CSI1102 Introduction to Software Design

Chapter 3:

Program Statements

Learning objectives: Program Statements

- Understand the concepts of "flow of control" through a method
- Selection statements: `if, if-else` and `switch`
- Operators
  - Boolean operators: AND, OR, NOT
  - Other Java operators: increment ++, decrement --, assignment += and conditional ?
- Repetition statements: `while, do` and `for`
- Understand the important program development stages

2

What is the "Flow of Control"?

- Some programming statements modify the linear flow of control, allowing us to:
  - decide whether or not to execute a particular statement, or
  - perform a statement over and over, repetitively
- These decisions are based on a *Boolean expression* (also called a *condition*) that evaluates to true or false
- **The order of statement execution is called the *flow of control***

3

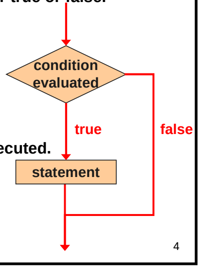
Selection Statement type 1: The IF Statement

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

if is a Java reserved word

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

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



4

The if Statement: An example Age.java

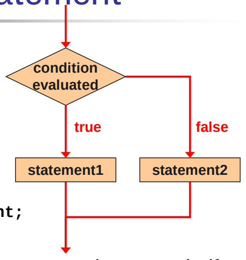
```
import cs1.Keyboard;

public class Age
{
// Reads your age from the keyboard and prints a comment
        public static void main (String[] args)
        {
                final int MINOR = 21;

                System.out.print("enter your age: ");
                int age = Keyboard.readInt();

                if (age < MINOR)
                        System.out.println("Enjoy life.");

                System.out.println("Age is a state of mind.");
        }
}
```

5

Selection Statement type 2: The if-else statement

```
if ( condition )
   statement1;
else
   statement2;
```

```
E.g.:
if (height <= MAX)
  adjustment = 0;
else
  adjustment = MAX – height;
```



- 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

6

Nested if Statements: MinOfThree.java

```
import cs1.Keyboard;

public class MinOfThree
{
// Read 3 integers from the screen and print the smallest

public static void main (String[] args)
      {
            int num1, num2, num3, min = 0;

            System.out.println("Enter three integers");
            num1 = Keyboard.readInt();
            num2 = Keyboard.readInt();
            num3 = Keyboard.readInt();
```

- Continued....

7

Nested if Statements: MinOfThree.java (cont)

```
            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);
      }
}
```

```
Enter three integers:
21341 3424 1233
Minimum value: 1233
```

8

Selection Statement type 3: 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

Important statements:
Break
Default

If *expression* matches *value2*, control jumps to here

9

The Switch Statement: GradeReport.java

```
import cs1.Keyboard;

public class GradeReport
{
   // Reads a grade and print a comment

   public static void main (String[] args)
   {
             int grade, category;

             System.out.println("Enter a numeric grade:");
             grade = Keyboard.readInt();

             category = grade/10;

             System.out.print ("That grade is ");
```

Continued....

10

The Switch Statement: GradeReport.java

```
      switch (category)
      {
            case 10:
                  System.out.println("a perfect score. Excellent.");
                  break;
            case 9:
                  System.out.println("well above average. Well done.");
                  break;
            case 8:
                  System.out.println("above average. Good!");
                  break;
            default:
                  System.out.println("not passing.");
      }
   }
}
```

```
Enter a numeric grade:
87
That grade is above average. Good!
```

11

Selection Statement type 3: The switch Statement

- The expression of a `switch` statement must result in an *integral type*, meaning an `int` or a `char`
- It **cannot be** a `boolean` value, a floating point value (`float` or `double`), a `byte`, a `short`, or a `long`
- The implicit boolean **condition** in a `switch` statement is equality - it tries to match the expression with a value
- **How can we implement the Switch in another way?**

12

What are Block Statements?

- Several statements can be grouped together into a *block statement*
- A block is delimited by braces : { … }
- A block statement can be used wherever a statement is called for by the Java syntax
- For example, in an `if-else` statement, the `if` the `else` portion, or both, could be block statements
- See Guessing.java (page 141)

13

What are Boolean expressions?

- 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

E.g. (age != 21)   (age >= 21)   (age == 21)

14

Logical Operators and Truth Tables

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

!	Logical NOT
&&	Logical AND
||	Logical OR

- They all take boolean operands and produce boolean results
  - Logical NOT is a unary operator (it operates on one operand)   e.g. !found
  - Logical AND and logical OR are binary operators (each operates on two operands)
    e.g. (Age != 60) && !found

15

Truth Tables

- A truth table shows the 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

- We can use truth tables for debugging!

16

More about Logical Operators

- Conditions can use logical operators to form complex expressions

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

- Logical operators have precedence relationships among themselves and with other operators
  - all logical operators have lower precedence than the relational or arithmetic operators
  - logical NOT has higher precedence than logical AND and logical OR

17

Beware: Short Circuited Operators

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

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

- **This type of processing must be used carefully: WHY????**

18

Comparing Characters

- We can use the relational operators on character data
- The results are based on the Unicode character set
- The following condition is true because the character + comes before the character J in the Unicode character set:

```
if ('+' < 'J')
   System.out.println ("+ is less than J");
```

- The uppercase alphabet (A-Z) followed by the lowercase alphabet (a-z) appear in alphabetical order in the Unicode character set

19

Comparing Strings

- Remember that a character string in Java is an object
  - We cannot use the relational operators to compare strings
- The `equals` method can be called with strings to determine if two strings contain exactly the same characters in the same order

  E.g. `(name1 == name2)`
- The `String` class also contains a method called `compareTo` to determine if one string comes before another

  E.g. `int result = name1.compareTo(name2);`

20

Comparing Strings: Lexicographic Ordering

- Because comparing characters and strings is based on a character set, it is called a *lexicographic ordering*
- This is not strictly alphabetical when uppercase and lowercase characters are mixed
  - For example, the string `"Great"` comes before the string `"fantastic"` because all of the **uppercase** letters come before all of the lowercase letters in Unicode
- Also, short strings come before longer strings with the same prefix (lexicographically)
- Therefore `"book"` comes before `"bookcase"`

21

Beware: Comparing Float Values

- You should **rarely use the equality operator** (==) when comparing two floats, rather determine if they are "close enough"
- Therefore, to determine the equality of two floats, you may want to use the following technique:

```
if (Math.abs(f1 - f2) < 0.00001)
   System.out.println ("Essentially equal.");
```

22

More Operators

- To round out our knowledge of Java operators, let's examine a few more
- In particular, we will examine
  - the increment (++) and decrement (--) operators
  - the assignment (+=) operators
  - the conditional (?) operator

`count++;` is functionally equivalent to

```
count = count + 1;
```

23

Increment and Decrement

- The increment and decrement operators can be applied in *prefix form* (before the operand) or *postfix form* (after the operand)
- When used alone in a statement, the prefix and postfix forms are functionally equivalent. That is,

```
count++;
```

is equivalent to

```
++count;
```

24

Increment and Decrement

- When used in a larger expression, the prefix and postfix forms have different effects
- In both cases the variable is incremented (decremented)
- But the value used in the larger expression depends on the form used:

Expression	Operation	Value Used in Expression
`count++`	**add 1**	**old value**
`++count`	**add 1**	**new value**
`count--`	**subtract 1**	**old value**
`--count`	**subtract** 1	**new value**

Increment and Decrement

- If `count` currently contains 45, then the statement

```
total = count++;
```

makes `total = ?` and `count = ?`

- If `count` currently contains 45, then the statement

```
total = ++count;
```

makes `total = ?` and `count = ?`

Assignment Operators

- Often we perform an operation on a variable, and then store the result back into that variable
- Java provides *assignment operators* to simplify that process
- For example, the statement

```
num += count;
```

is equivalent to

```
num = num + count;
```

Assignment Operators

- There are many assignment operators, including the following:

Operator	Example	Equivalent To
`+=`	`x += y`	`x = x + y`
`-=`	`x -= y`	`x = x - y`
`*=`	`x *= y`	`x = x * y`
`/=`	`x /= y`	`x = x / y`
`%=`	`x %= y`	`x = x % y`

Assignment Operators

- The right hand side of an assignment operator can be a complex expression
- The entire right-hand expression is evaluated first, then the result is combined with the original variable
- Therefore

```
result /= (total-MIN) % num;
```

is equivalent to

```
result = result / ((total-MIN) % num);
```

Assignment Operators

- The behavior of some assignment operators depends on the types of the operands
- If the operands to the += operator are strings, the assignment operator performs string concatenation
- The behavior of an assignment operator (+=) is always consistent with the behavior of the "regular" operator (+)
- **I do not recommend using this "shorthand" → It can lead to errors. Rather type the complete expression, except in Loops.**

The Conditional Operator

- Java has a *conditional operator* that evaluates a boolean condition that determines which of two other expressions is evaluated
- The result of the chosen expression is the result of the entire conditional operator
- Its syntax is:

```
condition ? expression1 : expression2
```

- If the `condition` is true, `expression1` is evaluated; if it is false, `expression2` is evaluated

The Conditional Operator

- The conditional operator is similar to an `if-else` statement, except that it forms an expression that returns a value
- For example:

```
larger = ((num1 > num2) ? num1 : num2);
```

- If `num1` is greater that `num2`, then `num1` is assigned to `larger`; otherwise, `num2` is assigned to `larger`
- The conditional operator is *ternary* because it requires three operands

The Conditional Operator

- Another example:

```
System.out.println ("Your change is " + count +
    ((count == 1) ? "Dime" : "Dimes"));
```

- If `count` equals 1, then `"Dime"` is printed
- If `count` is anything other than 1, then `"Dimes"` is printed

Repetition Statements

- Java has three kinds of repetition statements:
  - the *while loop*
  - the *do loop*
  - the *for loop*
- The programmer should choose the right kind of loop for the situation

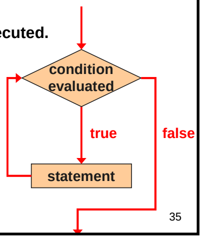
Repetition: The while Statement

`while` is a reserved word

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

If the `condition` is true, the `statement` is executed. Then the `condition` is evaluated again.

The `statement` is executed repeatedly until the `condition` becomes false.



The while Statement

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

The While Statement: Average.java

```
import java.text.DecimalFormat;
import cs1.Keyboard;

public class Average
{
   // Computes the average of a set of values
   public static void main (String[] args)
   {
         int sum = 0, value, count = 0;
         double average;

         DecimalFormat fmt = new DecimalFormat("0.###");

         System.out.print("Enter an integer (0 to quit) ");
         value = Keyboard.readInt();
```

Continued...

The While Statement: Average.java (cont)

```
         while (value != 0) // sentinal 0 terminates the loop
         {
                 count++;

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

                 System.out.print("Enter an integer (0 to quit) ");
                 value = Keyboard.readInt();
         }

         System.out.println("Number of values entered: " + count);
         average = (double) sum/count;
         System.out.println("Average number entered: " + average);
     }
}
```

Avoiding Infinite Loops

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

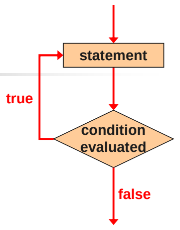
Avoiding infinite loops: Forever.java

```
public class Forever
   {
         // Computes the average of a set of values

         public static void main (String[] args)
         {
               int count = 1;

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

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

Nested Loops

- Similar to nested `if` statements, loops can be nested as well
- That is, the body of a loop can contain another loop
- Each time through the outer loop, the inner loop goes through its full set of iterations
- See PalindromeTester.java (page 167)

The do Statement



do and `while` are reserved words

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

The *`statement`* is executed once initially, and then the *`condition`* is evaluated

The *`statement`* is executed repeatedly until the *`condition`* becomes false

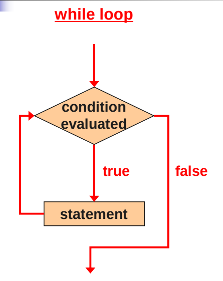
The do Statement

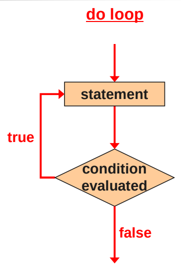
- A do loop is similar to a `while` loop, except that the condition is evaluated **after** the body of the loop is executed
- Therefore the body of a do loop will execute **at least once**
- What is printed if count = 0 and LIMIT =5?
- What is printed if count = 5 and LIMIT = 5?
- What is printed if count = 6 and LIMIT = 5?

```
do
{
    count = count + 1;
    system.out.println(count);
}
while (count < LIMIT);
```

```
 while (count < LIMIT)
{
   count = count + 1;
   system.out.println(count);
}
```

Comparing while and do





The for Statement

Reserved word

The ***initialization*** is executed once before the loop begins

The ***statement*** is executed until the ***condition*** becomes false

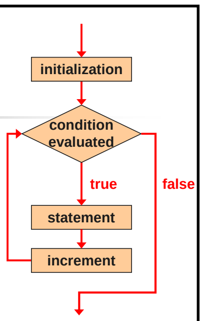
```
for ( initialization ; condition ; increment )
    statement;
```

The ***increment*** portion is executed at the end of each iteration
The ***condition-statement-increment*** cycle is executed repeatedly

The for Statement

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

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



- Like a `while` loop, the condition of a `for` statement is tested prior to executing the loop body
- Therefore, the body of a `for` loop will **execute zero or more times**

The for Statement: Stars.java

```
public class Stars
{
        // Print lines of stars, from 1 to 10

        public static void main (String[] args)
        {
                final int MAXROWS = 10;

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

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

The for Statement

- Each expression in the header of a for loop is optional
    - If the *`initialization`* is left out, no initialization is performed
    - If the *`condition`* is left out, it is always considered to be true, and therefore creates an infinite loop
    - If the *`increment`* is left out, no increment operation is performed
- Both semi-colons are always required in the `for` loop header

Choosing a Loop Structure: Some guidelines

- When you can't determine how many times you want to execute the loop body, use a `while` statement or a do statement
  - If it might be zero or more times, use a `while` statement
  - If it will be at least once, use a `do` statement
- If you can determine how many times you want to execute the loop body, use a `for` statement

49

A word about Program Development

- The creation of software involves four basic activities:
  - establishing the requirements
  - creating a design
  - implementing the code
  - testing the implementation

- The development process is much more involved than this, but these are the four basic development activities

50

Program Development

- Suppose you were given some initial requirements:
  - accept a series of test scores
  - compute the average test score
  - determine the highest and lowest test scores
  - display the average, highest, and lowest test scores
- Discuss how you would follow the program development steps to create a solution

51

Program Development: Requirement Analysis

- Clarify and flesh out specific requirements
  - How much data will there be?
  - How should data be accepted?
  - Is there a specific output format required?
- After conferring with the client, we determine:
  - the program must process an arbitrary number of test scores
  - the program should accept input interactively
  - the average should be presented to two decimal places
- **The process of requirements analysis may take a long time**

52

Program Development: Design

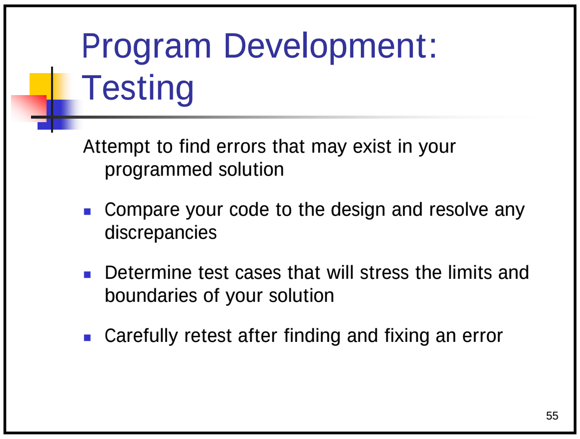
- Determine a possible general solution
  - Input strategy? (Sentinel value?)
  - Calculations needed?
- An initial algorithm might be expressed in pseudo-code
- Multiple versions of the solution might be needed to refine it
- Alternatives to the solution should be carefully considered

53

Program Development: Implementation

Translate the design into source code
- Make sure to follow coding and style guidelines
- Implementation should be integrated with compiling and testing your solution
- This process mirrors a more complex development model we'll eventually need to develop more complex software
- The result is a final implementation
- See the solution at ExamGrades.java (page 186)

54

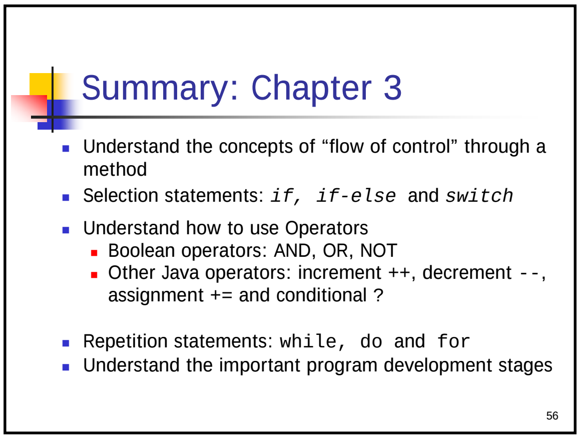


Program Development: Testing

Attempt to find errors that may exist in your programmed solution

- Compare your code to the design and resolve any discrepancies
- Determine test cases that will stress the limits and boundaries of your solution
- Carefully retest after finding and fixing an error

55



Summary: Chapter 3

- Understand the concepts of “flow of control” through a method
- Selection statements: *`if, if-else`* and *`switch`*
- Understand how to use Operators
  - Boolean operators: AND, OR, NOT
  - Other Java operators: increment ++, decrement `--`, assignment += and conditional ?
- Repetition statements: `while, do` and `for`
- Understand the important program development stages

56