Object Oriented Programming

Instructor: Asad Ullah Khan

Relational/Comparison Operators

- Comparison operators are used to compare two values (or variables).
- They help us to find answers and make decisions.

Operator	Result
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Boolean Logical Operators

- The Boolean logical operators shown here operate only on boolean operands.
- All of the binary logical operators combine two boolean values to form a resultant boolean value.

Operator	Result
&	Logical AND
|	Logical OR
^	Logical XOR (exclusive OR)
||	Short-circuit OR
&&	Short-circuit AND
!	Logical unary NOT
&=	AND assignment
|=	OR assignment
^=	XOR assignment
==	Equal to
!=	Not equal to
?:	Ternary if-then-else

The ? Operator

- Java includes a special ternary (three-way) operator that can replace certain types of if-then-else statements.
- This operator is the **?**.
- The **?** has this general form:

```
expression ? If true : If false
```

- `expression1` can be any expression that evaluates to a boolean value.
- If `expression1` is true, then `expression2` is evaluated; otherwise, `expression3` is evaluated.

```
ratio = denom == 0 ? 0 : num / denom;
```

Java Control Statements

- Control statements change the flow of execution
- Control statements can be put into three categories:
    1. Selection,
    2. Iteration, and
    3. Jumps

Selection Statements

- The if statement is Java's conditional branch statement.
- It can be used to route program execution through two different paths.

```
public class IfStatementExample {
  public static void main(String[] args) {
    int age = 25;
    // Check if age is greater than 18
    if (age > 18) {
      System.out.println("You are eligible to vote.");
    } else {
      System.out.println("You are not eligible to vote.");
    }
  }
}
```

If-Else-If Statements

- The if statements are executed from the top down.
- As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the ladder is bypassed.
- If none of the conditions is true, then the final **else** statement will be executed.

```
if(condition)
   statement;
else if(condition)
   statement;
else if(condition)
   statement;
...else
   statement;
```

Switch Statements

- The switch statement is Java's multiway branch statement.
- It provides an easy way to dispatch execution to different parts of your code based on the value of an expression.

```
switch (expression)
{case value1:
// statement sequence break;
case value2:
// statement sequence break;
...
case valueN :
// statement sequence break;
default:
// default statement
sequence}
```

Switch Statements

```
// A simple example of the switch.
class SampleSwitch {
  public static void main(String args[]) {
    for(int i=0; i<6; i++)
      switch(i) {
        case 0:
          System.out.println("i is zero.");
          break;
        case 1:
          System.out.println("i is one.");
          break;
        case 2:
          System.out.println("i is two.");
          break;
        case 3:
          System.out.println("i is three.");
          break;
        default:
          System.out.println("i is greater than 3.");
      }
  }
}
```

Iteration Statements

- Java’s iteration statements are
  1. ***for/ for-each,***
  2. ***while, and***
  3. ***do-while.***
- A loop repeatedly executes the same set of instructions until a termination condition is met.

While Loop

- The **while loop** is Java's most funda
  loop statement.
- It repeats a statement or block while it
  controlling expression is true.

```
while(condition)
  // body of loop
}
```

- The condition can be any Boolean expression.

```
// Demonstrate the while loop.
class While {
  public static void main(String args[]) {
    int n = 10;

    while(n > 0) {
      System.out.println("tick " + n);
      n--;
    }
  }
}
```

do-while Loop

- Sometimes it is desirable to e
  even if the conditional expres

```
do {
   // body o
} while(con
```

- The condition can be any Boo

```
// Demonstrate the do-while loop.
class DoWhile {
  public static void main(String args[]) {
    int n = 10;

    do {
      System.out.println("tick " + n);
      n--;
    } while(n > 0);
  }
}
```

for/ for-each Loop

- There are two forms o
- The first is the traditio
  version of Java.
- The second is the new

```
for (initia
  // body o
}
```

```
// Demonstrate the for loop.
class ForTick {
  public static void main(String args[]) {
    int n;

    for(n=10; n>0; n--)
      System.out.println("tick " + n);
  }
}
```

for/ for-each Loop

- A **for-each** style loop
- Such as an array, in
- The **for-each** style of

```
for (init
   // body
}
```

```
// Use a for-each style for loop.
class ForEach {
  public static void main(String args[]) {
    int nums[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
    int sum = 0;

    // use for-each style for to display and sum the values
    for(int x : nums) {
      System.out.println("Value is: " + x);
      sum += x;
    }

    System.out.println("Summation: " + sum);
  }
}
```

Arrays

- An array is a group of like-typed variables that are referred to by a common name.
- Arrays of any type can be created and may have one or more dimensions.
- A specific element in an array is accessed by its index.
- Arrays offer a convenient means of grouping related information.

```
type var-name[ ];
int month_days[];
```

Arrays

ty

in

- This declaration establis
- However, no array actu
- To link month_days with using ***new.***
- Assign it to ***month_da***
- **new** is a special opera

ar

```
// Demonstrate a one-dimensional array.
class Array {
  public static void main(String args[]) {
    int month_days[];
    month_days = new int[12];
    month_days[0] = 31;
    month_days[1] = 28;
    month_days[2] = 31;
    month_days[3] = 30;
    month_days[4] = 31;
    month_days[5] = 30;
    month_days[6] = 31;
    month_days[7] = 31;
    month_days[8] = 30;
    month_days[9] = 31;
    month_days[10] = 30;
    month_days[11] = 31;
    System.out.println("April has " + month_days[3] + " days.");
  }
}
```

Multidimensional Arrays

- Multidimensional arrays are implemented as arrays of arrays.
- To declare a multidimensional array variable, specify each additional index using another set of square brackets.

```
int twoD[][] = new int[4][5];
```

```
// Demonstrate a two-dimensional array.
class TwoDArray {
  public static void main(String args[]) {
    int twoD[][]= new int[4][5];
    int i, j, k = 0;

    for(i=0; i<4; i++)
      for(j=0; j<5; j++) {
        twoD[i][j] = k;
        k++;
      }

    for(i=0; i<4; i++) {
      for(j=0; j<5; j++)
        System.out.print(twoD[i][j] + " ");
      System.out.println();
    }
  }
}
```

Alternative Array Declaration Syntax

- There is a second form that may be used to declare an array:

```
type[ ] var-name;
```

- Here, the square brackets follow the type specifier, and not the name of the array variable.
- For example, the following two declarations are equivalent:

```
int al[ ] = new int[3];
int[ ] a2 = new int[3];
```

- The following declarations are also equivalent:

```
char twod1[ ][ ] = new char[3][4];
char[ ][ ] twod2 = new char[3][4];
```

Programming Exercises

1. Arrays:

- Write a program that reads 10 integers from the user and stores them in an array. Print the sum of all the elements in the array.
- Modify the previous program to find the largest and smallest element in the array.
- Write a program that reverses the elements of an array.

2. 2D Arrays:

- Write a program that reads a matrix of numbers from the user and prints its transpose (rows become columns and vice versa).

3. Multidimensional Arrays:

- Create a 3D array to store the grades of students in different subjects and semesters.
- Write functions to calculate the average grade of a particular student or the average grade for a specific subject in a semester.

Programming Exercises

4. var keyword:

- Explain the difference between using the var keyword and explicitly declaring the data type of a variable in a for-each loop.
- Rewrite the previous examples using the var keyword where applicable and discuss any potential advantages or disadvantages.

5. for Loop vs. for-each Loop:

- Explain the scenarios where a for loop might be preferable compared to a for-each loop and vice versa.
- Rewrite the program examples using both for and for-each loops to iterate over arrays and analyze their readability and efficiency.

Bonus:

- Implement a simple sorting algorithm (e.g., bubble sort, selection sort) for both one-dimensional and two-dimensional arrays.
- Combine the concepts of arrays and loops to create a program that plays a simple guessing game (e.g., number guessing, word guessing).