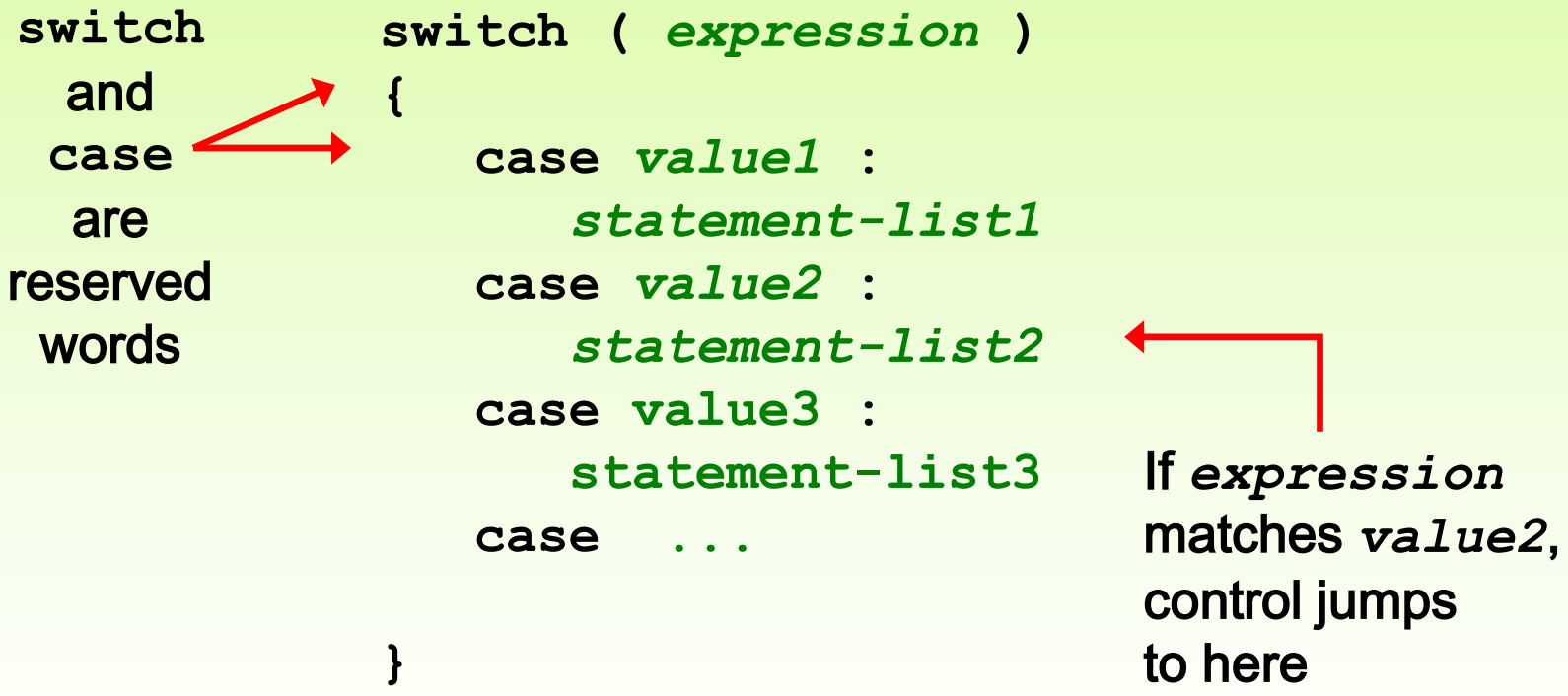
The `do` Statement

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:



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:

```
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`

```
//********************************************************************
//  GradeReport.java       Author: Lewis/Loftus
//
//  Demonstrates the use of a switch statement.
//********************************************************************

import java.util.Scanner;

public class GradeReport
{
   //-----------------------------------------------------------------
   //  Reads a grade from the user and prints comments accordingly.
   //-----------------------------------------------------------------
   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 ");
```

continue

continue

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

continue

Sample Run

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

```
        swi
        {
                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.");
        }
    }
}
```

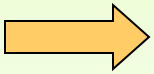
Outline

Boolean Expressions

The `if` Statement

The Conditional Operator (`?` `:`)

The `switch` Statement



The `while` Statement

The `for` Statement

The `do` Statement

Repetition (Iteration) Statements (Loops)

- *Repetition statements* allow us to execute a statement multiple times
- Often they are referred to as *loops*
- Like conditional statements, they are controlled by boolean expressions
- Java has three kinds of repetition statements: `while`, `do`, and `for` loops

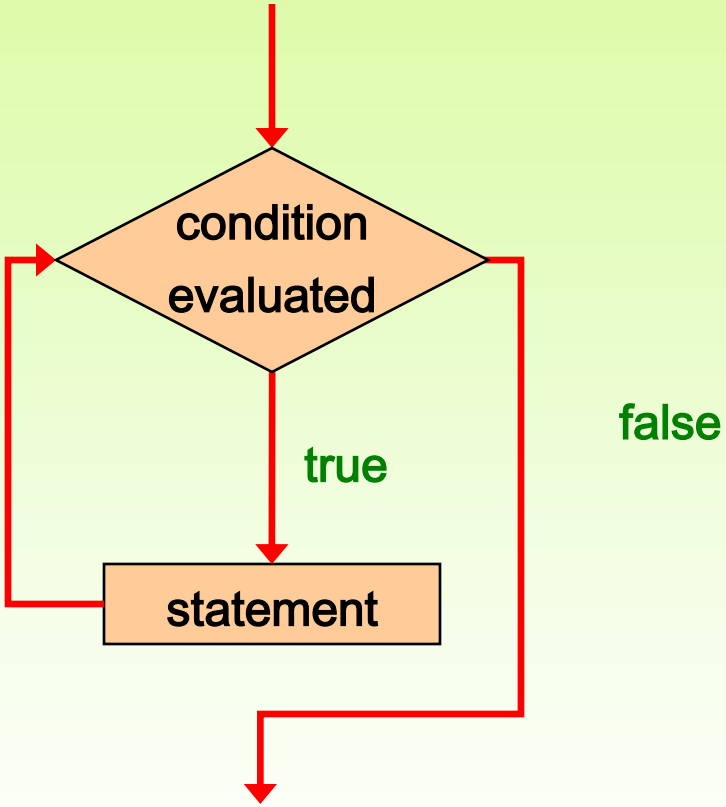
The while Statement

- A *while statement* has the following syntax:

```
while ( condition )
    statement;
```

- If the **`condition`** is true, the **`statement`** is executed
- Then the condition is evaluated again, and if it is still true, the statement is executed again
- **The statement is executed repeatedly until the condition becomes false**

Logic of a while Loop



The while Statement

- An example of a while statement:

```
int count = 1;
while (count <= 5)
{
   System.out.println (count);
   count++;
}
```

- If the condition of a `while` loop is false initially, the statement is never executed
- Therefore, the body of a `while` loop will execute zero or more times

Sentinel Values

- Let's look at some examples of loop processing
- A loop can be used to maintain a *running sum*
- A *sentinel value* is a special input value that represents the end of input
- See `Average.java`

```
//********************************************************************
//  Average.java       Author: Lewis/Loftus
//
//  Demonstrates the use of a while loop, a sentinel value, and a
//  running sum.
//********************************************************************

import java.text.DecimalFormat;
import java.util.Scanner;

public class Average
{
   //-----------------------------------------------------------------
   //  Computes the average of a set of values entered by the user.
   //  The running sum is printed as the numbers are entered.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      int sum = 0, value, count = 0;
      double average;

      Scanner scan = new Scanner (System.in);

      System.out.print ("Enter an integer (0 to quit): ");
      value = scan.nextInt();
```

continue

continue

```
      while (value != 0)  // sentinel value of 0 to terminate loop
      {
         count++;

         sum += value;
         System.out.println ("The sum so far is " + sum);

         System.out.print ("Enter an integer (0 to quit): ");
         value = scan.nextInt();
      }
```

continue

continue

```
      System.out.println ();

      if (count == 0)
         System.out.println ("No values were entered.");
      else
      {
         average = (double)sum / count;
         System.out.println ("The average is " + average);
      }
   }
}
```

continue

```
        System.ou

        if (count
           System
        else
        {
           average

           Decima
           System
        }
    }
}
```

```
at(average));
```

Sample Run

```
Enter an integer (0 to quit): 25
The sum so far is 25
Enter an integer (0 to quit): 164
The sum so far is 189
Enter an integer (0 to quit): -14
The sum so far is 175
Enter an integer (0 to quit): 84
The sum so far is 259
Enter an integer (0 to quit): 12
The sum so far is 271
Enter an integer (0 to quit): -35
The sum so far is 236
Enter an integer (0 to quit): 0

The average is 39.333
```

Input Validation

- A loop can also be used for *input validation*, making a program more *robust*
- It's generally a good idea to verify that input is valid (in whatever sense) when possible
- See `WinPercentage.java`
- Input validation using while added to compute letter grade example: `ComputeLetterGrade1DataValidation.java`
- Input validation using if (program stops):
- ComputeLetterGrade2DataValidationIf.java

```
//********************************************************************
//  WinPercentage.java       Author: Lewis/Loftus
//
//  Demonstrates the use of a while loop for input validation.
//********************************************************************

import java.util.Scanner;

public class WinPercentage
{
   //-----------------------------------------------------------------
   //  Computes the percentage of games won by a team.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      final int NUM_GAMES = 12;
      int won;
      double ratio;

      Scanner scan = new Scanner (System.in);

      System.out.print ("Enter the number of games won (0 to "
                        + NUM_GAMES + "): ");
      won = scan.nextInt();
```

continue

continue

```
      while (won < 0 || won > NUM_GAMES)
      {
         System.out.print ("Invalid input. Please reenter: ");
         won = scan.nextInt();
      }

      ratio = (double)won / NUM_GAMES;


      System.out.println ();
      System.out.println ("Winning percentage: " + ratio);
   }
}
```

continue

```
        while
        {
            Sy
            wo
        }

        ratio = (double)won / NUM_GAMES;


        System.out.println ();
        System.out.println ("Winning percentage: " + ratio + "%" );
    }
}
```

Sample Run

```
Enter the number of games won (0 to 12): -5
Invalid input. Please reenter: 13
Invalid input. Please reenter: 7

Winning percentage: 58%
```

Infinite Loops

- The body of a `while` loop eventually must make the condition false
- If not, it is called an *infinite loop*, which will execute until the user interrupts the program
- This is a common logical error
- You should always double check the logic of a program to ensure that your loops will terminate normally

Infinite Loops

- An example of an infinite loop:

```
double count = 1;
while (count <= 25)
{
   System.out.println (count);
   count = count - 1;
}
```

- This loop will continue executing until interrupted

Infinite or Finite?

- FiniteOrInfinite.java

```
public class FiniteOrInfinite {

   public static void main (String[] args) {

         byte count = (byte) 1;
         while (count <= (byte) 25) {
                  System.out.println (count);
                  count--;
         }

         System.out.println (count);
   }
}
```

The answer is "finite loop" and the output is as follows:

```
1
0
-1
-2
-3
-4
-5
-6
-7
-8
-9

.
.
.

-124
-125
-126
-127
-128
127
```

The minimum value for byte type is -128 and subtracting 1 from the minimum value results in the maximum value for byte type!

While Loop Example

Program calculates the sum of digits of an integer

See `SumOfDigits.java`

Outline

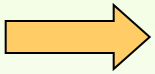
Boolean Expressions

The `if` Statement

The Conditional Operator (`?` `:`)

The `switch` Statement

The `while` Statement



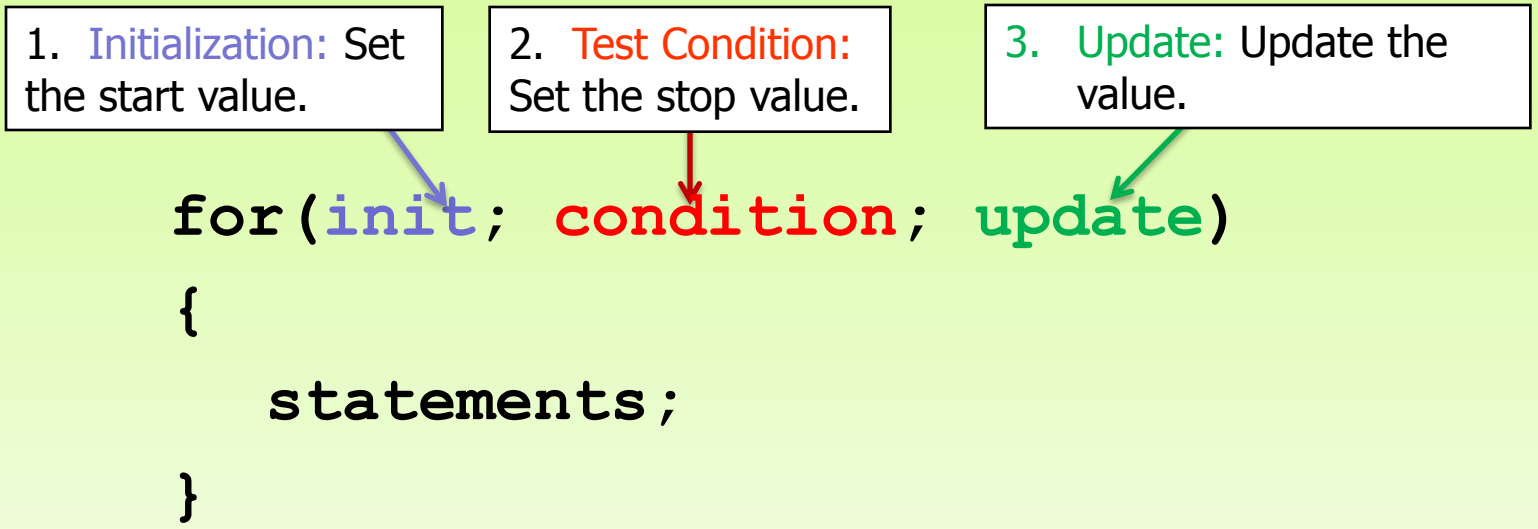
The `for` Statement

The `do` Statement

For Loops

- Another type of loop in Java is the `for` loop
- It is very good for definite repetition.
- All the parts (initialization, condition testing and update step) are in one place.
- Since the expressions are all in one place, many people prefer **`for`** to **`while`** when the number of iterations is known.

The for Loop Format



- **initialization** done once at the start of loop
- **condition** checked before every iteration through the loop
- we execute **the statements** if the condition is true
- **update** every time after the **statements**
- **Any of the initialization, condition and update parts may be omitted, but use of semicolons is a must!**

The for Loop Example

1. Initialization: Set the start value.

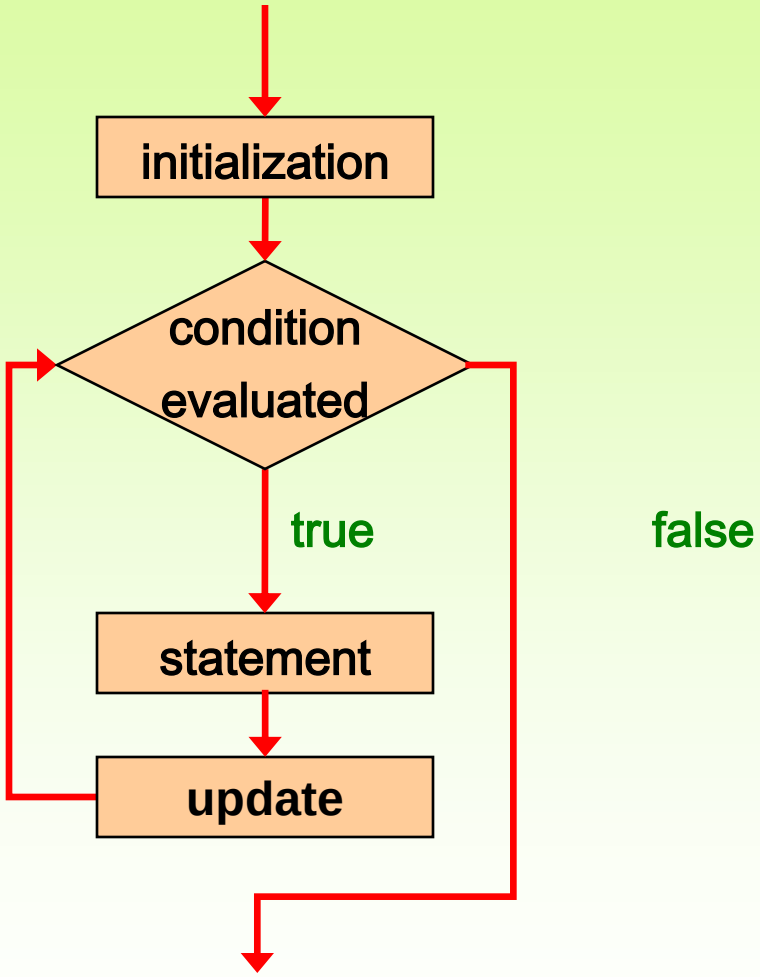
2. Test Condition: Set the stop value.

3. Update: Update the value.

```
for (int num = 0; num < 5; num++ )
{
   System.out.println(num);

}
```

Note that num is a valid identifier only within the for loop, but not outside the for loop.

Logic of a for Loop



For Loop Variations

- The limit can be a variable:

```
for ( i = 1; i <= limit; i++)
```

  - This counts from 1 to limit

- Update may be negative:

```
for (i = 100; i >= 1; i--)
```

  - This counts from 100 down to 1.

- Update may be greater than 1:

```
for (i = 100; i >= 5; i -= 5)
```

  - This counts from 100 down to 5 in steps of 5

The for Loop

```
for (int i = 6; i > 0; i--)
{
    System.out.println(i);

}
```

The for Loop

```
int i;
for (i = 100; i > 0; i-= 10)
{
    System.out.println(i);

}
```

The for Loop

- If the loop continuation condition is initially **`false`**
  - The body of the **`for`** structure is not performed
  - Control proceeds with the next statement after the **`for`** structure

The for Loop

Write a program to input two integer numbers, say `value` and `limit`, and then display the multiples of `value` from `value` to `limit`.

- See **`Multiples.java`**

```
//********************************************************************
//  Multiples.java       Author: Lewis/Loftus
//
//  Demonstrates the use of a for loop.
//********************************************************************

import java.util.Scanner;

public class Multiples
{
   //-----------------------------------------------------------------
   //  Prints multiples of a user-specified number up to a user-
   //  specified limit.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      final int PER_LINE = 5;
      int value, limit, mult, count = 0;

      Scanner scan = new Scanner (System.in);

      System.out.print ("Enter a positive value: ");
      value = scan.nextInt();
```

continue

continue

```
      System.out.print ("Enter an upper limit: ");
      limit = scan.nextInt();

      System.out.println ();
      System.out.println ("The multiples of " + value + " between " +
                          value + " and " + limit + " (inclusive) are:");

      for (mult = value; mult <= limit; mult += value)
      {
         System.out.print (mult + "\t");

         // Print a specific number of values per line of output
         count++;
         if (count % PER_LINE == 0)
            System.out.println();
      }
   }
}
```

Sample Run

```
Enter a positive value: 7
Enter an upper limit: 400

The multiples of 7 between 7 and 400 (inclusive) are:
7       14      21      28      35
42      49      56      63      70
77      84      91      98      105
112     119     126     133     140
147     154     161     168     175
182     189     196     203     210
217     224     231     238     245
252     259     266     273     280
287     294     301     308     315
322     329     336     343     350
357     364     371     378     385
392     399
```

Exercise 1

- Exercise 1: Write a Java program to input an integer n and then display the integers from 1 to n and also their sum.
- Exercise1.java

Quick Check?

How many times the following loop will execute?

```
for (int counter = 0; counter <= 10; counter--)
{
}
```

The for Statement

- A `for` loop is functionally equivalent to the following `while` loop structure:

```
initialization;
while ( condition )
{
    statement;
    increment;
}
```

- Exercise 2: Convert the for loop in Exercise 1 to a while loop.
- Exercise2.java

Warnings

- Do not use a float or double for the counter
  - May result in imprecise counter values and faulty evaluation for loop termination purposes
- Do not use commas instead of semicolons to separate the components of the for loop
  - (very common error)
- As in the if and while, do not put a semicolon ; right after the parentheses

Nested Loops

Nested Loop means a loop within another loop.

For each iteration of the outer loop is executed, the inner loop is executed completely.

```
for (int i=1; i<=5; i+=2)
{
      for (int j=6; j>0; j-=3)
      {
         System.out.print("i= " + i + " j= " + j);
      }
      System.out.println();
}
```

A program that prints a triangle of stars

- See Stars.java

```
//********************************************************************
//  Stars.java       Author: Lewis/Loftus
//
//  Demonstrates the use of nested for loops.
//********************************************************************

public class Stars
{
   //-----------------------------------------------------------------
   //  Prints a triangle shape using asterisk (star) characters.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      final int MAX_ROWS = 10;

      for (int row = 1; row <= MAX_ROWS; row++)
      {
         for (int star = 1; star <= row; star++)
            System.out.print ("*");

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

```
//********************                    ******************************
//  Stars.java       Auth                 s
//
//  Demonstrates the use                  oops.
//********************                    ******************************

public class Stars
{
   //-----------------                     -----------------------------
   //  Prints a triangle                   erisk (star) characters.
   //-----------------                     -----------------------------
   public static void mai                 s)
   {
      final int MAX_ROWS

      for (int row = 1; row <= MAX_ROWS; row++)
      {
         for (int star = 1; star <= row; star++)
            System.out.print ("*");

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

Output

```
*
**
***
****
*****
******
*******
********
*********
**********
```

Nested loop verses single loop

- Do you really need a nested loop to print the triangle of stars in the output of stars.java?
- The answer is no, see single loop version Stars1.java.

- Exercise 3: Try out a diamond shape of stars yourselves!

Nested for Loop

```
                Multiplication Table
           1   2   3   4   5   6   7   8   9
------------------------------------------------
1  |     1   2   3   4   5   6   7   8   9
2  |     2   4   6   8  10  12  14  16  18
3  |     3   6   9  12  15  18  21  24  27
4  |     4   8  12  16  20  24  28  32  36
5  |     5  10  15  20  25  30  35  40  45
6  |     6  12  18  24  30  36  42  48  54
7  |     7  14  21  28  35  42  49  56  63
8  |     8  16  24  32  40  48  56  64  72
9  |     9  18  27  36  45  54  63  72  81
```

See MultiplicationTable.java

Outline

Boolean Expressions

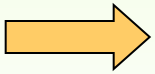
The `if` Statement

The Conditional Operator (`?` `:`)

The `switch` Statement

The `while` Statement

The `for` Statement



The `do` Statement

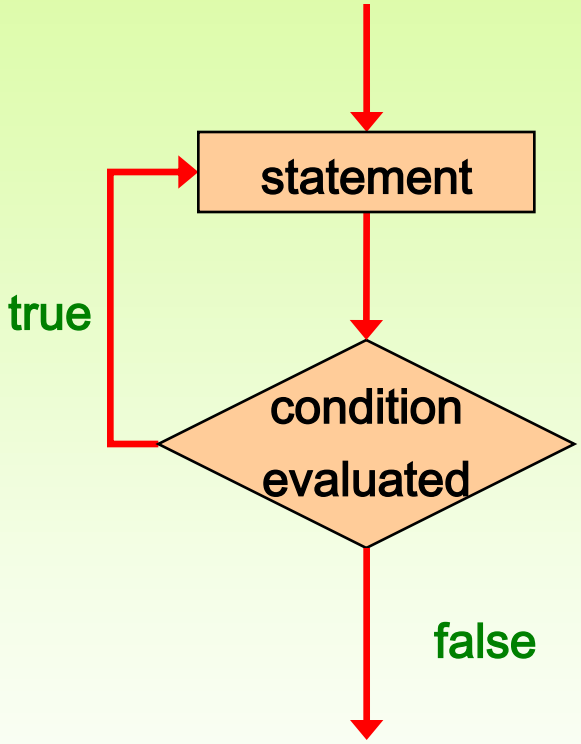
The do Statement

- A *do statement* has the following syntax:

```
do
{
    statement-list;
}
while (condition);
```

- The **`statement-list`** is executed once initially, and then the **`condition`** is evaluated
- The statement is executed repeatedly until the condition becomes false

Logic of a do Loop



The do Statement

- An example of a `do` loop:

```
int count = 0;
do
{
   count++;
   System.out.println (count);
} while (count < 5);
```

- The body of a **do while** loop executes at least once

- See **ReverseNumber.java**

```
//********************************************************************
//  ReverseNumber.java       Author: Lewis/Loftus
//
//  Demonstrates the use of a do loop.
//********************************************************************

import java.util.Scanner;

public class ReverseNumber
{
   //-----------------------------------------------------------------
   //  Reverses the digits of an integer mathematically.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      int number, lastDigit, reverse = 0;

      Scanner scan = new Scanner (System.in);
```

continue

continue

```
      do {
         System.out.print ("Enter a positive integer: ");
         number = scan.nextInt();
      } while (number<0);

      do {
         lastDigit = number % 10;
         reverse = (reverse * 10) + lastDigit;
         number = number / 10;
      } while (number != 0);

      System.out.println ("That number reversed is " + reverse);
   }
}
```

continue

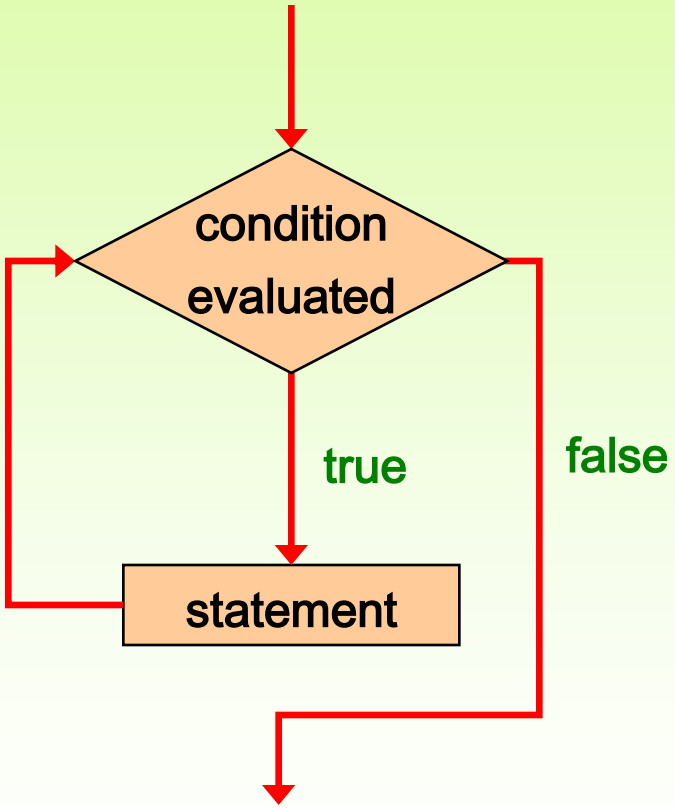
```
        System.out.
        number = sc

        do
        {
           lastDigit = number % 10;
           reverse = (reverse * 10) + lastDigit;
           number = number / 10;
        }
        while (number > 0);

        System.out.println ("That number reversed is " + reverse);
    }
}
```

Sample Run

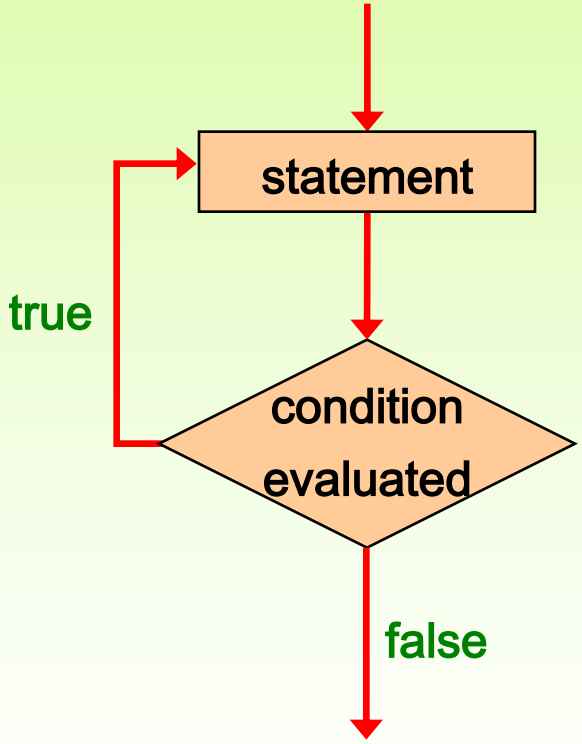
```
Enter a positive integer: 2896
That number reversed is 6982
```

Comparing while and do

The while Loop



The do Loop



Exercises

1. In a biology experiment a microorganism population doubles every 10 hours. Write a Java program to input the initial number of microorganisms and output how long (days and remaining hours) it will take to have more than 1000000 organisms.

2. Write a Java program to input the status (1- Full-time, 2-Part-time) and the salary of 10 instructors and output:
   - the number of full-time instructors.
   - the average salary of all instructors.

3. Write a Java program that produces the following output up to n^{th} term (where n is given by the user)

   1

   2 4

   3 6 9

   4 8 12 16

Solutions of Exercises

- ExerciseQ1.java
- ExerciseQ2.java
- ExerciseQ3.java