Outline

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

- **condition evaluated**
  - true
  - false

- **statement**
The while Statement

• An example of a while statement:

```java
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
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);
    }
}
System.out.println();
if (count == 0)
    System.out.println("No values were entered.");
else {
    average = (double)sum / count;
    DecimalFormat fmt = new DecimalFormat("0.###");
    System.out.println("The average is " + fmt.format(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*
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

```java
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);
```

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%

ratio = (double)won / NUM_GAMES;

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

```java
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](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:

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

- 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

for(init; condition; update)
{
    statements;
}

- **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.

```java
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

- **Initialization**
- **Condition evaluated**
  - True
    - Statement
    - Update
  - False
    - (No further actions taken)

Copyright © 2012 Pearson Education, Inc.
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

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

```java
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 \texttt{value} and \texttt{limit}, and then display the multiples of \texttt{value} from \texttt{value} to \texttt{limit}.

• See \texttt{Multiples.java}
import java.util.Scanner;

public class Multiples {
    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
    }
}

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();
}
}
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 + " ");
    // 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?

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

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

  ```java
  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.

```java
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](#)
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       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();
        }
    }
}
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

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>14</td>
<td>16</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>6</td>
<td>9</td>
<td>12</td>
<td>15</td>
<td>18</td>
<td>21</td>
<td>24</td>
<td>27</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>8</td>
<td>12</td>
<td>16</td>
<td>20</td>
<td>24</td>
<td>28</td>
<td>32</td>
<td>36</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>25</td>
<td>30</td>
<td>35</td>
<td>40</td>
<td>45</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>12</td>
<td>18</td>
<td>24</td>
<td>30</td>
<td>36</td>
<td>42</td>
<td>48</td>
<td>54</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>14</td>
<td>21</td>
<td>28</td>
<td>35</td>
<td>42</td>
<td>49</td>
<td>56</td>
<td>63</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>16</td>
<td>24</td>
<td>32</td>
<td>40</td>
<td>48</td>
<td>56</td>
<td>64</td>
<td>72</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>18</td>
<td>27</td>
<td>36</td>
<td>45</td>
<td>54</td>
<td>63</td>
<td>72</td>
<td>81</td>
</tr>
</tbody>
</table>

See [MultiplicationTable.java](#)
Outline

The while Statement

The for Statement

The do Statement
The do Statement

• A do statement has the following syntax:

```java
    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

true

condition evaluated

false

statement
The do Statement

• An example of a do loop:

```java
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`
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
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);
}
System.out.print("Enter a positive integer: ");
number = scan.nextInt();
do {
    lastDigit = number % 10;
    reverse = (reverse * 10) + lastDigit;
    number = number / 10;
} while (number > 0);
System.out.println("That number reversed is "+reverse);
Comparing while and do

The while Loop

- condition evaluated
  - true
  - statement
  - false

The do Loop

- statement
  - true
  - condition evaluated
  - false
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 1,000,000 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)

\[
\begin{array}{c}
1 \\
2 & 4 \\
3 & 6 & 9 \\
4 & 8 & 12 & 16 \\
\end{array}
\]
Solutions of Exercises

• ExerciseQ1.java
• ExerciseQ2.java
• ExerciseQ3.java