


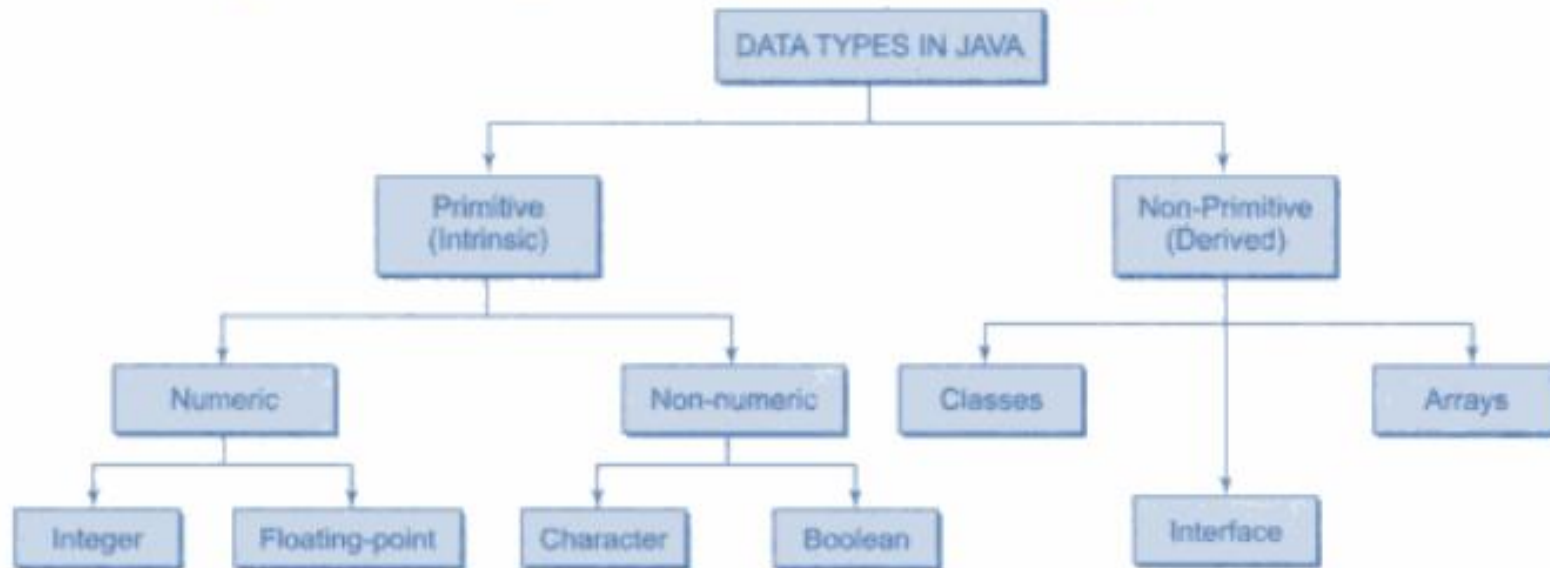
# OVERVIEW OF JAVA LANGUAGE

Java programming

# Primitive Data Types

- There are eight primitive datatypes supported by Java.
- Primitive datatypes are predefined by the language and named by a keyword.

- 
- There are two data types available in Java –
  - Primitive Data Types
  - Reference/Object Data Types



**Fig. 4.2** *Data types in Java*

- ❑ `int side, SIDE;`
- ❑ `side=12;`
- ❑ `float f1=1.5f;`
- ❑ `double d1=1.5;`
- ❑ `char c1='A';`

# Java Tokens

- ❑ Java tokens are smallest elements of a program which are identified by the compiler.
- ❑ Tokens in java include identifiers, keywords, literals, operators and, separators.

# Keyword

- ❑ Keywords are pre-defined or reserved words in a programming language.
- ❑ Each keyword is meant to perform a specific function in a program.
- ❑ Since keywords are referred names for a compiler, they can't be used as variable names because by doing so, we are trying to assign a new meaning to the keyword which is not allowed

# Identifiers

- ❑ Identifiers are used as the general terminology for naming of variables, functions and arrays.
- ❑ These are user-defined names consisting of an arbitrarily long sequence of letters and digits with either a letter or the underscore(\_) as a first character.
- ❑ Identifier names must differ in spelling and case from any keywords. You cannot use keywords as identifiers; they are reserved for special use.
- ❑ Once declared, you can use the identifier in later program statements to refer to the associated value.
- ❑ A special kind of identifier, called a statement label, can be used in goto statements.



# Constants

- ❑ Constants are also like normal variables. But, the only difference is, their values can not be modified by the program once they are defined.
- ❑ Constants refer to fixed values.
- ❑ They are also called as literals.
- ❑ Constants may belong to any of the data type.

**Syntax: final data\_type variable\_name;**

# Special Symbols

- The following special symbols are used in Java having some special meaning and thus, cannot be used for some other purpose.

[] () {}, ; \* =

# Special Symbols

- **Brackets[]**: Opening and closing brackets are used as array element reference. These indicate single and multidimensional subscripts.
- **Parentheses()**: These special symbols are used to indicate function calls and function parameters.
- **Braces{ }**: These opening and ending curly braces marks the start and end of a block of code containing more than one executable statement.
- **comma (, )**: It is used to separate more than one statements like for separating parameters in function calls.
- **semi colon :** It is an operator that essentially invokes something called an initialization list.
- **assignment operator=**: It is used to assign values.

# Operators

- Java provides many types of operators which can be used according to the need. They are classified based on the functionality they provide

# Operators

Operator	Symbols	Operator	Symbols
Arithmetic	+ , - , / , * , %	Logical	&& ,
Unary	++ , - - , !	Ternary	(Condition) ? (Statement1) : (Statement2);
Assignment	= , += , -= , *= , /= , %= , ^=	Bitwise	& ,   , ^ , ~
Relational	== , != , < , > , <= , >=	Shift	<< , >> , >>>

# What is a Variable in Java?

- ❑ **Variable in Java** is a data container that stores the data values during Java program execution.
- ❑ Every variable is assigned data type which designates the type and quantity of value it can hold.
- ❑ Variable is a memory location name of the data.
- ❑ The Java variables have mainly three types : Local, Instance and Static.

# Variable in Java

- ❑ In order to use a variable in a program you need to perform 2 steps
- ❑ Variable Declaration
- ❑ Variable Initialization

# Variable Declaration:

- ❑ To declare a variable, you must specify the data type & give the variable a unique name.

- ❑ Examples of other Valid Declarations are

`int a,b,c;`

`float pi;`

`double d;`

`har a;`



# Rules to create Variables:

- ❑ A variable name can consist of Capital letters A-Z, lowercase letters a-z, digits 0-9, and two special characters such as underscore and dollar Sign.
- ❑ The first character must be a letter.
- ❑ Blank spaces cannot be used in variable names.
- ❑ Java keywords cannot be used as variable names.
- ❑ Variable names are case-sensitive.

# Variable Initialization:

- To initialize a variable, you must assign it a valid value.

```
float pi =3.14f;
```

```
a='v';
```

```
S=123;
```

# Assignment 1

- Write a program to print area of Triangle; where  $\text{area} = 1/2 \times \text{base} \times \text{height}$

# Types of variables

- ❑ Local variables
- ❑ Instance variables
- ❑ Class/Static variables

1) Local Variable: A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.

A local variable cannot be defined with "static" keyword.

2) Instance Variable: A variable declared inside the class but outside the body of the method, is called an instance variable. It is not declared as static.

It is called an instance variable because its value is instance-specific and is not shared among instances.

3) Static variable: A variable that is declared as static is called a static variable. It cannot be local. You can create a single copy of the static variable and share it among all the instances of the class. Memory allocation for static variables happens only once when the class is loaded in the memory.

# Example

```
□ public class A
□ {
□     static int m=100;//static variable
□     void method()
□     {
□         int n=90;//local variable
□     }
□     public static void main(String args[])
□     {
□         int data=50;//instance variable
□     }
□ }//end of class
```

# Local Variables

- ❑ Local variables are declared in methods, constructors, or blocks.
- ❑ Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor, or block.
- ❑ Access modifiers cannot be used for local variables.
- ❑ Local variables are visible only within the declared method, constructor, or block.
- ❑ Local variables are implemented at stack level internally.
- ❑ There is no default value for local variables, so local variables should be declared and an initial value should be assigned before the first use.

```
class Addition {  
    int c;  
  
    void getData(int x, int y ){  
        c=x+y;  
        System.out.println("Addition  
result= "+c);  
  
    } }  
}
```

```
public class Add2Numbers {  
    double a,b;  
    public static void main(String s1 []){  
  
        Scanner sc=new Scanner(System.in);  
        System.out.println("Enter value for first  
variable");  
        a=sc.nextDouble();  
        System.out.println("Enter value for Second  
variable");  
        b=sc.nextDouble();  
        Addition a1=new Addition();  
        A1.getData();  
    } }  
}
```



# Instance Variables

- Instance variables are declared in a class, but outside a method, constructor or any block.
- When a space is allocated for an object in the heap, a slot for each instance variable value is created.
- Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.
- Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object's state that must be present throughout the class.
- Instance variables can be declared in class level before or after use.
- Access modifiers can be given for instance variables.
- The instance variables are visible for all methods, constructors and block in the class. Normally, it is recommended to make these variables private (access level). However, visibility for subclasses can be given for these variables with the use of access modifiers.
- Instance variables have default values. For numbers, the default value is 0, for Booleans it is false, and for object references it is null. Values can be assigned during the declaration or within the constructor.
- Instance variables can be accessed directly by calling the variable name inside the class. However, within static methods (when instance variables are given accessibility), they should be called using the fully qualified name. *ObjectReference.VariableName*.

# Class/Static Variables

- Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block.
- There would only be one copy of each class variable per class, regardless of how many objects are created from it.
- Static variables are rarely used other than being declared as constants. Constants are variables that are declared as public/private, final, and static. Constant variables never change from their initial value.
- Static variables are stored in the static memory. It is rare to use static variables other than declared final and used as either public or private constants.
- Static variables are created when the program starts and destroyed when the program stops.
- Visibility is similar to instance variables. However, most static variables are declared public since they must be available for users of the class.
- Default values are same as instance variables. For numbers, the default value is 0; for Booleans, it is false; and for object references, it is null. Values can be assigned during the declaration or within the constructor. Additionally, values can be assigned in special static initializer blocks.
- Static variables can be accessed by calling with the class name *ClassName.VariableName*.
- When declaring class variables as public static final, then variable names (constants) are all in upper case. If the static variables are not public and final, the naming syntax is the same as instance and local variables.

# Java - Modifier

Modifiers are keywords that you add to those definitions to change their meanings. Java language has a wide variety of modifiers, including the following –

- Java Access Modifiers
- Non Access Modifiers

# Access Control Modifiers

Java provides a number of access modifiers to set access levels for classes, variables, methods and constructors. The four access levels are –

- ❑ Visible to the package, the default. No modifiers are needed.
- ❑ Visible to the class only (private).
- ❑ Visible to the world (public).
- ❑ Visible to the package and all subclasses (protected).

# Non-Access Modifiers

Java provides a number of non-access modifiers to achieve many other functionality.

- ❑ The *static* modifier for creating class methods and variables.
- ❑ The *final* modifier for finalizing the implementations of classes, methods, and variables.
- ❑ The *abstract* modifier for creating abstract classes and methods.
- ❑ The *synchronized* and *volatile* modifiers, which are used for threads.

# Constants in JAVA

- ❑ **Constant** is a value that cannot be changed after assigning it. Java does not directly support the constants
- ❑ In Java, to declare any variable as constant, we use static and final modifiers. It is also known as **non-access** modifiers.
- ❑ According to the Java naming convention the identifier name must be in **capital letters**.

# Static and Final Modifiers

- ❑ The purpose to use the static modifier is to manage the memory.
- ❑ It also allows the variable to be available without loading any instance of the class in which it is defined.
- ❑ The final modifier represents that the value of the variable cannot be changed. It also makes the primitive data type immutable or unchangeable.



**Syntax: static final** datatype identifier\_name=value;

Example: **static final double** PI=3.14;



# Points to remember:

- ❑ Write the identifier name in capital letters that we want to declare as constant. For example, **MAX=12**.
- ❑ If we use the **private** access-specifier before the constant name, the value of the constant cannot be changed in that particular class.
- ❑ If we use the **public** access-specifier before the constant name, the value of the constant can be changed in the program.

# Symbolic Constants

- Constants may appear repeatedly in number of places in the program.
- Constant values are assigned to some names at the beginning of the program, then the subsequent use of these names in the program has the effect of caving their defined values to be automatically substituted in appropriate points.
- The constant is declared as follows:

**Syntax :** final type symbolicname= value;

Eg    final float    PI =3.14159;  
      final int    STRENGTH =100;

# Rules

- ❑ symbolic names take the same form as variable names. But they are written in capitals to distance from variable names. This is only convention not a rule.
- ❑ After declaration of symbolic constants they shouldn't be assigned any other value within the program by using an assignment statement.
- ❑ Symbolic constants are declared for types that are not done in C & C++ where symbolic constants are defined using the define statement.
- ❑ They can't be declared inside a method. They should be used only as class data members in the beginning of the class.

# Operators in Java

- ❑ Operators in Java are the special type of tokens in Java which when coupled with entities such as variables or constants or data types result in a specific operation.
- ❑ An operator is a character that **represents an action**, for example  $+$  is an arithmetic operator that represents addition.

# Types of Operators

Java provides a rich set of operators to manipulate variables. We can divide all the Java operators into the following groups –

- ❑ Arithmetic Operators
- ❑ Relational Operators
- ❑ Bitwise Operators
- ❑ Logical Operators
- ❑ Assignment Operators
- ❑ Misc Operators

- Unary Operators:  $a++$ ;  $--b$ ;
- Binary operators:  $a+b$ ;  $a*b$ ;
- Ternary Operators:  $c=a>b? a : b$

# Arithmetic Operators(Binary Operators):

- They are used to perform simple arithmetic operations on primitive data types.
  - ▣ \* : Multiplication
  - ▣ / : Division
  - ▣ % : Modulo
  - ▣ + : Addition
  - ▣ - : Subtraction

# Relational Operators(Binary operators):

- The **Java Relational operators** compare between operands and determine the relationship between them.

==	Is equal to
!=	Is not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to



# Logical Operators

- ❑ The **Java Logical Operators** work on the Boolean operand. It's also called Boolean logical operators.
- ❑ It operates on two Boolean values, which return Boolean values as a result.

# Logical Operators

<b>&amp;&amp;</b>	<b>Logical AND</b>	<b>If both operands are true then only "logical AND operator" evaluate true.</b>
<b>  </b>	Logical OR	The logical OR operator is only evaluated as true when one of its operands evaluates true. If either or both expressions evaluate to true, then the result is true.
<b>!</b>	Logical Not	Logical NOT is a Unary Operator, it operates on single operands. It reverses the value of operands, if the value is true, then it gives false, and if it is false, then it gives true.

# Assignment Operators

- ❑ The **Java Assignment Operators** are used when you want to assign a value to the expression.
- ❑ The assignment operator denoted by the single equal sign =

Syntax: variable = expression;

<b>+=</b>	<b>Increments then assigns</b>
<b>-=</b>	Decrements then assigns
<b>*=</b>	Multiplies then assigns
<b>/=</b>	Divides then assigns
<b>%=</b>	Modulus then assigns
<b>&lt;&lt;=</b>	Binary Left Shift and assigns
<b>&gt;&gt;=</b>	Binary Right Shift and assigns

<b>&gt;&gt;&gt;=</b>	<b>Shift right zero fill and assigns</b>
<b>&amp;=</b>	Binary AND assigns
<b>^=</b>	Binary exclusive OR and assigns
<b> =</b>	Binary inclusive OR and assigns

# Bitwise Operators

- The **Java Bitwise Operators** allow access and modification of a particular bit inside a section of the data.
- It can be applied to integer types and bytes, and cannot be applied to float and double.

# Bitwise Operators

- **&, Bitwise AND operator:** returns bit by bit AND of input values.
- **|, Bitwise OR operator:** returns bit by bit OR of input values.
- **^, Bitwise XOR operator:** returns bit by bit XOR of input values.
- **~, Bitwise Complement Operator:** This is a unary operator which returns the one's complement representation of the input value, i.e. with all bits inversed.

- ❑  $12 = 00001100$  (In Binary)
- ❑  $25 = 00011001$  (In Binary)
- ❑ Bitwise OR Operation of 12 and 25

00001100

$\wedge$  00011001

---

00010101 = 21 (In Decimal)

# Bitwise Operators-Shift Operators

- ❑ **Shift Operators** : These operators are used to shift the bits of a number left or right thereby multiplying or dividing the number by two respectively.
- ❑ They can be used when we have to multiply or divide a number by two.

General format:

number **shift\_op** number\_of\_places\_to\_shift;



# Shift Operators

- **<<, Left shift operator:** shifts the bits of the number to the left and fills 0 on voids left as a result. Similar effect as of multiplying the number with some power of two.
- **>>, Signed Right shift operator:** shifts the bits of the number to the right and fills 0 on voids left as a result. The leftmost bit depends on the sign of initial number. Similar effect as of dividing the number with some power of two.
- **>>>, Unsigned Right shift operator:** shifts the bits of the number to the right and fills 0 on voids left as a result. The leftmost bit is set to 0.

# Assignment #1

- Differentiate between  $>>$  and  $>>>$ .
- Define JVM and its working.

# Conditional Operator

- ❑ The **Java Conditional Operator** selects one of two expressions for evaluation, which is based on the value of the first operands.
- ❑ It is also called **ternary operator** because it takes three arguments.

**Syntax: expression1 ? expression2:expression3;**

# instanceof Operator

- The **Java instanceof Operator** is used to determining whether this object belongs to this particular (class or subclass or interface) or not.

**Syntax: object-reference instanceof type;**

# Unary Operators

Unary operators are those which have only one operand. They are of the following types:

+	<b>Unary plus:</b> not necessary to use since numbers are positive without using it
-	<b>Unary minus:</b> inverts the sign of an expression
++	<b>Increment operator:</b> increments value by 1
--	<b>Decrement operator:</b> decrements value by 1
!	<b>Logical complement operator:</b> inverts the value of a boolean

# References

- E.Balaguruswamy, Programming with JAVA, A primer, 3e, TATA McGraw-Hill Company.
- <https://www.javatpoint.com/operators-in-java>
- <https://www.geeksforgeeks.org/operators-in-java/>