

Language Fundamentals

Unit 2

Unit 2 - Objectives

- **Keywords**
- **Identifiers**
- **Literals**
- **Data Types**
- **Type Conversion**
- **Formatted Output**
- **Operators**
- **Operator Precedence**
- **Comments**
- **Conditional Constructs**
- **Looping Constructs**
- **Processing Arrays**
- **Break and Continue**
- **Recursion**

Keywords

- Keywords are the standard words that constitute the Java language.
- They have pre-defined meaning and cannot be redefined.
- Keywords are all in lowercase.
- There are about 50 keywords in java.

List of Java Keywords

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

Identifiers

Identifiers are user-defined names that are given to variables, functions, arrays, classes etc.

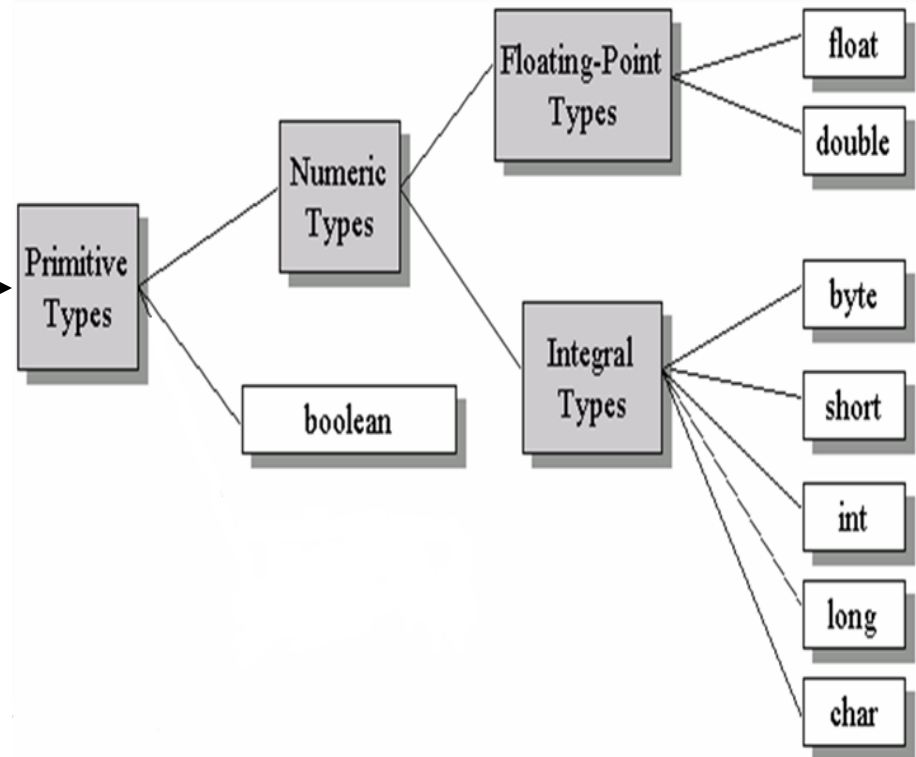
- They are the fundamental requirement of any language.
-
- Rules for naming these identifiers:
 - Only alphabetic characters, digits are permitted. The only other special characters permitted are under score and dollar
 - The name cannot start with a digit.
 - Case-sensitive, uppercase and lowercase letters are distinct.
 - A declared keyword cannot be used as a variable name.

Identifiers

- Identify the legal and illegal identifiers from the following (state the reasons):
 - first
 - Employee Salary
 - conversion
 - Hello!
 - One+two
 - \$test
 - 2nd
 - _myName
 - Employee

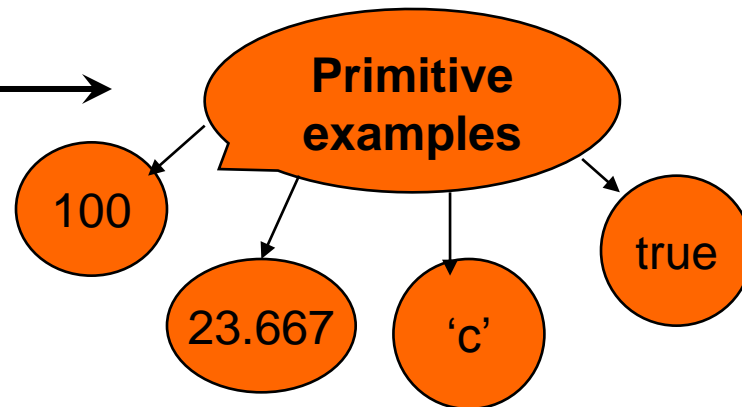
Data types

- Data type determines the values that a variable can contain and the operations that can be performed.
- Are of two types
- **Primitive Types** →
- Reference Types



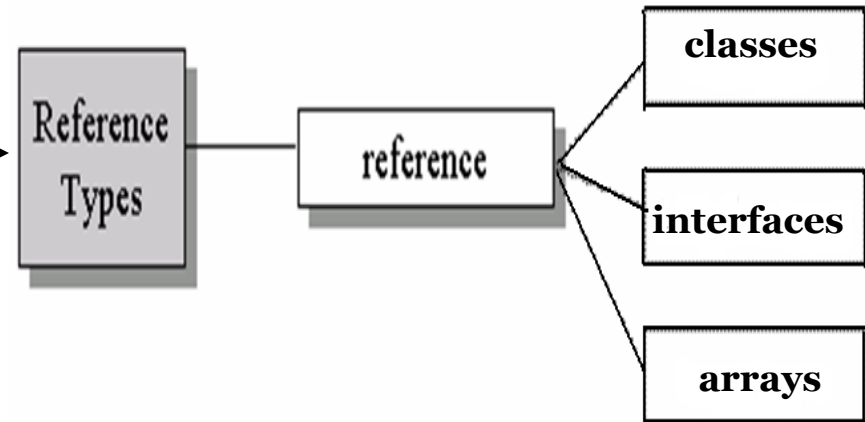
Data types

- Data type determines the values that a variable can contain and the operations that can be performed.
- Are of two types
- **Primitive Types** →
- Reference Types



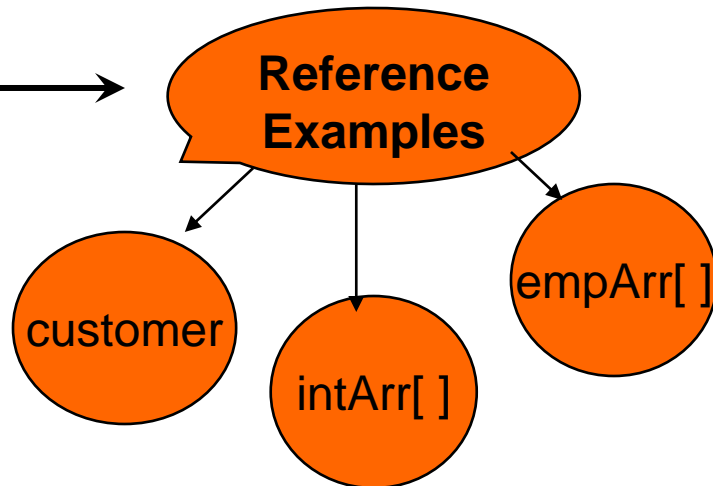
Data types

- Data type determines the values that a variable can contain and the operations that can be performed.
- Are of two types
- Primitive Types
- **Reference Types**



Data types

- Data type determines the values that a variable can contain and the operations that can be performed.
- Are of two types
- Primitive Types
- **Reference Types** →



Primitive types

- There are eight primitive types in Java.
- The size that these data types occupy, irrespective of the platform they operate on, are listed below

Primitive type	Size
boolean	1-bit
char	16-bit
byte	8-bit
short	16-bit
int	32-bit
long	64-bit
float	32-bit
double	64-bit

Primitive types

- **boolean**

- Can take values 'true' or 'false'
- Cannot be used interchangeably with 1 and 0

- **byte**

- Takes values in the range -128 to 127

- **short**

- Takes values in the range -32768 to 32767

- **int**

- All integer values are signed int
- Takes values in the range -2,147,483,648 to 2,147,483,647

- **long**

- Takes values in the range -9223372036854775808 to 9223372036854775807

- **float**

- Takes floating point constant 'f'
- Fractional precision of --- digits after decimal.
- Takes values in the range 1.40239846e-45 to 3.40282347e+38

- **double**

- Takes floating point constant 'd'
- Fractional precision of ---digits after decimal.
- Takes values in the range 4.94065645841246544e-324 to 1.79769313486231570e+308

- **char**

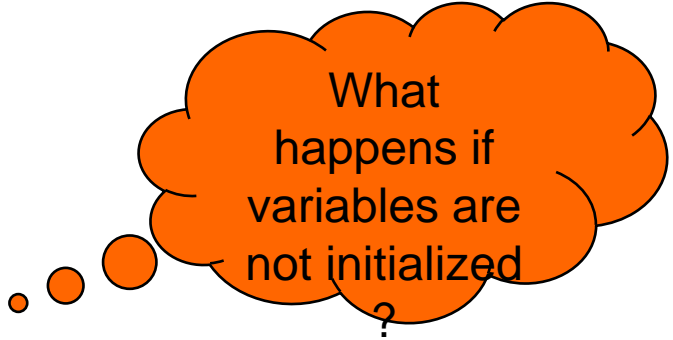
- Unicode character set

Literals

- Literals are the values that are assigned to the identifiers.
- Example
 - 3L
 - Long literals take constant L
 - 100
 - Integer literal
 - 98.6 or 98.6D
 - Double literals optionally take double constant D
 - 98.6f
 - Float literals take floating point constant f
 - 'A'
 - Character literals are enclosed in "
 - "This is a test"
 - String literals are enclosed in ""

Declaring Variables

- General format of variable declaration
type var1, var2, varN;
- Examples
 - `int num1, num2, sum;`
 - `char ch;`
 - `double x, y;`
- It is also possible to initialize the variables at the time of declaration.
 - `int a = 10 , b = 20;`
 - `char ch = 'A';`



What happens if variables are not initialized?

See Listing : **PrimitiveDemo.java**

Exercise

- Write a program to swap the values of two numbers. Display the numbers before swapping and display them again after values are swapped.

Type Casting of Primitives

- A primitive of one data type can be cast to other type in Java.
 - Casting is possible if the the two types are compatible.
 - All numeric types are compatible with each other.
 - Integers are compatible with characters.
 - Boolean is not compatible with any of the data type.
 - Casting is implicit if destination type is larger than source type.
 - Eg : int to double
int to long, short to int.
 - Casting needs to be explicit if the destination type is smaller than source type.This may lead to loss of data.
 - Eg : double to int
long to int

See listing : **PrimitiveTypeCast.java**

Exercise

- To extract the whole and decimal parts of a fractional number.
 - Develop a 'DecimalSplitter' class with methods
 - `getWhole(double d)`
 - Contains logic of extracting the whole part of d
 - `getFraction(double d)`
 - Contains logic of extracting the fractional part of d

The 'String' data-type

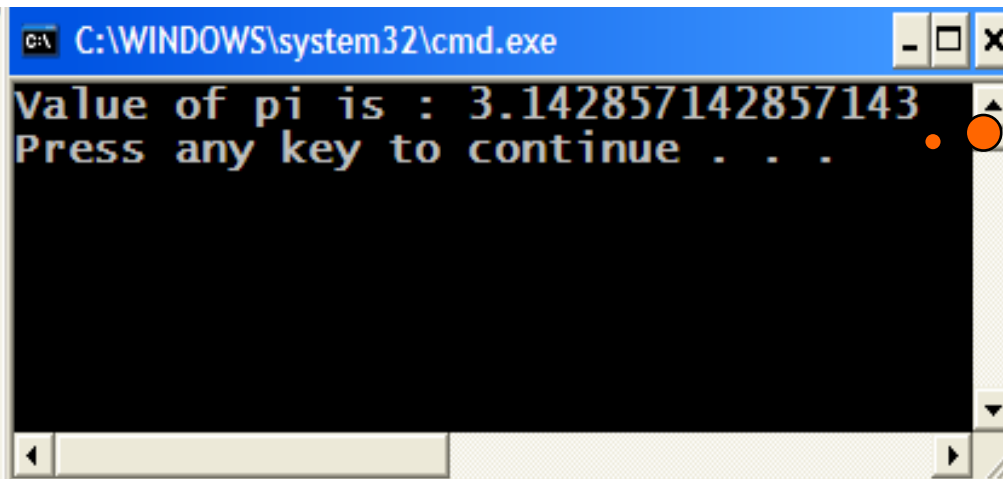
- Management of data comprised of multiple characters can be done through a **String** object.
- The built-in library class 'String' provides support for representing string of characters and performing basic operations on them.
- Concatenation of strings is done using the '+' operator.

See Listing : **StringDemo.java**

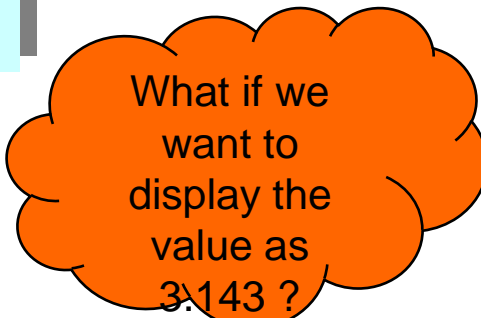
Formatted Output

- Consider the below code and its output

```
class Unformatted
{
    public static void main(String[] args)
    {
        double pi = 22.0/7;
        System.out.println ("Value of pi is : " + pi);
    }
}
```



A screenshot of a Windows command prompt window. The title bar shows the path 'C:\WINDOWS\system32\cmd.exe'. The command prompt displays the output of the Java program: 'Value of pi is : 3.142857142857143' followed by 'Press any key to continue . . .'. The window has standard Windows window controls (minimize, maximize, close) in the top right corner and a scroll bar on the right side.



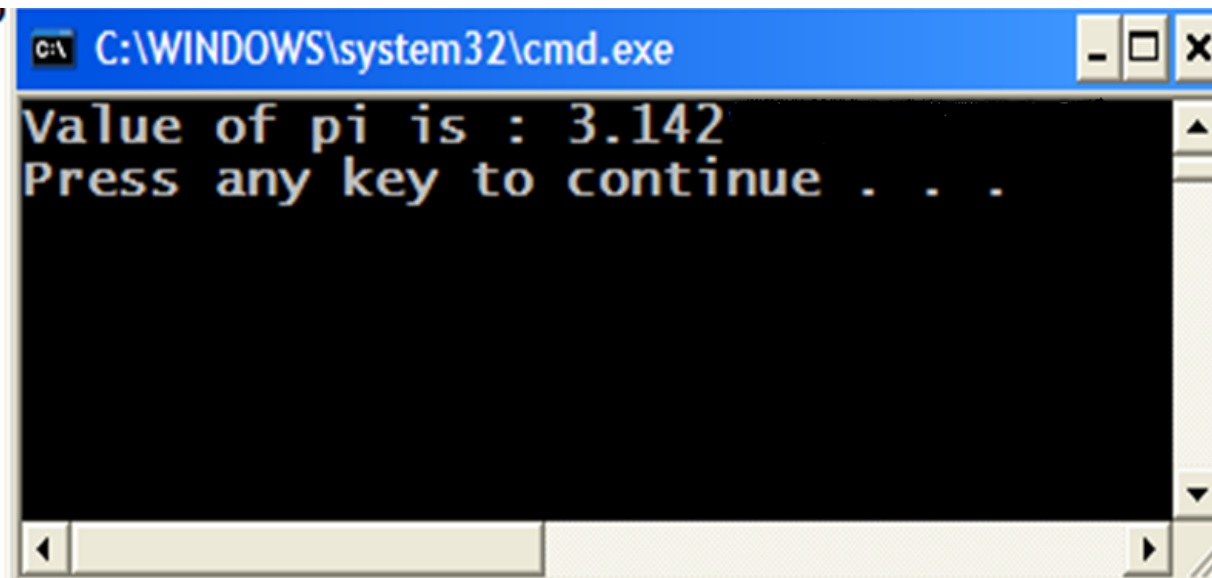
What if we
want to
display the
value as
3.143 ?

Formatted Output

- Java 1.5 introduces to the language the C-style `printf()` method for making formatted output.
- `System.out.printf()` method lets us format a string in accordance with a format specifier, before making an output on the console.
- Each argument to be formatted is described using a string that begins with `%` and ends with the formatted object's type
- Simplest overloaded form of this method
 - **`printf (String format, Object... args)`**
 - `printf (“%d”,i) // where i is an integer`
- Most of C's string formats are available

Formatted Output

```
class Unformatted
{
    public static void main(String[] args)
    {
        double pi = 22.0/7;
        System.out.printf (" Value of pi is : %.3f " + pi);
    }
}
```



A screenshot of a Windows command prompt window. The title bar is blue and contains the text 'C:\WINDOWS\system32\cmd.exe' along with standard window control buttons (minimize, maximize, close). The main area of the window is black with white text. It displays the output of the Java program: 'Value of pi is : 3.142' followed by 'Press any key to continue . . .'. The text is centered and uses a monospaced font. At the bottom of the window, there is a horizontal scrollbar.

Formatted Output

- The below table lists some important format specifiers

Specifier	Description
%n	Outputs the line separator for the platform.
%d	Formats the value as a base-10 integer. Arguments must be Byte, Short,Integer, Long
%f	Formats the value as a floating-point number in base-10, without exponential notation. Arguments must be Float, Double
%e	Formats the value as a base-10 floating-point number, using exponential notation. Arguments must be Float, Double
%%	Escape sequence to allow printing of % in a String.
%c	Formats the value supplied as a single character. Supplied value must be a Byte, Short, Character, or Integer.

Formatted Output

- `System.out.format()` method performs the same function as `System.out.printf()` and they can be used interchangeably.
- For a complete list of all format specifiers refer to java docs for class `Formatter`
- See Listing : [PrintfDemo.java](#)

Java Language Operators

- Operators are used to perform a function on variables.
- Operators that require one operand are called *unary operators*. Ex : num++;
- Operators that require two operands are called *binary operators*. Ex : num1 = num2;
- Operators that require three operands are called *ternary operators*.
Ex : num1>0 ? num1=100 : num1=200;

Java Language Operators

- Arithmetic Operators

- `+, -, *, /, %`

- Relational

- `<, >, <=, >=`

- Conditional / Logical

- `&&, ||, ==, !=`

- Bitwise

- `<<, >>, &, ^, |, ~`

- Assignment

- `=`

Operator Precedence

postfix operators	<code>[] . (params) expr++ expr--</code>
unary operators	<code>++expr --expr +expr -expr ~ !</code>
creation or cast	<code>New (type)expr</code>
multiplicative	<code>* / %</code>
additive	<code>+ -</code>
shift	<code><< >> >>></code>
relational	<code>< > <= >= instanceof</code>
equality	<code>== !=</code>
bitwise AND	<code>&</code>
bitwise exclusive OR	<code>^</code>
bitwise inclusive OR	<code> </code>
logical AND	<code>&&</code>
logical OR	<code> </code>
conditional	<code>? :</code>
assignment	<code>= += -= *= /= %= &= ^= = <<= >>= >>>=</code>

Operator Precedence

- What will the following expressions evaluate to ?
 - $a+b-c*d/e$
 - where $a=2, b=2, c=1, d=3, e=10$

Exercise

- To find the largest of three numbers using ternary operator
 - Develop a 'LargestFinder' class with the following method

```
public int getLargest (int num1, int num2, int num3)
```

 - Write the logic of finding the largest of three numbers using the ternary operator

Comments

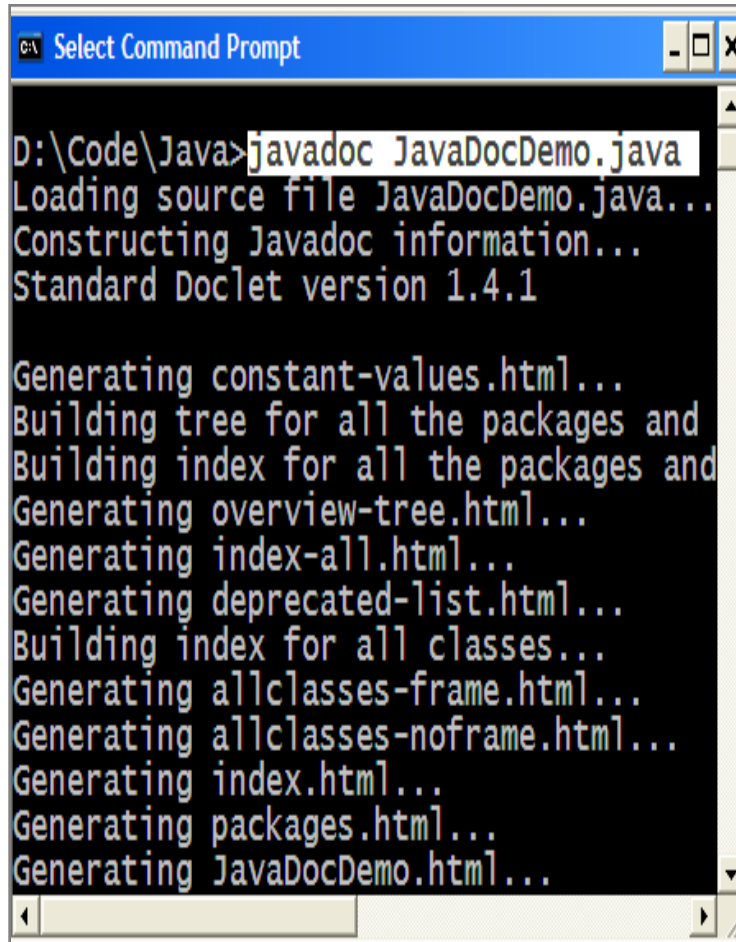
Three kinds of comments

- **Double slashes**
 - `//` Single line Comment
- **C-style**
 - `/*` C-style Comment
Is used to comment out multiple lines `*/`
- **Javadoc comments**
 - **Used to generate documentation**
`/**` This class displays a text string at
* the console.
`*/`

Javadoc comments

```
/** This program is to demonstrate the use of javadoc tool
    @author Sindhu
    */
public class JavaDocDemo
{
    /** This variable holds the value of the computed sum */
    public static int sum = 0;
    /** This is the main() method
        This is the point at which execution starts */
    public static void main(String[] args)
    {
        for(int i=1;i<=5;i++)
            sum = sum + i;
        System.out.println(sum);
    }
}
```

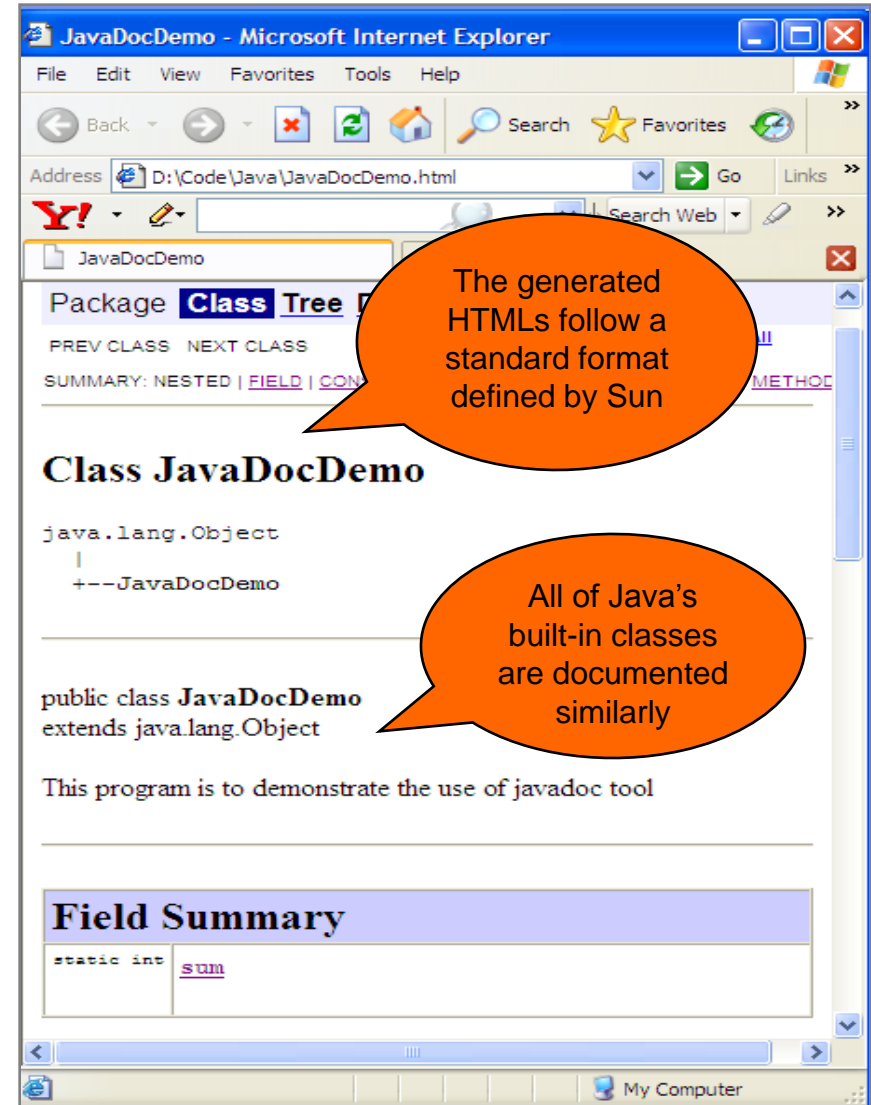
Generating Java docs



```
C:\ Select Command Prompt

D:\Code\Java>javadoc JavaDocDemo.java
Loading source file JavaDocDemo.java...
Constructing Javadoc information...
Standard Doclet version 1.4.1

Generating constant-values.html...
Building tree for all the packages and
Building index for all the packages and
Generating overview-tree.html...
Generating index-all.html...
Generating deprecated-list.html...
Building index for all classes...
Generating allclasses-frame.html...
Generating allclasses-noframe.html...
Generating index.html...
Generating packages.html...
Generating JavaDocDemo.html...
```



JavaDocDemo - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Home Search Favorites

Address <D:\Code\Java\JavaDocDemo.html> Go Links

Y! Search Web

JavaDocDemo

Package **Class** Tree

PREV CLASS NEXT CLASS

SUMMARY: NESTED | [FIELD](#) | [CON](#)

Class JavaDocDemo

`java.lang.Object`
|
+---JavaDocDemo

public class **JavaDocDemo**
extends `java.lang.Object`

This program is to demonstrate the use of javadoc tool

Field Summary

static int	sum
------------	---------------------

My Computer

The generated HTMLs follow a standard format defined by Sun

All of Java's built-in classes are documented similarly

Conditional statements

- Some statements are executed only if certain conditions are met
 - A condition is represented by a logical (Boolean) expression that has a value of either true or false
 - A condition is met if it evaluates to true
- For decision making
 - if – else
 - switch – case

if - else Statement

- if – else statement

```
if (condition)  
{  
    statements  
}  
else  
{  
    statements  
}
```

- Nested if–else statement

```
if (condition)  
{  
    statements  
}  
else if (condition)  
{  
    statements  
}  
else  
{  
    statements  
}
```

Exercise

- To display a student's result
 - Develop a class 'TestResult' with the following method

```
public String getResult( int marks, int marks2, int marks3)
```

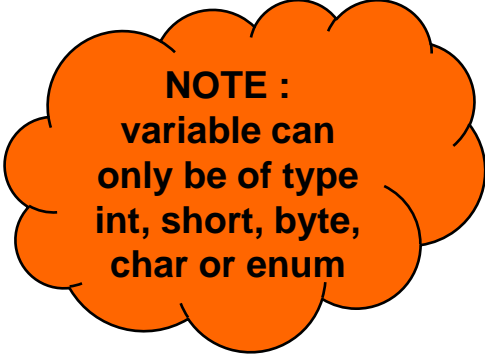
 - Write the logic of returning the result as 'First Class', 'Second Class', 'Pass Class', or 'Fails' based on the average marks secured.

switch – case Statement

- A switch statement is a multi-way decision maker that tests the value of an expression against a list of values.
- When a match is found, the statements associated with that value are executed.

- Syntax

```
switch(variable)
{
    case value-1 : statements
                  break;
    case value-2 : statements
                  break;
    default : statements
}
```



NOTE :
variable can
only be of type
int, short, byte,
char or enum

Structured Loops

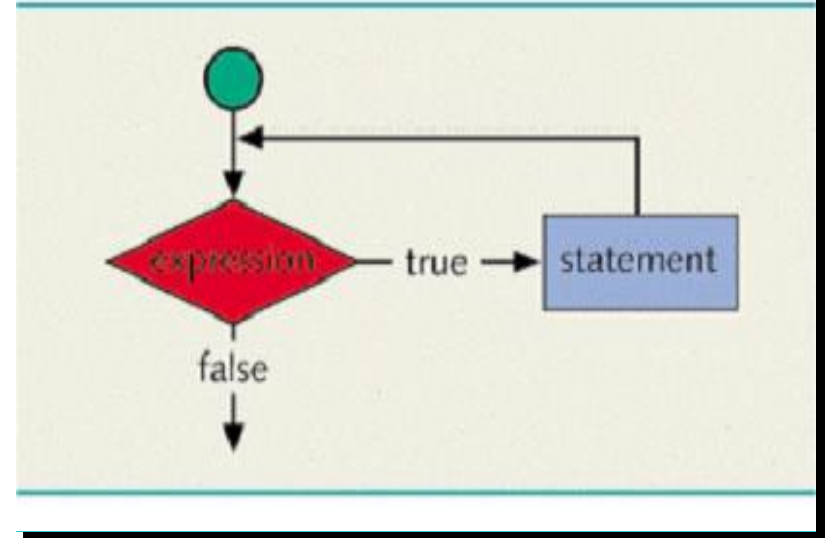
- A loop maybe defined as a set of statements that are repeatedly executed.
- A loop permits the repeated execution of a sequence of statements while some condition is true.
- There are 3 types of loops:
 - **while** loop
 - **do...while** loop
 - **for** loop

while Loop

- The general form of the while statement is:

```
while(expression)
{
    statement....
}
```

```
int i =1;
while (i<=10)
{
    System.out.print (
        i+"\t");
    i++;
}
```



- The loop is entered when the expression is true.
- The loop is terminated when the expression is false.
- This is a pre-tested loop.
- This loop need not be compulsorily executed once.

Exercise

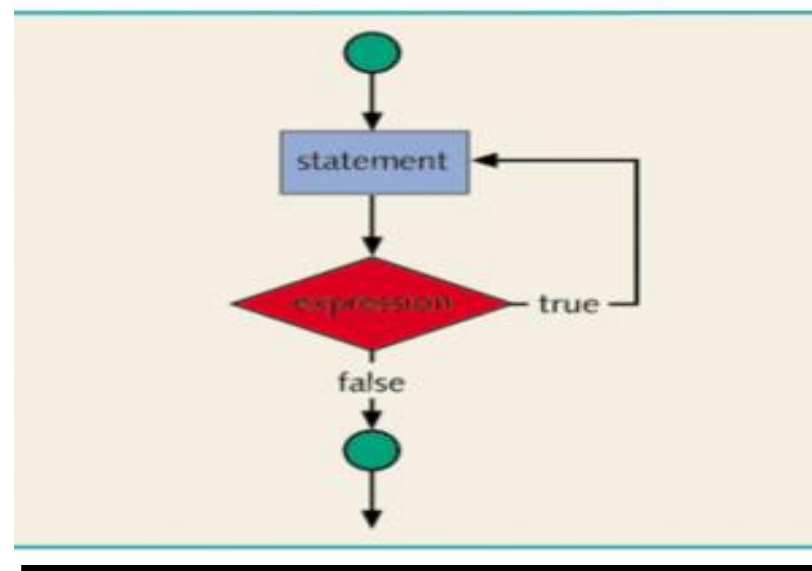
- To display a number in words
 - Develop a class 'NumToWordsConverter' with the following method
`public String numToWords(int number)`
 - Write the logic of constructing a string that represents the number in words.

do... while Loop

- The general form of a do...while statement

```
do
{
    statement....
} while(expression);
```

```
int i =1;
do
{
    System.out.println(i);
    i++;
} while (i<=10);
```



- This loop will be executed at least once.
- The loop is re-entered when the expression is true.
- The loop is terminated when the expression is false.
- This is a post-tested loop.

Exercise

- To accept user choice and perform mathematical computation

for Loop

- The general form of the for statement is:

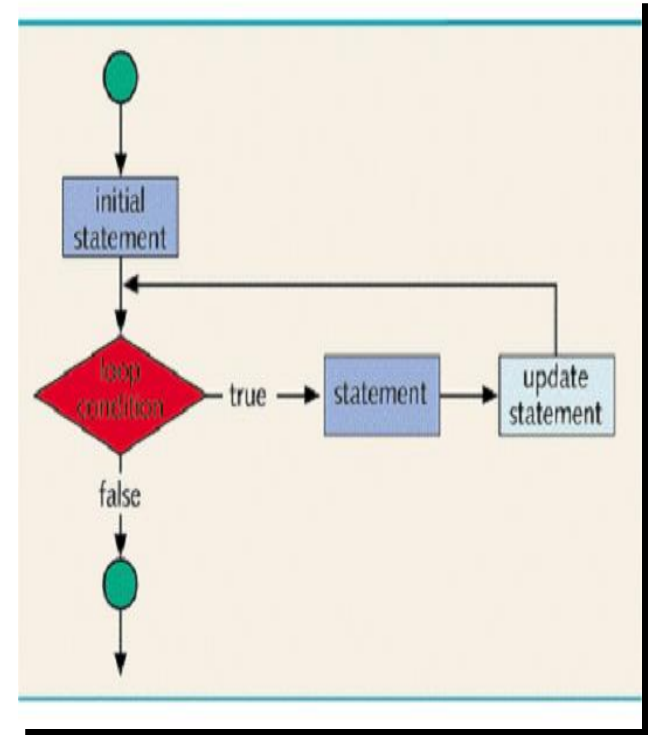
```
for (init counter; testcondition;  
    re-valuation counter)  
{  
    statement.....  
}
```

```
for(int i=0;i <=10;i++)  
{  
    System.out.println(i) ;  
}
```

- Steps involved in execution
 - The initial statement executes
 - The loop condition is evaluated
 - if the loop condition evaluates to true
 - execute the for loop statement
 - execute the update statement (the third expression in the parentheses)
 - Repeat the previous step until the loop condition evaluates to false

for Loop

- The initial statement initializes a variable
- The initial statement in the for loop is the first to be executed and is executed only once
- If the loop condition is initially false, the loop body does not execute
- The update expression, when executed, changes the value of the loop control variable which eventually sets the value of the loop condition to false
- The for loop executes indefinitely if the loop condition is always true



Exercise

- To display the following pattern

*

* *

* * *

* * * *

Break Statement

- The break statement, when executed in a switch structure, provides an immediate exit from the switch structure
 - When the break statement executes in a repetition structure, it immediately exits from these structures
- **The break statement is typically used for two purposes:**
 - 1. To exit early from a loop
 - 2. To skip the remainder of the switch structure

```
while(condition)
{
    statement;
    if(condition)
        break;
    statement;
}
statement;
```

- After the break statement executes, the program continues to execute with the first statement after the structure
- The use of a break statement in a loop can eliminate the use of certain (flag) variables

Exercise

- Write a program to find the sum of all the prime numbers in the range n to m .
Display each prime number and also the final sum.

Continue Statement

- The continue statement is used in while, for, and do-while structures

- **When the continue statement executed in a loop, it skips the remaining statements and proceeds with the next iteration of the loop**

```
while(condition)
{
    statement;
    if(condition)
        continue;
    statement;
}
```

- In a while and do-while structure, the expression (loop-continue test) is evaluated immediately after the continue statement
- In a for structure, the update statement is executed after the continue statement, and then the loop condition executes

Exercise

- Write a program to display the 1st , 2nd , and 4th multiple of 7 which gives the remainder 1 when divided by 2,3,4,5 and 6.

Arrays

- Array - a collection of a fixed number of components wherein all of the components are of the same data type.
 - It is a homogeneous datatype.
- Whenever an array is used, the subscript or the index is involved. The value to be stored or retrieved from the array has to be specified by using both the name of the array and the subscript.

	0	1	2	3	4	5	6	7	8	9
arr =	5	10	15	20	25	30	35	40	45	50

```
arr[0] = 5;  
System.out.println(arr[0]);
```

- In an array of size 'n', the first subscript is always 0 and the last subscript is n-1.

Declaring an Array

- To Declare an array that can hold integer values

```
int[ ] arrayOfInts;
```

```
int arrayOfInts[ ];
```

```
int[ ] arrayOfInts= new int[10];
```

NOTE

Without use of
'new' keyword,
the array is not
allocated memory

- Arrays can contain any legal Java data type including reference types such as objects or other arrays.
- For example, the following declares an array that can contain ten Customer objects.

```
Customer[ ] arrayOfCust = new Customer[10];
```

Initializing an Array

- We can store values into an array by using the index to specify position

Eg:

```
int a[ ] = new int[5];  
a[0] = 10;  
a[1] = 20;  
a[2] = 30;  
a[3] = 40;  
a[4] = 50;
```

- An array can also be initialized at the time of declaration

Eg:

```
int iArray[ ] = { 2, 3, 5, 7, 11, 13 };
```

Accessing elements of the array

- Elements of an array can be accessed by looping through the array, and accessing the element stored at a particular index

```
int[ ] arr = new int[5];  
for(int i=0;i<arr.length;i++)  
    System.out.print(arr[i] + "\t");
```

- See Listing : **ArrayDemo.java**

Exercise

- Write a program to store N elements in an array of integer. Display the elements. Accept a number to be searched. Display whether the number is found or not in the array (LINEAR SEARCH).

Multi-Dimensional Arrays

- An array where elements can be accessed by using more than one subscript is known as multi-dimensional array.
- ***A two-dimensional array can be logically visualized as a collection of rows and columns, like in a matrix***
- A two-dimensional array requires 2 subscript for accessing its elements.

■ Eg:

int a[][] = new int[3][3];

Logically, the array can be visualized as a 3x3 matrix.

	[0]	[1]	[2]
a[0]	100	200	300
a[1]	400	500	600
a[2]	700	800	900

The diagram illustrates a 3x3 matrix. The rows are labeled a[0], a[1], and a[2] on the left. The columns are labeled [0], [1], and [2] at the top. The elements in the matrix are: a[0][0]=100, a[0][1]=200, a[0][2]=300; a[1][0]=400, a[1][1]=500, a[1][2]=600; a[2][0]=700, a[2][1]=800, a[2][2]=900. An arrow points from the cell containing 600 to the label a[1][2].

Exercise

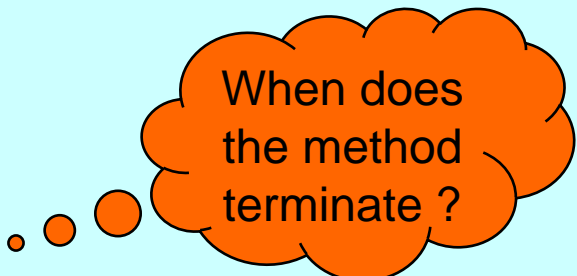
- To check if a given square matrix is identity and / or symmetric
 - Develop a MatrixWizard class with the following methods
`public boolean isIdentity(int[][] matrix)`
 - Check if the matrix is identity or not and return a boolean value indicating it
`public boolean isSymmetric(int[][] matrix)`
 - Check if the matrix is symmetric or not and return a boolean value indicating it

Recursion

- Recursion is an algorithmic technique where a function / method, in order to accomplish a task, calls itself with some part of the task.

- Example

```
public void recurse (int n)  
{  
    System.out.println(n);  
    recurse(n-1);  
}
```



When does
the method
terminate ?

- Method call should terminate at some point, else the recursive method will be called infinitely.

```
public void recurse(int n)  
{  
    System.out.println(n);  
    recurse(n-1);  
    if(n == 0)  
        return;  
}
```

Recursion

- Consider finding the sum of elements in an array from a start index to the last element in the array.
- This can be recursively performed
See Listing : **RecursionDemo.java**

Exercise

- To find the factorial of a number using recursion
 - Develop a class `FactorialGenerator` with the following method
`public int getFactorial(int num)`
 - Write the logic of finding the factorial by using the technique of recursion.

Question time

