CS6308- Java Programming

V P Jayachitra
Assistant Professor
Department of Computer Technology
MIT Campus
Anna University

Java's fundamental elements

- Data types
- Variables
- Arrays

Data types

- **No automatic coercions or conversions of conflicting types.**
- **Primitive data type**
- **Non primitive type**

Primitive data type

- Java defines eight primitive types of data: **byte, short, int, long, char, float, double, and boolean**

type	size	range
byte	8 bits	-128 to 127
short	16 bits	-32,768 to 32,767
int	32 bits	-2,147,483,648 to 2,147,483,647
long	64 bits	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
double	64 bits	-4.9e-324 to 1.8e+308
float	32 bits	-1.4e-045 to 3.4e+038
char	16 bits	0 to 65,536
boolean	1 bit or byte *	true or false

Integer literals

- **When a literal value is assigned to a byte or short variable, no error is generated if the literal value is within the range of the target type.**
  - **An integer literal can always be assigned to a long variable.**
  - **long literal ?**
    - **you will need to explicitly tell the compiler that the literal value is of type long.**
    - **Do this by appending an upper- or lowercase L to the literal.**
      - **For example, 0x7fffffffffffffL or 9223372036854775807L is the largest long.**
  - **An integer can also be assigned to a char as long as it is within range.**
  - **Decimal numbers cannot have a leading zero. Therefore signify a hexadecimal constant with a leading zero-x, (0x or 0X).**
    - **int x = 0b1010; //0B or 0b for binary**
  - **embed one or more underscores in an integer literal. cannot come at the beginning or the end of a literal**
    - **int x = 123_456_789; // x will be 123456789. The underscores will be ignored.**
    - **int x = 123___456___789; // x will be 123456789. The underscores will be ignored.**
    - **int x = 0b1101_0101_0001_1010; // x will be 54554**

Float literal

- **a float literal, must append an *F* or *f* to the constant.**
- **A double literal, must append a *D* or *d to the constant.***
- ***When the literal is compiled, the underscores are discarded.***
- **double num = 9_423_497.1_0_9; // 9423497.109.**
- **double num = 9_423_497_862.0;**

Boolean Literals

- **Two logical values that a boolean value can have, true and false.**
- **The values of true and false do not convert into any numerical representation.**
- **The true literal in Java does not equal 1, nor does the false literal equal 0.**

Character Literals

- A literal character is represented inside a pair of single quotes.
- All of the visible ASCII characters can be directly entered inside the quotes, such as '*a*', '*z*', and '@'.
- For characters that are impossible to enter directly, use escape sequences that allow you to enter the character
  - ' \' ' for the single-quote character itself and ' \n' for the newline character.
- For octal notation, use the backslash followed by the three-digit number.
  - For example, ' *141*' is the letter '*a*'.
- enter a backslash-u (\u), then exactly four hexadecimal digits.
  - For hexadecimal example, ' *u0061*' is the ISO-Latin-1 '*a*'
  - ' *ua432* ' is a Japanese Katakana character.

Escape characters	description
\ddd	octal
\uxxxx	Unicode
'	single quote
"	double quote
\	backslash
\r	Carriage return
\n	New line
\f	Form feed
\t	Tab
\b	backspace

String Literals

- **String literals in Java are specified by enclosing a sequence of characters between a pair of double quotes.**
- **Examples of string literals are**

```
"Hello World"
"two\nlines"
" \"This is in quotes\""
```

Variables

- **The variable is the basic unit of storage in a Java program.**
- **A variable is defined by the combination of an identifier, a type, and an optional initializer.**
- **In addition, all variables have a scope, which defines their visibility, and a lifetime.**

Declaring a Variable

- **In Java, all variables must be declared before they can be used. The basic form of a variable declaration is shown here:**

type identifier [= *value*][, *identifier* [= *value*] …];

Variables contd...

Variable declarations of various types.

```
public class VariableDeclarations {
    public static void main(String[] args) {
        // Declare and initialize variables in a single line
        int a, b, c;                // Multiple declarations for int
        int d = 1, e, f = 10;       // Initialize d and f, declare e
        byte g = 12;                // Initialize byte variable g
        double pi = 3.14159;        // Initialize double variable pi
        char x = 'x';               // Initialize char variable x
        boolean done = false;       // Initialize boolean variable done
        float h = 10.0f;            // Initialize float variable h (include 'f' suffix)
        long l = 9876543210L;       // Initialize long variable l (include 'L' suffix)
        // Print the variables to verify
        System.out.println("int a: " + a);
        System.out.println("int b: " + b);
        System.out.println("int c: " + c);
        System.out.println("int d: " + d);
        System.out.println("int e: " + e);
        System.out.println("int f: " + f);
        System.out.println("byte g: " + g);
        System.out.println("double pi: " + pi);
        System.out.println("char x: " + x);
        System.out.println("boolean done: " + done);
        System.out.println("float h: " + h);
        System.out.println("long l: " + l);
    }
}
```

Variables contd...

Dynamic Initialization

```
import java.util.Scanner;

public class CircleMath {
    public static void main(String[] args) {
        // Create a Scanner object for user input
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the radius of the circle: ");
        double radius = scanner.nextDouble();  // Read user input
        double area = Math.PI * radius * radius;
        double circumference = 2 * Math.PI * radius;
        System.out.printf("For a circle with radius %.2f:\n", radius);
        System.out.printf("Area: %.2f\n", area);
        System.out.printf("Circumference: %.2f\n", circumference);
        }
}
```

```
Enter the radius of the circle: 5
For a circle with radius 5.00:
Area: 78.54
Circumference: 31.42
```

Scope:Block Scope

```
class ScopeExample {
 public static void main(String[] args[]) {
      int alpha;  // known to all code within main
      alpha = 15;
      if (alpha == 15) {  // start new scope
            int beta = 25;  // known only to this block
             // alpha and beta both known here
              System.out.println("alpha and beta: " + alpha + " " + beta);
              alpha = beta * 2;
      }
  // beta = 100;
 // Error! beta not known here
// alpha is still known here
System.out.println("alpha is " + alpha);
  }
}
```

```
alpha is 15
```

Scope

```
class VariableLifetime {
  public static void main(String[] args[]) {
     int i;
     for (i = 0; i < 3; i++) {
               int j = -5;  // j is initialized each time block is entered
               System.out.println("j is: " + j);
               // this always prints -5
               j = 200;
                System.out.println("j is now: " + j);
     }
 }
}
```

```
j is now: 200
j is now: 200
j is now: 200
```

Scope

- Although blocks can be nested, you cannot declare a variable to have the same name as one in an outer scope.

```
// This program will not compile
class ScopeError {
 public static void main(String args[]) {
       int foo = 1;
  {            // creates a new scope
         int foo = 2;  // Compile-time error - foo already defined!
  }
}
}
```

```
java: variable foo is already defined in method main(java.lang.String[])
```

Type Conversion and Casting

- **If the two types are compatible, then Java will perform the conversion automatically.**
  - **For example, it is always possible to assign an int value to a long variable.**
  - **For instance, there is no automatic conversion defined from double to byte.**
    - **use a *cast*, which performs an explicit conversion between incompatible types.**
- **Java's Automatic Conversions: no explicit cast statement is required.**
  - **The two types are compatible.**
  - **The destination type is larger than the source type.**
  - **no automatic conversions from the numeric types to char or boolean.**
  - **performs an automatic type conversion when storing a literal integer constant into variables of type byte, short, long, or char.**

Casting Incompatible Types

- what if you want to assign an int value to a byte variable?
- **This conversion will not be performed automatically, because a byte is smaller than an int.**
- **This kind of conversion is sometimes called a narrowing conversion**
  - **explicitly making the value narrower so that it will fit into the target type.**
- **To create a conversion between two incompatible types, you must use a cast.**
- **A cast is simply an explicit type conversion. It has this general form:**
  - **(target-type) value**

```
int a;
byte b;
// …
b = (byte) a;
```

Casting Incompatible Types

```
class TypeConversion
public static void main(String args[]) {
        byte a;
        int m = 257;
        double n = 323.142;

        System.out.println("\nConversion of int to byte.");
        a = (byte) m;
        System.out.println("m and a " + m + " " + a);

        System.out.println("\nConversion of double to int.");
        m = (int) n;
        System.out.println("n and m " + n + " " + m);

        System.out.println("\nConversion of double to byte.");
        a = (byte) n;
        System.out.println("n and a " + n + " " + a);
    }
```

```
Conversion of int to byte.
m and a 257 1

Conversion of double to int.
n and m 323.142 323

Conversion of double to byte.
n and a 323.142 67
```

- **Automatic Type Promotion in Expressions**

```
238.14 + 515 - 126.3616
result = 626.7784146484375
```

```
class TypePromotion {
    public static void main(String args[]) {
            byte p = 42;
            char q = 'a';
            short r = 1024;
            int s = 50000;
            float t = 5.67f;
            double u = .1234;
            double result = (t * p) + (s / q) - (u * r);
            System.out.println((t * p) + " + " + (s / q) + " - " + (u * r));
            System.out.println("result = " + result); }
}
```

- **Type Promotion Rules**
  - First, all byte, short, and char values are promoted to int, as just described.
  - Then, if one operand is a long, the whole expression is promoted to long.
  - If one operand is a float, the entire expression is promoted to float.
  - If any of the operands are double, the result is double.

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.**

One-Dimensional Arrays

```
type var-name[ ];
array-var = new type [size];
```

```
int month_days[];
month_days = new int[12];
```

```
int month_days[] = new int[12];
```

```
public class ComputeAverage {
    public static void main(String[] args) {
        double[] values = {10.1, 11.2, 12.3, 13.4, 14.5};
        double sum = 0;
        int count;
        for (count = 0; count < 5; count++)
            sum = sum + values[count];
        System.out.println("Average is " + sum / 5);
    }
}
```

```
Average is 12.3
```

```
public class MonthDaysArray {
    public static void main(String[] args) {
        int[] daysInMonth = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
        System.out.println("April has " + daysInMonth[3] + " days.");
    }
}
```

```
April has 30 days.
```

Multidimensional Arrays

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

```
public class TwoDimensionalArrayDemo {
    public static void main(String[] args) {
        int[][] matrix = new int[4][5];
        int row, col, value = 0;

        for (row = 0; row < 4; row++)
            for (col = 0; col < 5; col++) {
                matrix[row][col] = value;
                value++;
            }

        for (row = 0; row < 4; row++) {
            for (col = 0; col < 5; col++)
                System.out.print(matrix[row][col] + " ");
            System.out.println();
        }
    }
}
```

```
0 1 2 3 4
5 6 7 8 9
10 11 12 13 14
15 16 17 18 19
```

```
class IrregularArray {
    public static void main(String[] args[]) {
        int irregular[][] = new int[4][];
        irregular[0] = new int[1];
        irregular[1] = new int[2];
        irregular[2] = new int[3];
        irregular[3] = new int[4];

        int row, col, element = 0;

        for(row=0; row<4; row++)
            for(col=0; col<row+1; col++) {
                irregular[row][col] = element;
                element++;
            }
        for(row=0; row<4; row++) {
            for(col=0; col<row+1; col++)
                System.out.print(irregular[row][col] + " ");
            System.out.println();
        }
    }
}
```

```
0
1 2
3 4 5
6 7 8 9
```

```
// Initialize a two-dimensional array.
    class MatrixInitialization {
        public static void main(String[] args[]) {
// Initialize a 4x4 two-dimensional array
            double grid[][] = {
                    { 0*0, 1*0, 2*0, 3*0 },
                    { 0*1, 1*1, 2*1, 3*1 },
                    { 0*2, 1*2, 2*2, 3*2 },
                    { 0*3, 1*3, 2*3, 3*3 }
            };
            int r, c;

            for(r=0; r<4; r++) {
                for(c=0; c<4; c++)
                    System.out.print(grid[r][c] + " ");
                System.out.println();
            }
        }
    }
```

```
0 0 0 0
0 1 2 3
0 2 4 6
0 3 6 9
```

Alternative Array Declaration Syntax

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

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

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

```
int[] nums, nums2, nums3; // create three arrays
int nums[], nums2[], nums3[]; // create three arrays
```

Introducing Type Inference with Local Variables

- **Recently, an exciting new feature called *local variable type inference* was added to the Java language.**
- **In the past, all variables required an explicitly declared type, whether they were initialized or not.**
- **Beginning with JDK 10, it is now possible to let the compiler infer the type of a local variable based on the type of its initializer, thus avoiding the need to explicitly specify the type.**
- **Local variable type inference offers a number of advantages.**
  - **eliminating the need to redundantly specify a variable's type when it can be inferred from its initializer.**
  - **It can simplify declarations in cases in which the type name is quite lengthy, such as can be the case with some class names.**

Introducing Type Inference with Local Variables

```
double avg = 10.0;
var avg = 10.0;
```

```
var myArray = new int[10]; // This is valid.
var[] myArray = new int[10]; // Wrong
var myArray[] = new int[10]; // Wrong
var counter; // Wrong! Initializer required.
var myArray = new int[10]; // This is valid.
var myArray = new int[10]; // This is valid.
```

```
public class TypeInferenceExample {
    public static void main(String[] args) {

        // Using type inference with 'var'
            var d = 5;       // Inferred as int
            var e = 5.5f;    // Inferred as float
            var f = "Java";      // Inferred as int
            System.out.println("Inferred int: " + d);
            System.out.println("Inferred float: " + e);
            System.out.println("Inferred String: " + f);
    }
}
```

```
Inferred int: 5
Inferred float: 5.5
Inferred String: Java
```

Strings

- **String, is not a primitive type**

```
String str = "this is a test";
System.out.println(str);
```

Round a number using format

```
public class Decimal {
    public static void main(String[] args) {
                double num = 1.234567;
                System.out.format("%.2f", num);
    }
     }
```

Formatted output in Java

```
class JavaFormat
{
  public static void main(String args[])
  {
    int x = 10;
    System.out.printf("Print integer: x = %d\n", x);

    // print it upto 2 decimal places
    System.out.printf("Formatted with precision: PI = %.2f\n", Math.PI);
     float n = 5.2f;
     // automatically appends zero to the rightmost part of decimal
    System.out.printf("Formatted to specific width: n = %.4f\n", n);
    n = 2324435.3f;

    // width of 20 characters
    System.out.printf("Formatted to right margin: n = %20.4f\n", n);
  }
}
```