University of Kelaniya

Kesavan Selvarajah

BSc (UOK), MSc (Reading, UCSC)

Temporary Lecturer

Software Engineering Department

Faculty of Computing and Technology

skesa231@kln.ac.lk

Object-Oriented Programming

CSCI 21052 / ETEC 21062

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.
- To create an array, you first must create an array variable of the desired type.
  - `int month_days[];`

Arrays

- Although the above declaration establishes the fact that **month_days** is an array variable, no array actually exists.
- To link **month_days** with an actual, physical array of integers, you must allocate one using new and assign it to **month_days**.
- **new** is a special operator that allocates memory.
  - `month_days = new int[12];`
- To add an element to the array you need to specify the index.
  - `Month_days[0] = 31;`
- You can also do the declaration and initialization of the array in one line like this,
  - `month_days[] = new int[12];`

Array

```
public class ShortArrayExample {
    public static void main(String[] args) {
        // Declare and initialize a short array
        short[] temperatures = { 22, 24, 23, 21, 20 };

        // Access and print values from the array
        System.out.println("Temperature at 2nd hour: " + temperatures[1]);

        // Calculate and print the length of the array
        int length = temperatures.length;
        System.out.println("Number of temperature readings: " + length);

        // Update an element in the array
        temperatures[0] = 25;
        System.out.println("Updated temperature at 1st hour:" + temperatures[0]);
    }
}
```

The if Statement

- We use an if statement to specify which block of code to execute.
- A block of code may contain zero or more statements.
- Which block is executed depends on the result of evaluating a test condition, called a **boolean expression**.
- The **boolean expression** is a conditional expression that is evaluated to either true or false.
- The six **relational operators** we can use in conditional expressions are:

<	- less than	**!=**	- not equal to
<=	- less than or equal to	**>**	- greater than
==	- equal to	**>=**	- greater than or equal to

Control statements

- Statements in programs are executed in sequence, which is called **sequential execution** or **sequential control flow**.
- However, we can add decision-making statements to a program to alter this control flow.
- The statement that alters the control flow is called a **control statement**.
- There are two main types of control statements:
    - **Selection statements**
        - The if statement
        - The switch statement
    - **Repetition statements**
        - The while statement
        - The do-while statement
        - The for statement

The if Statement syntax

```
if (boolean expression) {
    //then block
} else {
    //else block
}
```

```
if (number >= 100) {
    System.out.println("Number is greater than or equal 100.");
} else {
    System.out.println("Number is lesser than 100.");
}
```

The if Statement

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter test score: ");
        int testScore = scanner.nextInt();
        if (testScore < 70) {
            System.out.println("You did not pass");
        } else {
            System.out.println("You did pass");
        }
    }
}
```

Nested if Statements

- The **then** and **else** blocks of an if statement can contain any statement including another if statement.
- An if statement that contains another if statement in either its then or else block is called a **nested if** statement.

```
if (boolean expression) {
    //then block
    if (boolean expression) {
    } else {
        //else block
    }
} else {
    //else block
}
```

Nested if Statements #1

```
....
        System.out.print("Enter test score: ");
        int testScore = scanner.nextInt();
        System.out.print("Enter your age: ");
        int studentAge = scanner.nextInt();
        if (testScore >= 70) {
            if (studentAge < 10) {
                System.out.println("You did a great job");
            } else {
                System.out.println("You did pass");
            }
        } else {
            System.out.println("You did not pass");
        }
....
```

Nested if Statements #2

```
....
        System.out.print("Enter test score: ");
        int testScore = scanner.nextInt();
        System.out.print("Enter your age: ");
        int studentAge = scanner.nextInt();
        if (testScore >= 70 && studentAge < 10) {
            System.out.println("You did a great job");
        } else {
            if (testScore >= 70) {
                System.out.println("You did pass");
            } else {
                System.out.println("You did not pass");
            }
        }
....
```

Increment & decrement operators

- The double plus operator (++) is called the **increment operator**, which increments the variable by 1.
  - **count = count + 1;**

    Can be written as,
  - **count++;**
- The double minus operator (--) is the **decrement operator**, which decrements the variable by 1.
  - **count = count - 1;**

    Can be written as,
  - **count- -;**

Boolean Expressions & Variables

- A **boolean operator**, also called a logical operator, takes boolean values as its operands and returns a boolean value.
- There are three boolean operators:
  - **AND - &&**
    - The AND operation results in true only if both operands are true.
  - **OR - ||**
    - The OR operation results in true if atleast one of the operand is true.
  - **NOT - !**
    - The NOT operation is true if operand is false and is false if operand is true.

Boolean Expressions & Variables

P	Q	P && Q	P || Q	!P
false	false	false	false	true
false	true	false	true	true
true	false	false	true	false
true	true	true	true	false

Quick question

- Write a Java program that accepts marks as input and prints the corresponding grade according to the chart below, using if statements.

80 - 100	: A
70 - 80	: B
60 - 70	: C
50 - 60	: D
40 - 50	: E
00 - 40	: F

Answer

```
import java.util.Scanner;

public class GradeCalculator {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the marks: ");
        int marks = scanner.nextInt();
        scanner.close();

        char grade;

…
```

Answer

```
•••       if (marks >= 80 && marks <= 100) {
              grade = 'A';
          } else if (marks >= 70 && marks < 80) {
              grade = 'B';
          } else if (marks >= 60 && marks < 70) {
              grade = 'C';
          } else if (marks >= 50 && marks < 60) {
              grade = 'D';
          } else if (marks >= 40 && marks < 50) {
              grade = 'E';
          } else {
              grade = 'F';
          }
          System.out.println("Grade: " + grade);
      }
  }
```

The Switch Statement

- Another Java statement that implements a selection control flow is the **switch statement**.
- The syntax for the switch statement is,

```
switch (integer expression) {
        case 1:  case body
        case 2:  case
        ....
        case n:  case body
}
```

The Switch Statement

```
import java.util.Scanner;

public class SimpleSwitchCaseExample {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter a number (1-3): ");
        int number = scanner.nextInt();
        scanner.close();

...
```

The Switch Statement

```
•••      switch (number) {
             case 1:
                 System.out.println("You entered number one.");
                 break;
             case 2:
                 System.out.println("You entered number two.");
                 break;
             case 3:
                 System.out.println("You entered number three.");
                 break;
             default:
                 System.out.println("Invalid number.");
                 break;
         }
     }
 }
```

Quick question

- Write a Java program that uses a switch statement to determine the day of the week based on a given integer input. The program should print the corresponding day of the week for valid inputs from 1 to 7. If the input is not within this range, the program should print "Invalid day".

  Example:

  For an input of 3, the program should print "Wednesday".

  For an input of 6, the program should print "Saturday".

  For an input of 9, the program should print "Invalid day".

Answer

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number (1-7): ");
        int input = scanner.nextInt();
        scanner.close();

        String day;
...
```

Answer

```
          switch (input) {
              case 1:
                  day = "Sunday";
                  break;
              case 2:
                  day = "Monday";
                  break;
              case 3:
                  day = "Tuesday";
                  break;
              case 4:
                  day = "Wednesday";
                  break;
...
```

Answer

```
            case 5:
                day = "Thursday";
                break;
            case 6:
                day = "Friday";
                break;
            case 7:
                day = "Saturday";
                break;
            default:
                day = "Invalid day";
                break;
        }

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

Repetition statements

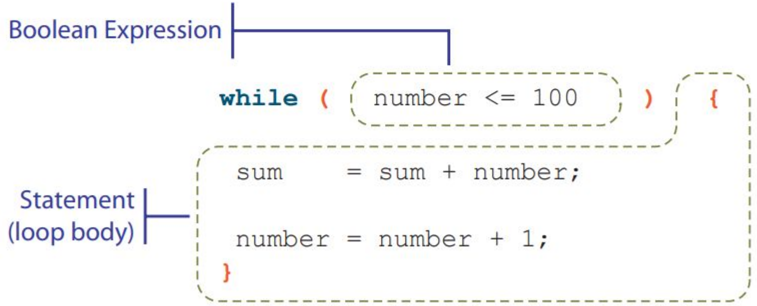
- **Repetition statements** control a block of code to be executed for a fixed number of times or until a certain condition is met.
- Java’s three repetition statements:
    - While statements
    - Do–while statements
    - For statements
- Instead of a repetition statement, a recursive method can also be used to program the repetition control flow.
- A **recursive method** is a method that calls itself.

The while Statement

- The **while loop,** loops through a block of code as long as a specified condition is true.

```
while(<boolean expression>){
    <statement>
}
```

- As long as the <boolean expression> is true, the <statement> inside the loop body is executed.



```
while ( number <= 100 ) {
    sum    = sum + number;

    number = number + 1;
}
```

The while Statement #1

```
public class Main {
    public static void main(String[] args) {
        int sum = 0, number = 1;
        while (number <= 100) {
            sum = sum + number;
            System.out.println(number);
            number = number + 1;
        }
    }
}
```

The while Statement #2

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        int sum = 0;
        int number;
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter an integer: ");
        number = scanner.nextInt();
        while (number >= 0) {
            sum = sum + number;
            System.out.print("Enter integer ");
            number = scanner.nextInt();
        }
    }
}
```

The while Statement

- A **count-controlled loop** is used when the number of iterations to occur is already known. (Eg #1)
- A counter controlled loop is also known as definite repetition loop, since the number of iterations is known before the loop begins to execute.
- A **sentinel loop** continues to process data until reaching a special value called sentinel that signals the end. (Eg #2)
- A sentinel controlled loop is also called an indefinite repetition loop because the number of iterations is not known before the loop starts executing.

Quick questions

1. Write a while statement to add numbers 11 through 20. Is this a count controlled or sentinel-controlled loop?
2. Write a while statement to read in real numbers and stop when a negative number is entered. Is this a count-controlled or sentinel-controlled loop?

The while Statement

- No matter what you do with the while statement (and other repetition statements), make sure that the loop will eventually terminate. Watch out for an **infinite loop** such as this one:

```
public class Main {
    public static void main(String[] args) {
        int product = 0;
        while (product < 500000) {
            product = product * 5;
            System.out.println(product);
        }
    }
}
```

The while Statement

- Another thing to watch out for in writing a loop is the **off-by-1 error (OBOE)**. Suppose we want to execute the loop body 10 times. Does the following code work?

```
public class Main {
    public static void main(String[] args) {
        int count = 1;
        while (count < 10 ) {
            System.out.println(count);
            count++;
        }
    }
}
```

The do-while Statement

- The while statement is characterized as a **pretest loop** because the test is done before execution of the loop body.
- Because it is a pretest loop, the loop body may not be executed at all.
- The do–while is a repetition statement that is characterized as a **posttest loop**.
- With a posttest loop statement, the loop body is executed at least once.

```
do{
    <statement>
}while(<boolean expression>);
```

The do-while Statement #1

```
public class Main {
    public static void main(String[] args) {
        int sum = 0, number = 1;
        do {
            sum += number;
            System.out.println(sum);
            number++;
        } while ( sum <= 1000000 );
    }
}
```

The do-while Statement #2

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        int age;
        Scanner scanner = new Scanner(System.in);
        do {
            System.out.print("Your Age (between 0 and 130): ");
            age = scanner.nextInt();
            if (age < 0 || age > 130) {
                System.out.println("Invalid age");
            }
        } while (age < 0 || age > 130);
    }
}
```

Quick questions

1. Write a do–while loop to compute the sum of the first 30 positive odd integers.
2. Rewrite the following while loops as do–while loops.

a.

```
int count = 0, sum = 0;
while ( count < 10 ) {
    sum += count;
    count++;
}
```

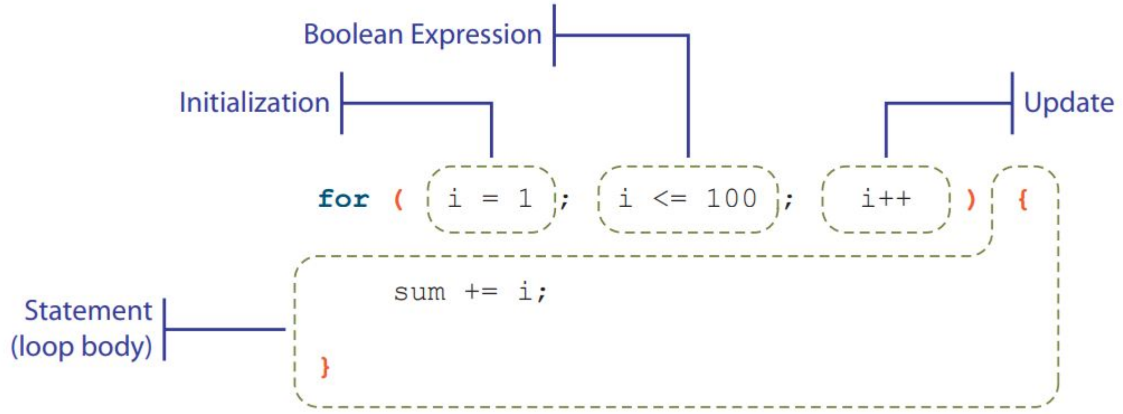
b.

```
int count = 1, sum = 0;
while ( count <= 30 ) {
    sum += count;
    count += 3;
}
```

The for Statement

- The **for statement** is the third repetition control statement and is especially suitable for count-controlled loops.
- The general format of the for statement is,

```
for(<initialization>; <boolean expression>; <update>){
    <statement>
}
```



```
for ( i = 1 ; i <= 100 ; i++ ) {
      sum += i;
}
```

The for Statement #1

```
public class Main {
    public static void main(String[] args) {
        for (int i = 0; i < 6; i++) {
            System.out.println(i);
        }
    }
}
```

The for Statement #2

```
class Main {
    public static void main(String[] args) {
        int sum = 0;
        int n = 100;
        for (int i = 1; i <= n; ++i) {
            sum += i;
        }
        System.out.println("Sum = " + sum);
    }
}
```

Quick questions

1. Write a for loop to compute the following.

    a. Sum of 1, 2, . . . , 100

    b. Sum of 2, 4, . . . , 500

    c. Product of 5, 10, . . . , 50

The nested for Statement #1

```
public class Main {
    public static void main(String[] args) {
        for (int i = 1; i <= 5; i++) {
            for (int j = 1; j <= i; j++) {
                System.out.print("* ");
            }
            System.out.println();//new line
        }
    }
}
```

The nested for Statement #2

```
public class Main {
    public static void main(String[] args) {
        int term = 6;
        for (int i = 1; i <= term; i++) {
            for (int j = term; j >= i; j--) {
                System.out.print("* ");
            }
            System.out.println();
        }
    }
}
```

Recursive Methods

- In addition to the three repetition control statements, there is a fourth way to control the repetition flow of a program by using recursive methods.
- A **recursive method** is a method that contains a statement (or statements) that makes a call to itself.

Recursive Methods #1

```
public class Main {
    public static void main(String[] args) {
        int number = 5;
        int sum = calculateSum(number);
        System.out.println("Sum of positive integers up to " +
number + " is: " + sum);
    }

    public static int calculateSum(int n) {
        if (n == 1) {
            return 1;
        } else {
            return n + calculateSum(n - 1);
        }
    }
}
```

Sample Java Standard Classes

- **String (Cont.)**
  - String class provides a variety of constructors.
  - We can create a String initialized by an array of characters using the following constructor,

```
char country[] = {S,r,i, ,L,a,n,k,a};
String srilanka= new String(country);
```

  - We can specify a subrange of a character array as an initializer, using the following constructor,

```
char country[] = {S,r,i, ,L,a,n,k,a};
String srilanka= new String(country,4,5);
```

Sample Java Standard Classes

- **StringBuffer**
  - As you know, String represents fixed-length, immutable character sequences, In contrast, StringBuffer represents growable and writable character sequences.
  - StringBuffer may have characters and substrings inserted in the middle or appended to the end.
  - StringBuffer will automatically grow to make room for such additions and often has more characters preallocated than are actually needed, to allow room for growth.

```
StringBuffer sb = new StringBuffer("Sri");
String message = sb.toString();
System.out.println(message);
sb.append("Lanka");
String message = sb.toString();
System.out.println(message);
```

Sample Java Standard Classes

- **StringTokenizer**
  - The processing of text often consists of parsing a formatted input string.
  - Parsing is the division of text into a set of discrete parts, or tokens, which in a certain sequence can convey a semantic meaning.
  - The StringTokenizer class provides the first step in this parsing process,

```
String text = "C,Java,Python";
StringTokenizer st = new StringTokenizer(text, ",");
while (st.hasMoreTokens())
    System.out.println(st.nextToken());
```

Thank you