Re: Objectives in Selection Statements

° To declare boolean type and write Boolean expressions
° To use 6 comparison operators and 4 boolean operators
° To use Boolean expressions to control selection statements
° To implement selection control using if and nested if statements
° To implement selection control using switch statements
° To write expressions using the conditional operator
° To display formatted output using the System.out.printf method and to format strings using the String.format method
° To know the rules governing operator precedence and operand evaluation order
Objectives in Loops

- Sequence and selection aside, we need repetition (loops)
- To use while, do-while, and for loop statements to control the repetition of statements
- To understand the flow of control in loop statements
- To use Boolean expressions to control loop statements
- To write nested loops
- To know the similarities and differences of three types of loops
- To implement program control with break and continue

If you can write programs using loops, you know how to program!

while Loop and the Flow Chart

```java
while (loop-continuation-condition) {
    // loop-body;
    Statement(s);
}
```

[Diagram of the while loop flow chart with nodes and edges labeled for control flow.]
while Loop (cont.)

° In general, in order for a while loop to behave correctly, it is essential that the programmer verify that:
  • The loop-continuation-condition (Boolean condition) at the top of the loop can be evaluated before entering the loop for the first time
  • The loop modifies the value of the loop-continuation-condition (Boolean condition) so that the loop can potentially terminate

° Example: to sum a series of numbers from 1 to a specified limit (say 1,000,000)

Example: Summation from 1 to 1,000,000

```java
//import javax.swing.JOptionPane;

public class SummationTutor {
    public static void main(String[] args) {
        long sum = 0L;
        long index = 1L;

        while (index <= 1000000) {
            sum += index;
            index ++; // to terminate the loop!
        }

        //JOptionPane.showMessageDialog(null, "The sum is " + sum);
        System.out.printf("The sum is %d\n", sum);
    }
}
```

There is NO ;
Exercise

° How many times if the following loop body repeated? What is the output?

```java
int i = 1;
while (i > 10) {
    if (i % 2 == 0)
        System.out.println(i);
    i++;
}
```

What if we change ‘>’ to ‘<’ and trace it?

Example: An Advanced Math Learning Tool

The Math subtraction learning tool program generates just one question for each run. You can use a loop to generate questions repeatedly. This example gives a program that generates ten questions and reports the number of the correct answers after a student answers all ten questions. The program should also display the time spent on the test and lists all the questions.
Example: An Advanced Math Learning Tool

```java
int correctCount = 0;
int count = 0;
long startTime = System.currentTimeMillis();
String output = "";
while (count < 10) {
    // 1. Generate two random single-digit integers
    int number1 = (int)(Math.random() * 10);
    int number2 = (int)(Math.random() * 10);
    // 2. If number1 < number2, switch number1 with number2
    // 3. Prompt the student to answer "what is number1 – number2?"
    // 4. Grade the answer and display the result; and record the current question
    // 5. record the current question
    output += "\n" + number1 + ";" + number2 + "=" + answerString +
        ((number1 - number2 == answer) ? " correct" : " wrong");
    // 6. Increase the count
    count++;
}
long endTime = startTime = System.currentTimeMillis();
long testTime = endTime – startTime;
```

Subtraction Tutor Loop

---

Ending a `while` Loop with a Sentinel Value

What if the number of times a loop is executed is NOT predetermined?

We may use an input value to signify the end of the loop. Such a value is known as a **sentinel value**. Write a program that calculates the sum of an unspecified number of integers. The input 0 signifies the end of the input.

```java
import javax.swing.JOptionPane;
public class SentinelValue {
    public static void main(String[] args) {
        // Read an initial data
        String dataString = JOptionPane.showInputDialog("……");
        int data = Integer.parseInt(dataString);
        // Keep reading data until the input is 0
        int sum = 0;
        while (data != 0) {
            sum += data;
            // Read the next data
            dataString = JOptionPane.showInputDialog("……");
            data = Integer.parseInt(dataString);
        }
        JOptionPane.showMessageDialog(null, "The sum is " + sum);
    }
}
```
Caution

Don’t use floating-point values for equality checking in a loop control. Since floating-point values are approximations, using them could result in imprecise counter values and inaccurate results.

What will be the output of the following code segment?

```java
// data should be zero
double data = Math.pow(Math.sqrt(2), 2) - 2;
if (data == 0)
    System.out.println("data is zero");
else
    System.out.println("data is not zero");
```

do-while Loop and the Flow Chart

° The `do-while` loop is the only bottom-testing loop in Java

```plaintext
do {
    // Loop body;
    Statement(s);
} while (loop-continuation-condition);
```
Ending a do-while Loop with a Sentinel Value

What is the key difference between a while loop and a do-while loop?

What could be the benefit of using a do-while loop, instead of a while?

```java
import javax.swing.JOptionPane;
public class SentinelValue {
    public static void main(String[] args) {
        // Read an initial data
        String dataString = JOptionPane.showInputDialog(“"……"’);
        int data = Integer.parseInt(dataString);
        // Keep reading data until the input is 0
        int sum = 0;
        do {
            // Read the next data
            dataString = JOptionPane.showInputDialog(“"……"’);
            data = Integer.parseInt(dataString);
            sum += data;
        } while (data != 0);
        JOptionPane.showMessageDialog(null, “The sum is ” + sum);
    }
}
```

for Loops

```
for (initial-action; loop-continuation-condition; action-after-each-iteration) {
    // loop body;
    Statement(s);
}
```

(A) Initial-Action

(B) i = 0

LooP Continuation Condition

true

false

Statement(s) (loop body)

Action-After-Each-Iteration

true

false

int i;
for (i = 0; i < 100; i++) {
    System.out.println("Welcome to Java!");
}

(A) i = 0

(B) i++

true

false

System.out.println("Welcome to Java!");

true
Example: Summation Numbers from 1 to 1000000

```java
//import javax.swing.JOptionPane;

public class SummationTutor2 {
    public static void main(String[] args) {
        long sum = 0L;
        long index;
        for (index = 0L; index <= 1000000; index ++)
            sum += index;
        //JOptionPane.showMessageDialog(null, "The sum is \" + sum);
        System.out.printf("The sum is %d\n", sum);
    }
}
```

What you would do by the use of `while` loop?

What you would do if to sum EVEN numbers from 1 to 1000000?

---

Summation Even Numbers from 1 to 1000000

```java
//import javax.swing.JOptionPane;

public class SummationTutor3 {
    public static void main(String[] args) {
        long sum = 0L;
        long index;
        for (index = 0L; index <= 1000000; index += 2)
            sum += index;
        //JOptionPane.showMessageDialog(null, "The sum is \" + sum);
        System.out.printf("The sum is %d\n", sum);
    }
}
```
**Note 1**

The initial-action in a for loop can be a list of zero or more comma-separated expressions. The action-after-each-iteration in a for loop can be a list of zero or more comma-separated statements. Therefore, the following for loop is correct. It is rarely used in practice, however.

```java
for (int i = 0, j = 0; (i + j < 10); i++, j++) {
    // Do something
}
```

**Note 2**

If the loop-continuation-condition in a for loop is omitted, it is implicitly true. Thus the statement given below in (a), which is an infinite loop, is correct. Nevertheless, it is better to use the equivalent loop in (b) to avoid confusion:

(a) ```java
for (; ; ) {
    // Do something
}
```

(b) ```java
while (true) {
    // Do something
}
```
Which Loop to Use?

The three forms of loop statements, while, do-while, and for, are expressively equivalent; that is, you can write a loop in any of these three forms. For example, a while loop in (a) in the following figure can always be converted into the following for loop in (b):

(a)  

```java
while (loop-continuation-condition) {
    // Loop body
}
```

(b)  

```java
for ( ; loop-continuation-condition; ) {
    // Loop body
}
```

A for loop in (a) in the following figure can generally be converted into the following while loop in (b) except in certain special cases, such when `continue` is used (see Review Question 4.12 for one of them):

(a)  

```java
for (initial-action;
    loop-continuation-condition;
    action-after-each-iteration) {
    // Loop body
}
```

(b)  

```java
while (loop-continuation-condition) {
    // Loop body
    action-after-each-iteration;
}
```

Recommendations

Use the one that is most intuitive and comfortable for you.

In general, a for loop may be used if the number of repetitions is known, as, for example, when you need to print a message 100 times.

A while loop may be used if the number of repetitions is not known, as in the case of reading the numbers until the input is 0.

A do-while loop can be used to replace a while loop if the loop body has to be executed before testing the continuation condition.
Cautions

for (int i=0; i<10; i++;
{ System.out.println("i is " + i);
} 

What are errors?

Adding a semicolon at the end of the for and while clauses before the loop body is a common logic mistake.

Missing a semicolon at the end of do-while clause is also a common mistake, but syntax mistake.

int i=0;
while (i < 10);
{ System.out.println("i is " + i);
i++;
} 

Example: Minimizing Numerical Errors

Problem: Write a program that sums a series that starts with 0.01 and ends with 1.0. The numbers in the series will increment by 0.01, as follows: 0.01 + 0.02 + 0.03 and so on.

Use float type in a for loop:

float sum = 0;
for (float i = 0.01f; i <= 1.0; i = i + 0.01f)
sum += i;

° What if we use double type i in a for loop, better precision? Or not? Why?

° Two general way to minimize errors
  • process larger number first, how to rewrite the code?
  • use an integer count to ensure that all the numbers are processed, how to rewrite the code?
**Example: Finding the Greatest Common Divisor**

Problem: Write a program that prompts the user to enter two positive integers and find their greatest common divisor.

Thinking before coding!

How do we find the greatest common divisor of two numbers say \((n_1, n_2)\)?

What is a common divisor of two numbers?

1 is a common divisor, but may not be the greatest; a start point!

What could be the greatest divisor for \(n_1\), for \(n_2\), respectively?

Can the greatest common divisor be greater than or equal to \(n_1\) or \(n_2\)?

Then, translate the idea to the code!

```c
int gcd = 1;
int k = 1;
while ((k <= n1) && (k <= n2)) {
    if ((n1 % k ==0) && (n2 % k ==0))
        gcd = k;
    k++;
}
```

**Example: Finding the Greatest Common Divisor**

Translating the idea to the code!

```c
int gcd = 1;
int k = 1;

while ((k <= n1) && (k <= n2)) {
    if ((n1 % k ==0) && (n2 % k ==0))
        gcd = k;
    k++;
}
```
Example: Nested Loops for a Multiplication Table

Problem: Write a program that uses nested for loops to print a multiplication table.

Thinking before coding!

Once we have a logical solution, type the code to translate the solution into a Java program.

Exercise

° What is the output of the following program?

    public class Test{
        public static void main(String[] args) {
            for (int i = 1; i < 5; i++) {
                int j = 0;
                while (j < i) {
                    System.out.println(j + " ");
                    j++;
                }
            }
        }
    }
Example: Displaying a Pyramid of Numbers

Problem: Write a program that prompts the user to enter an integer from 1 to 15 and displays a pyramid. For example, if the input integer is 12, the output is shown below (each number occupies three spaces).

```
Example: Displaying a Pyramid of Numbers

for (int row = 1; row <= numberOfLines; row++) {
    // Print numberOfLines – row) leading spaces
    for (int column = 1; column <= numberOfLines - row; column++)
        System.out.print("   ");

    // Print leading numbers row, row – 1, ..., 1
    for (int num = row; num >= 1; num--)
        System.out.print((num >= 10) ? " " + num : "  " + num);

    // Print ending numbers 2, 3, ..., row – 1, row
    for (int num = 2; num <= row; num++)
        System.out.print((num >= 10) ? " " + num : "  " + num);

    // Start a new line
    System.out.println();
}
```

Example: Displaying a Pyramid of Numbers

for (int row = 1; row <= numberOfLines; row++) {
    // Print numberOfLines – row) leading spaces
    for (int column = 1; column <= numberOfLines - row; column++)
        System.out.print("   ");

    // Print leading numbers row, row – 1, ..., 1
    for (int num = row; num >= 1; num--)
        System.out.print((num >= 10) ? " " + num : "  " + num);

    // Print ending numbers 2, 3, ..., row – 1, row
    for (int num = 2; num <= row; num++)
        System.out.print((num >= 10) ? " " + num : "  " + num);

    // Start a new line
    System.out.println();
}
Using `break`

- Example: adds the integers from 1 to 20 to `sum` until `sum` is greater than or equal to 100.

- `break` immediately ends the *innermost loop* that contains it. It is often used with an *if* statement. It provides loops with additional control.

```java
public class TestBreak {
    /** Main method */
    public static void main(String[] args) {
        int sum = 0;
        int number = 0;
        while (number < 20) {
            number++;
            sum += number;
            if (sum >= 100) break;
        }
        System.out.println("The number is "+number);
        System.out.println("The sum is "+sum);
    }
}
```

Or, one can change the loop termination condition, how?

Using `continue`

- Example: adds the integers from 1 to 20 except 10 to `sum`.

- `continue` only ends the current iteration. Program control goes to the end of the loop body. It is often used with an *if* statement.

```java
public class TestBreak {
    /** Main method */
    public static void main(String[] args) {
        int sum = 0;
        int number = 0;
        while (number < 20) {
            number++;
            sum += number;
            if (number == 10) continue;
        }
        System.out.println("The number is "+number);
        System.out.println("The sum is "+sum);
    }
}
```
Example: Displaying Prime Numbers

Problem: Write a program that displays the first 50 prime numbers in five lines, each of which contains 10 numbers. An integer greater than 1 is prime if its only positive divisor is 1 or itself. For example, 2, 3, 5, and 7 are prime numbers, but 4, 6, 8, and 9 are not.

Solution: The problem can be broken into the following tasks:

- Determine whether a given number is prime.
- For number = 2, 3, 4, 5, 6, ..., test whether the number is prime.
- Count the prime numbers.
- Print each prime number, and print 10 numbers per line.

```java
boolean isPrime = true;
for (int divisor = 2; divisor <= number / 2; divisor++) {
    if (number % divisor == 0) {
        isPrime = false;
        break
    }
}
```

Debugging Loops using NetBeans

Programming errors often occur in loops. Debugger can help to locate the errors. You can examine the change of variables after each iteration by placing a breakpoint at the last statement inside the loop and executing the program using the Run to End of Method command.

Suppose you forgot to increment count in Line 27 in the preceding example, PrimeNumber.java. Let us trace the program using NetBeans, as follows:

1. Click the cutter of the Line 41 to set a breakpoint in the last statement inside the while loop.
2. Right-click PrimeNumber.java in the project pane to display a context menu, and choose Debug File – Step Out to start the debugger. The execution is paused at the breakpoint.
3. Add variables number, count, and isPrime in the watch view. Choose Step Over to observe the change of variables in the watch view, as shown in the next slide. You will see that count is not changed after each iteration.
Reading/Homework

- Review: Chapter 4 of the textbook
- Preview: Chapter 5 of the textbook for the next lecture
- Do review questions: 4.4, 4.6, 4.8, 4.10, 4.11, 4.17, 4.18