Java Programming

TEK RAJ CHHETRI

Course Objective

- Explain the Java programming environment
- Describe the concepts of programming elements using Java and object-oriented
- programming concepts
- Apply the exception handling and input/output in Java programming
- Apply the event handling, GUI programming using swing, and Java database connectivity

Unit 2: Introduction to Java

- Writing Comments
- Basic Data Types
- Variables and Constants
- Operators
- Type Casting
- Control Flow
- Arrays

Learning Outcome (Unit 2)

- Ability to define suitable data types, variables and constants
- Make use of operators, type casting, arrays along with control statements to solve the problem

Comments

- Used to provide optional information
- Can be single line comment given using //
  - // some information
- And multiline comment given using /* … */
  - /*

    This is multi line comment.

    I can place optional information

    */

Data Types

- Primitive Data Types: 8 types of data: char, int, boolean,…
  - **Integer**: byte, short, int, long
  - **Floating Point**: float, double
  - **Character:** char
  - **Boolean:** boolean

Data Types

Type	Description	Default	Size	Example Literals
boolean	true or false	false	1 bit	true, false
byte	twos complement integer	0	8 bits	(none)
char	Unicode character	\u0000	16 bits	'a', '\u0041', '\101', '\\', '\'', '\n', 'ß'
short	twos complement integer	0	16 bits	(none)
int	twos complement integer	0	32 bits	-2, -1, 0, 1, 2
long	twos complement integer	0	64 bits	-2L, -1L, 0L, 1L, 2L
float	IEEE 754 floating point	0.0	32 bits	1.23e100f, -1.23e-100f, .3f, 3.14F
double	IEEE 754 floating point	0.0	64 bits	1.23456e300d, -1.23456e-300d, 1e1d

Data Types

```
1.  class Data_Types{
2.      public static void main(String[] args) {
3.          int number = 2;
4.          char alpha = 'A';
5.          double decimal = 4.56;
6.          boolean status = true;
7.          System.out.println("Integer: "+number);
8.          System.out.println("Character: "+alpha);
9.          System.out.println("Float: "+decimal);
10.         System.out.println("Boolean: "+status);
11.     }
12. }
```

Literals

- Representation of fixed value in source code.
- Literals can be string, integer, character, Boolean and floating point.
- Eg:
  - “Hello World” = String literal
  - “Two\nLines” = String literal
  - byte a = 68;
  - char a = 'A';

Variables

- DataType variable_name [=value]
- Eg:
  - int a,b,c
  - int d =3, e = 4, f = 5
- Dynamic Initialization
  - double c = Math.sqrt(d*d +e*e)

Constants

- Java do not provide direct mechanism to define constants.
- The keyword static and final are used to define constant.
- Static allows variable to be accessed without loading instance of class.
- The final keyword makes the variable unchangeable.

```
public static final int DEFAULT_VALUE = 40;
```

Keywords

- In addition to keyword, java reserves: true, false, and null.

abstract	continue	for	new	switch
assert***	default	goto*	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum****	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp**	volatile
const*	float	native	super	while

* not used
** added in 1.2
*** added in 1.4
**** added in 5.0

Arithmetic Operators

- Operands of arithmetic operators must be of numeric type.
- Used for mathematical expression.
- Eg:

```
int a=5, b=4;
int c = a + b;
int d = a  % b
```

Operator	Result
+	Addition (also unary plus)
–	Subtraction (also unary minus)
*	Multiplication
/	Division
%	Modulus
++	Increment
+=	Addition assignment
– =	Subtraction assignment
*=	Multiplication assignment
/=	Division assignment
%=	Modulus assignment
– –	Decrement

- **Operators Precedence:**

Highest			
( )	[ ]	.	
++	– –	~	!
*	/	%	
+	–		
>>	>>>	<<	
>	>=	<	<=
==	!=		
&			
^			
|			
&&			
||			
?:			
=	op=		
Lowest			

```
1. class CalculateScore{
2.     public static void main(String[] args) {
3.         double score = 1.0 + 2.0 *3.0;
4.         System.out.println("Score: "+ score);
5.         score = score / 2.0;
6.         System.out.println("New Score: "+score);
7.     }
8. }
```

Bitwise Operators

- Can be applied to integer types, char and byte.
- Used for mathematical expression.
- Eg:

```
int number = 4;
//shift right:: value >> number_of_position
int rightShift = number >> 1;
// result should be 2
System.out.println(rightShift);
```

Operator	Result
~	Bitwise unary NOT
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
>>	Shift right
>>>	Shift right zero fill
<<	Shift left
&=	Bitwise AND assignment
|=	Bitwise OR assignment
^=	Bitwise exclusive OR assignment
>>=	Shift right assignment
>>>=	Shift right zero fill assignment
<<=	Shift left assignment

Operators

a	b	a&b	a|b	a^b	~a
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

```
1.  class Bitwise{
2.      public static void main(String[] args) {
3.          int numberA = 42; //00101010
4.          int numberB = 15; //00001111
5.          //AND operation result needs to be 10
6.          int andOperation = numberA & numberB;
7.          System.out.println("And Operation: "+ andOperation);
8.          //XOR result needs to be 37
9.          int Xor_operation  = numberA ^ numberB;
10.         System.out.println("Xor Operation: "+ Xor_operation);
11.         //OR result needs to be 47
12.         int Or_operation = numberA | numberB;
13.         System.out.println("Or Operation: "+ Or_operation);
14.     }
15. }
```

Relational Operator

- Determine the relationship that one operand has to other.

- Eg:

```
1.  class Relational{
2.      public static void main(String[] args) {
3.                  int number = 4;
4.                  int numberB = 5;
5.                  if(number < numberB){
6.                  System.out.println(number+" is less than"+numberB);

7.                  }
8.
9.      }
10. }
```

Relational Operatiors

Operatior	Operations	Example
<	Less than	a<b
>	Greater than	a>b
<=	Less than or equal to	a<=b
>=	Greater than equal to	a>=b
=	Equal to	a==b
!=	Not equal to	a!=b

Boolean Logical Operator

- Operates only on Boolean operands.
- & , | and ^ operate on **Boolean** values in the same way they operate on bits of integer.

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

Assignment Operator

- Used to assign value.
- = is called assignment operator.
- Eg:
  - int a = 5;
  - String name = “Java Programming”;

Ternary (?) Operator

- ? Is called ternary operator.
- **Syntax:**
  - Expression 1 ? Expression 2 : expression 3
  - If condition in expression 1 is true then expression 2 is evaluated otherwise expression 3 is evaluated.
  - Both expression 1 and 2 are required and can’t be void.

Ternary (?) Operator

```
1.  int num = 5;
2.  int numb = 0;
3.  if (num <= 5){
4.          numb = 10;
5.  }else{
6.          numb = 20;
7.  }
8.  System.out.println(numb);
9.  //same above operations with ternary operator
10. int numc = num <= 5? 10: 20;
11. System.out.println("Ternary: "+numc);
```

Type Casting

- Converting the value from one data type to another.
- 2 types:
  - Implicit / Automatic:
    - Conversion takes place when there is no loss of value on doing so i.e. destination is larger than source.
    - The two types are compatible.
    - Eg: byte b = 5; int c = b;
  - Explicit / Incompatible:
    - Used when we want to convert incompatible type like byte to int.
    - There will be loss of value.
    - Syntax:
    - **(target-type) value**
    - Eg: float b = 5.57; int c = (int) b;

Automatic type promotion in Expressions

- Happens when data type exceeds range or when expression contains different data types.

- Eg:

- byte b = 50;
- b = b * 2; //generate error because b *2 are automatically promoted to int, can't convert byte to int

- byte b = 50;
- b = (byte)b * 2; //no error

Automatic type promotion in Expressions

- **Rules for Type Promotion:**

1. Byte, short, char are automatically promoted to int when evaluating expression.
2. If one operand is long whole expression is promoted to long.
3. If one operand is float, entire expression is promoted to float.
4. If one operand is double, the result is double.

Automatic type promotion in Expressions

```
1.  class Promote{
2.      public static void main(String[] args) {
3.              //Demonstration of automatic type promotion in expression
4.              byte b = 42;
5.              char c = 'a';
6.              short s = 1024;
7.              int i = 50000;
8.              float f = 5.67f;
9.              double d = 0.1234;
10.             double result = (f*b) + (i/c) - (d*s);
11.             System.out.println((f*b) +"+"+ (i/c) +"-"+ (d*s));
12.             System.out.println(result);
13.
14.     }
15. }
```

Control Flow Statements

- These statements execute from top to bottom, in the order they appear and change the flow by employing decision making, looping and branching.
- Decision making includes if-then, if-then-else, switch statements.
- Looping includes for, while, do-while statements .
- Branching statements include break, continue, return.

Control Flow Statements

- **If-then:**
- **Syntax:**
  - if(condition) {
  - // Statements will execute if the condition is true
  - }
- Eg:

```
1. void applyBrakes() {
2.     // the "if" clause: bicycle must be moving
3.     if (isMoving){
4.         // the "then" clause: decrease current speed
5.         currentSpeed--;
6.     }
7. }
```

Control Flow Statements

- **If-then-else:**
- **Syntax:**
  - if(condition) {
  - // Statements will execute if the condition is true
  - }else{
  - //Statements will execute when if condition is false
  - }
- Eg:

```
1.  void applyBrakes() {
2.      if (isMoving) {
3.          currentSpeed--;
4.      } else {
5.          System.err.println("The bicycle has already stopped!");
6.      }
7.  }
```

Control Flow Statements

- **switch:**
- Unlike if-then and if-then-else statements, the switch statement can have a number of possible execution paths.
- **Syntax:**

```
switch(expression) {
   case value :
     // Statements
     break; // optional
   case value :
     // Statements
     break; // optional
     //You can have any number of case statements.
   default : // Optional
     // Statements
}
```

Control Flow Statements

- **While Loop:**

- The while statement continually executes a block of statements while a particular condition is true.
- Syntax:

```
1. while (expression) {
2.     statement(s)
3. }
```

Control Flow Statements

- **Do-while Loop:**

- Do-while evaluates its expression at the bottom of the loop instead of the top.

- Therefore, the statements within the do block are always executed at least once

- **Syntax:**

```
1. do {
2.     statement(s)
3. } while (expression);
```

Control Flow Statements

- **Eg:**

```
1.  class WhileDemo {
2.      public static void main(String[] args){
3.          int count = 1;
4.          while (count < 11) {
5.              System.out.println("Count is: " + count);
6.              count++;
7.          }
8.      }
9.  }
```

Control Flow Statements

- **Eg:**

```
1.  class DoWhileDemo {
2.      public static void main(String[] args){
3.          int count = 1;
4.          do {
5.              System.out.println("Count is: " + count);
6.              count++;
7.          } while (count < 11);
8.      }
9.  }
```

Control Flow Statements

- **For Loop:**
- Loops over a range of value till condition is satisfied.
- **Syntax**:

```
1. for (initialization; termination; increment) {
2.     statement(s)
3. }
```

- When using this version of the for statement, keep in mind that:
  - The initialization expression initializes the loop; it's executed once, as the loop begins.
  - When the termination expression evaluates to false, the loop terminates.
  - The increment expression is invoked after each iteration through the loop; it is perfectly acceptable for this expression to increment or decrement a value.

Control Flow Statements

- **Eg:**

```
1.  class ForDemo {
2.      public static void main(String[] args){
3.          for(int i=1; i<11; i++){
4.              System.out.println("Count is: " + i);
5.          }
6.      }
7.  }
```

Control Flow Statements

- **Advanced For Loop / For-each loop:**
- Used for iterating over collections and arrays.
- **Syntax**:

```
for (type itr-variable: collection) {
    statement(s)
}
```

- Eg:

```
class EnhancedForDemo {
    public static void main(String[] args){
        int[] numbers =  {1,2,3,4,5,6,7,8,9,10};
        for (int item : numbers) {
            System.out.println("Count is: " + item);
        }
    }
```

Control Flow Statements

- **Break:**
- Used to terminate the loop.
- 2 types: labeled and unlabeled.

- **Syntax for break with label**:
  1. break **label**

```
search:
    for (i = 0; i < arrayOfInts.length; i++) {
        for (j = 0; j < arrayOfInts[i].length;
             j++) {
            if (arrayOfInts[i][j] == searchfor) {
                foundIt = true;
                break search;
            }
        }
    }
```

Control Flow Statements

- **Continue:**
- The continue statement skips the current iteration of a for, while , or do-while loop.
- The unlabeled form skips to the end of the innermost loop's body and evaluates the boolean expression that controls the loop.
- A labeled continue statement skips the current iteration of an outer loop marked with the given label.
- **Syntax for continue with label**:
  1. continue **label**

Control Flow Statements

- Eg:

```
1.  for(int i = 1; i < 10 ; i++){
2.      if(i == 5)continue;
3.      System.out.println(i);
4.  }
```

Control Flow Statements

- **Return:**
- The return statement exits from the current method, and control flow returns to where the method was invoked.
- The return statement has two forms: one that returns a value, and one that doesn't.
- Eg:

```
return 50;

return;
```

Arrays

- An array is group of like-typed variables that are referred to by common name.
- Array elements is accessed by index starting from 0(zero) to length-1.
- Two types:
  - One / Single dimensional arrays
  - Multidimensional arrays
- Declaring array
  - DataType array_name[] or
  - DataType [] array_name
  - Single [] specify one dimensional array, increase [] to increase dimension of array.
- Allocating memory
  - new DataType[Size]
- Eg:
  - //creating and allocating memory at the same time
  - int month_days = new int[30];
  - int twoD[][] = new int[4][5];

Arrays

- Array can be initialized when declared.
- For this we use curly braces with values inside it separated by comma.
- Eg:
- Multidimensional array initialization
  - int[][] arrayOfInts = {
  - { 32, 87, 3, 589 },
  - { 12, 1076, 2000, 8 },
  - { 622, 127, 77, 955 }
  - };

- Single dimensional array initialization
  - int[] numbers = {1,2,3,4,5,6,7,8,9,10};

Arrays

- **Accessing array value**:

```
1. int[] numbers = {1,2,3,4,5,6,7,8,9,10};
2. //getting array value at third position
3. numbers[2]; //array index starts at 0
```

```
1. for (int i = 0; i < arr.length; i++){
2.  System.out.println("Element at index " + i + " : "+ arr[i]);
3. }
```

```
1.  double[] myList = {1.9, 2.9, 3.4, 3.5};
2.      // Print all the array elements
3.      for (int i = 0; i < myList.length; i++) {
4.        System.out.println(myList[i] + " ");
5.      }
```

Arrays

- Java array allows rows to vary in length.

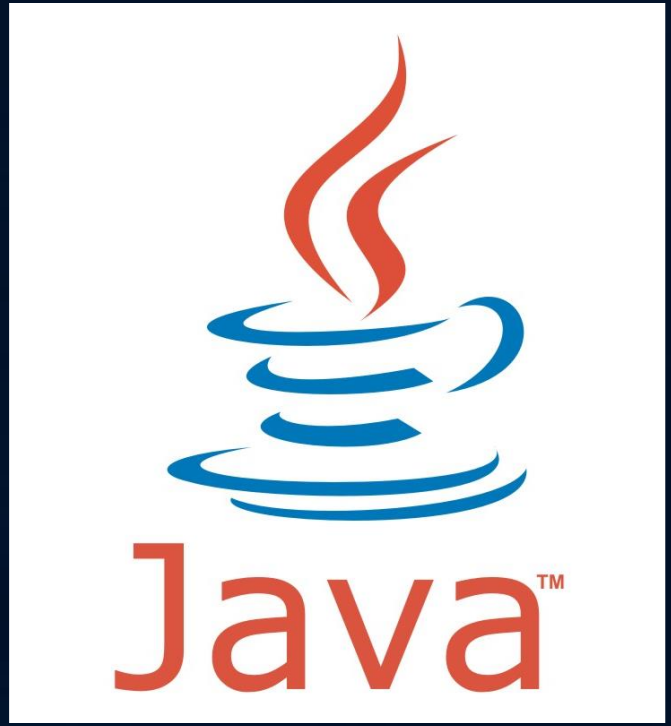
```
1.  class MultiDimArrayDemo {
2.    public static void main(String[] args) {
3.      String[][] names = {
4.        {"Mr. ", "Mrs. ", "Ms. "},
5.        {"Smith", "Jones"}
6.      };
7.      // Mr. Smith
8.      System.out.println(names[0][0] + names[1][0]);
9.      // Ms. Jones
10.     System.out.println(names[0][2] + names[1][1]);
11.   }
12. }
```

- The output from this program is:

- Mr. Smith
- Ms. Jones

Suggested Readings

- The complete Reference Java 7 by Hebert Schildt (P 33–P104)



References

- The complete Reference Java 7 by Hebert Schildt
- Java 8 in Action by Dreamtech press.
- Mit Opencourseware
- https://docs.oracle.com/javase/tutorial/java/
- https://images.google.com for Images