OOP using Java

Trainer: Mr. Rohan Paramane

Entry point method

- Syntax:
  1. **public static void main(String[] args)**
  2. **public static void main(String... args)**

- Java compiler do not check/invoke main method. JVM invoke main method.

- When we start execution of Java application then JVM starts execution of two threads:
  1. Main thread : responsible for invoking main method.
  2. Garbage Collector : responsible for deallocating memory of unused object.

- We can overload main method in Java.

- We can define main method per class. But only one main method can be considered as entry point method.

Language Basics

- Keywords
- Data Types
- Variables
- Operators
- Conditional Statements
- Loops
- Coding Conventions

Keywords

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

Data Types

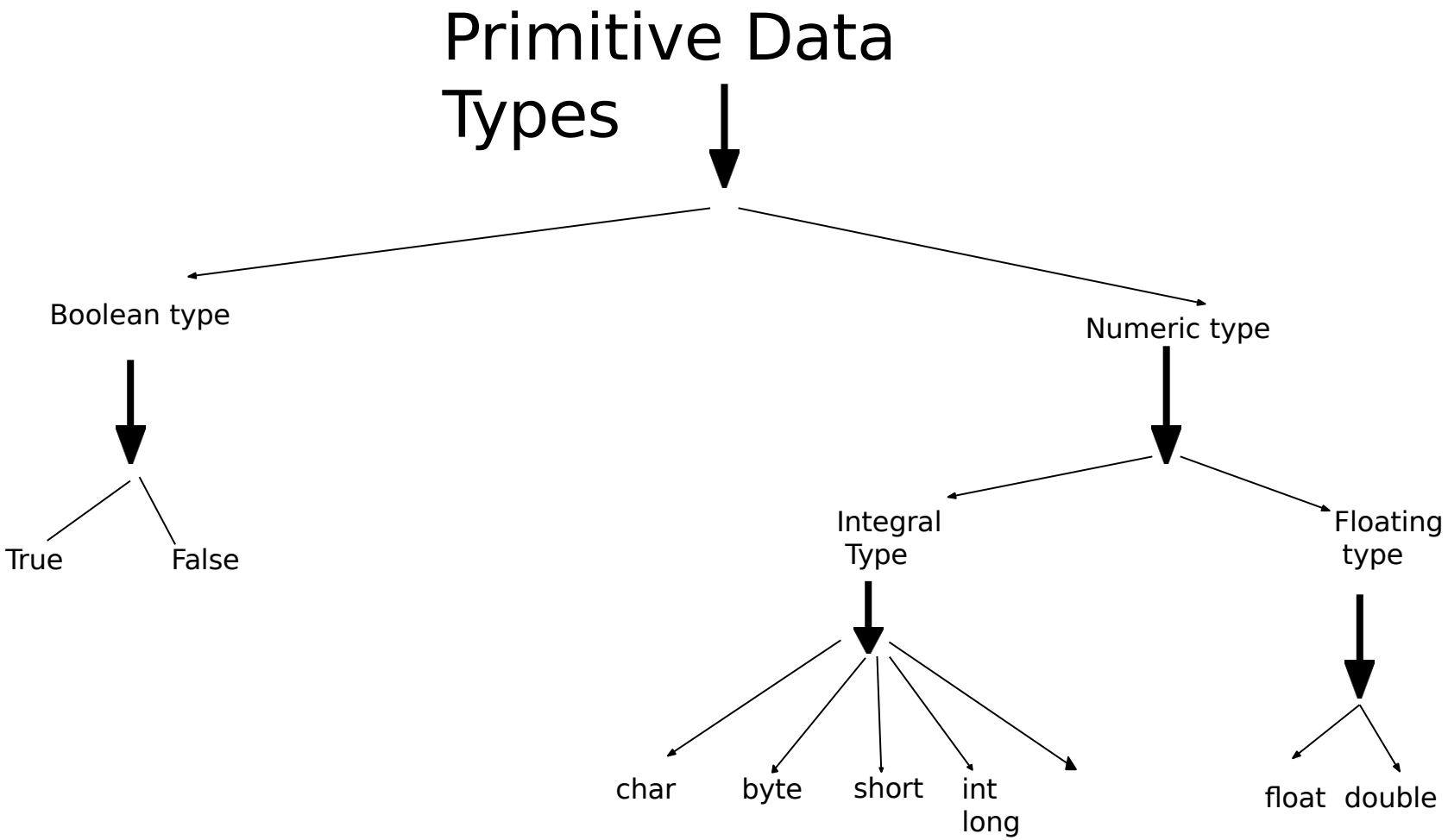
- Data type of any variable decide following things:
  1. Memory: How much memory is required to store the data.
  2. Nature: Which kind of data is allowed to store inside memory.
  3. Operation: Which operations are allowed to perform on the data stored in memory.
  4. Range: Set of values that we can store inside memory.

- The Java programming language is a statically typed language, which means that every variable and every expression has a type that is known at compile time.

  Types of data type:

  1. Primitive type(also called as value type)
     a. boolean type
     b. Numeric type
        i. Integral types (byte, char, short, int, long)
        ii. Floating point types (float, double)
  2. Non primitive type(also called as reference type)
     a. Interface, Class, Type variable, Array

Data Types



Reference Types

1. Class types
2. Interface types
3. Array types

Data Types

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits. Range is 1.40239846e-45f to 3.40282347e+38f
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits. Range is 4.94065645841246544e-324 to 1.79769313486231570e+308
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

Variables

- A **variable** is a name given memory location.
- That memory is associated to a data type and can be assigned a value.
- int n; float f1; char ch; double d;

- Rules of Variables
  - All variable names must begin with a letter of the alphabet, an underscore (_), or a dollar sign ($). Can't begin with a digit. The rest of the characters may be any of those previously mentioned plus the digits 0-9.
  - The convention is to always use a (lower case) letter of the alphabet. The dollar sign and the underscore are discouraged.

```
int n1;
n1=21;              // assignment
int val=50;         //initialization
double d = 21.8;    // initialization
d = n1;             // assignment
float f1 = 16.13f;
```

Operators

1. Arithmetic Operators
2. Unary Operators
3. Assignment Operator
4. Relational Operators
5. Logical Operators
6. Ternary Operator
7. Bitwise Operators
8. Shift Operators

Arithmetic and Unary Operators

- **Arithmetic Operators**
  1. They are used to perform simple arithmetic operations on primitive data types.
  2. * : Multiplication
  3. / : Division
  4. % : Modulo
  5. + : Addition
  6. – : Subtraction

- **Unary Operators**
  - Unary operators need only one operand. They are used to increment, decrement or negate a value.
  - ++ :Increment operator, used for incrementing the value by 1.
  - There are two varieties of increment operator.
    - Post-Increment : Value is first used for computing the result and then incremented.
    - Pre-Increment : Value is incremented first and then result is computed.
  - -- : Decrement operator, used for decrementing the value by 1.
  - There are two varieties of decrement operator.
    - Post-decrement : Value is first used for computing the result and then decremented.
    - Pre-Decrement : Value is decremented first and then result is computed.

Assignment and Relational operators

- **Assignment Operators**
  - '=' Assignment operator is used to assign a value to any variable. It has a right to left associativity.
  - Eg. int val = 500;
  - assignment operator can be combined with other operators to build a shorter version of statement called Compound Statement.
  - +=, for adding left operand with right operand and then assigning it to variable on the left.
  - -=, for subtracting left operand with right operand and then assigning it to variable on the left.
  - *=, for multiplying left operand with right operand and then assigning it to variable on the left.
  - /=, for dividing left operand with right operand and then assigning it to variable on the left.
  - %=, for assigning modulo of left operand with right operand and then assigning it to variable on the left.

- **Relational Operators**
  - These operators are used to check for relations like equality, greater than, less than.
  - They return boolean result after the comparison and are used in looping, conditional and if else statements.
  - ==, Equal to : returns true if left hand side is equal to right hand side.
  - !=, Not Equal to : returns true if left hand side is not equal to right hand side.
  - <, less than : returns true if left hand side is less than right hand side.
  - <=, less than or equal to : returns true if left hand side is less than or equal to right hand side.
  - >, Greater than : returns true if left hand side is greater than right hand side.
  - >=, Greater than or equal to: returns true if left hand side is greater than or equal to right hand side.

Logical & Ternary operators

- **Logical Operators :**
  - These operators are used to perform “logical AND” and “logical OR” operation.
  - &&, Logical AND : returns true when both conditions are true.
  - ||, Logical OR : returns true if at least one condition is true.
  - eg : int data1=100; int data2=50;
  - if(data1 > 60 && data2 < 100)
    - System.out.println("test performed...");
  - else
    - System.out.println("test not performed...");

- **Ternary operator :**
  - Ternary operator is a shorthand version of if-else statement.
  - It has three operands and hence the name ternary.
  - General format is : condition ? if true : if false
  - The above statement means that if the condition evaluates to true, then execute the statements after the ‘?’ else execute the statements after the ‘:’.
  - eg : int data=100;
  - System.out.println(data>100?"Yes":"No");

Bitwise and Left shift Operator

- **Bitwise Operators :**
  - These operators are used to perform manipulation of individual bits of a number. They can be used with any of the integer types.
  - &, 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 compliment representation of the input value, i.e. with all bits inversed.

- **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.
  - <<, 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.
  - eg : int a = 25;
  - System.out.println(a<<4); //25 * 16 = 400
  - a=-25;
  - System.out.println(a<<4);//-25 * 16 = -400

Right Shift Operator

- **Signed right shift operator :** The signed right shift operator '>>' uses the sign bit to fill the trailing positions.
  For example, if the number is positive then 0 will be used to fill the trailing positions and if the number is negative then 1 will be used to fill the trailing positions.

  Assume if a = 60 and b = -60; now in binary format, they will be as follows −
  a = 0000 0000 0000 0000 0000 0000 0011 1100
  b = 1111 1111 1111 1111 1111 1111 1100 0100

  In Java, negative numbers are stored as 2's complement.

  Thus a >> 1 = 0000 0000 0000 0000 0000 0000 0001 1110
  And b >> 1 = 1111 1111 1111 1111 1111 1111 1110 0010

- **Unsigned right shift operator :** The unsigned right shift operator '>>' do not use the sign bit to fill the trailing positions. It always fills the trailing positions by 0s.

  Thus a >>> 1 = 0000 0000 0000 0000 0000 0000 0001 1110
  And b >>> 1 = 0111 1111 1111 1111 1111 1111 1110 0010

Conditional Statements

1. if
2. else

```
if(expression)
    statement;
```

⇨ A single statement.

```
if(expression){
    statement1;
    statement2;
    }
```

⇨ A block of statements.

```
if(expression)
        statement;
else    statement;
```

⇨ Single statement in the if and a single statement in the else.

```
if(expression)
        statement;
    else
    {
    statement1;
    statement2;
    }
```

⇨ A single statement in the if and a block of statements in the else.

Conditional statements

1. else-if

```
if(expression)
            statement;
      else
   if(expression)
            statement;
      else
            statement;
```

⇨ A single statement in the if,else-if and in the else block.

- The nested if can become complicated and unreadable.
- The switch statement is an alternative to the nested if.
- Usually, but not always, the last statement of a case is break.
- default case is optional.

2. switch

```
Switch(expression)
{
case constant expr:
        statement(s);
        break;
case constant expr:
        statement(s);
        break;
case constant expr:
        statement(s)
        break;
default :
        statement(s);
        break;
}
```

Loops

- Loops break the serial execution of the program.
- A group of statements is executed a number of times.
- There are three kinds of loops :
  1. **for**
  2. **while**
  3. **do … while**

```
for (expr1 ; expr2 ; expr3)
    statement;
```

or

```
for (expr1 ; expr2 ; expr3)
{
    statements;
}
```

- Is equivalent to:

```
expr1;
while (expr2)
{
    {statements;}  expr3;
}
```

```
do
{
   Statements;
}while (expression);
```

The condition expression for looping is evaluated only after the loop body had executed.

break and continue statement

1. break statement :

- We have seen how to use the break statement within the switch statement.
- A break statement causes an exit from the innermost containing while, do, for or switch statement.

1. continue statement :

- In some situations, you might want to skip to the next iteration of a loop without finishing the current iteration.
- The **`continue`** statement allows you to do that.
- When encountered, **`continue`** skips over the remaining statements of the loop, but **`continues`** to the next iteration of the loop.

Pascal Case Coding Conventions

- Example
  - System
  - StringBuilder
  - NullPointerException
  - IndexOutOfBoundsException

- In this case, including first word, first character of each word must in upper case.

- We should use this convention for:
  - Type Name(Interface, class, Enum, Annotation)
  - File Name

Camel Case Coding Conventions

- Example
  - main
  - parseInt
  - showInputDialog
  - addNumberOfDays

- In this case, excluding first word, first character of each word must in upper case.

- We should use this convention for:
  - Method Parameter and Local variable
  - Field
  - Method
  - Reference

Widening

- Process of converting value of variable of narrower type into wider type is called widening.
- E.g. Converting int to double
- In case of widening, there is no loss of data
- So , explicit type casting is optional.

```
public static void main(String[] args) {
    int num1 = 10;
    //double num2 = ( double )num1;    //Widening : OK
    double num2 = num1;    //Widening : OK
    System.out.println("Num2    :    "+num2);
}
```

Narrowing (Forced Conversion)

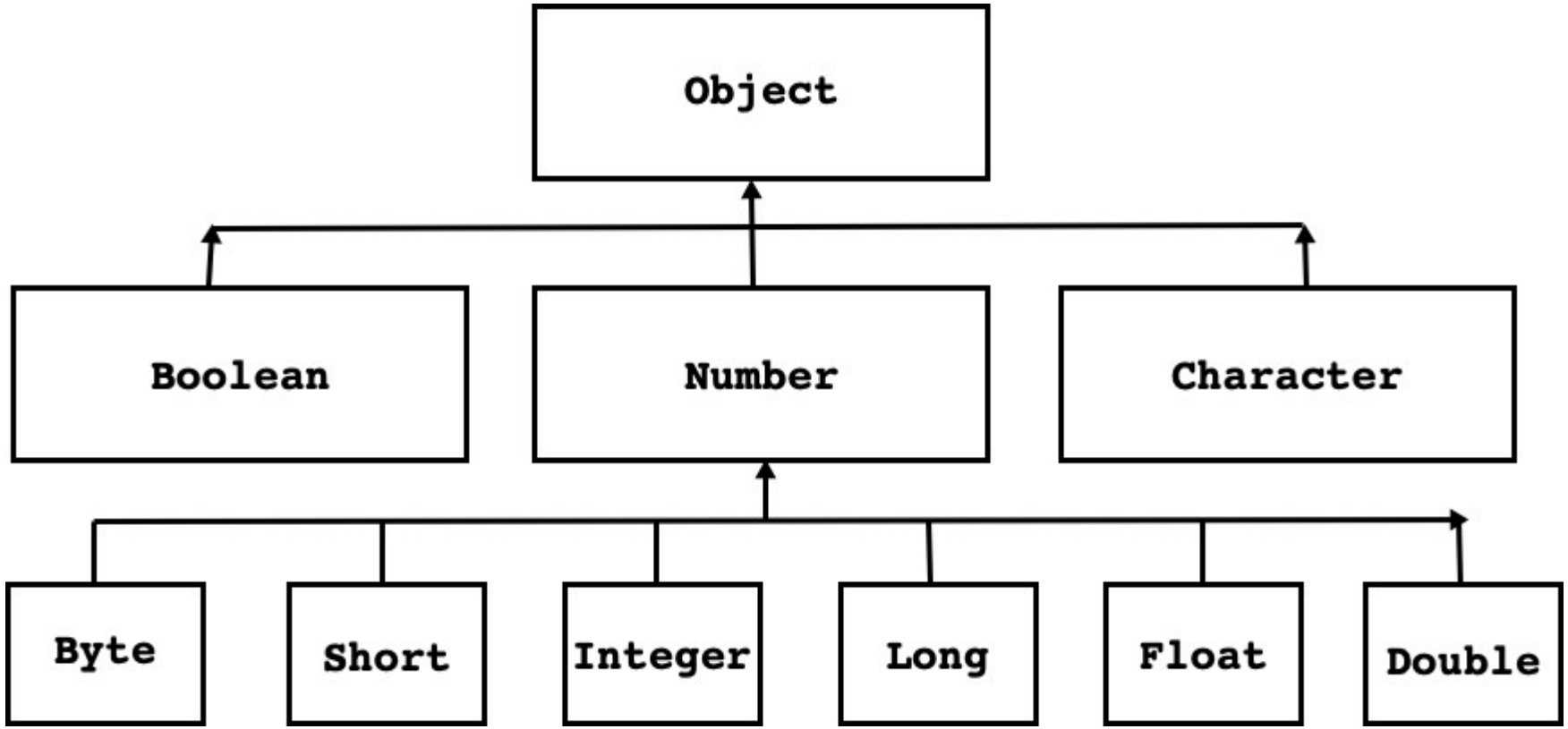
- Process of converting value of variable of wider type into narrower type is called narrowing.
- In case of narrowing, explicit type casting is mandatory.
- **Note : In case of narrowing and widening both variables are of primitive**

```
public static void main(String[] args) {
    double num1 = 10.5;
    int num2 = ( int )num1;  //Narrowing : OK
    //int num2 = num1;  //Narrowing : NOT OK
    System.out.println("Num2     :    "+num2);
}
```

Wrapper class

- In Java, primitive types are not classes. But for every primitive type, Java has defined a class. It is called wrapper class.
- All wrapper classes are final and are declared in java.lang package.
- Uses of Wrapper class
  - To parse string(i.e. to convert state of string into numeric type).
  - example :
  - int num = Integer.parseInt(“123”)
  - float val = Float.parseFloat(“125.34f”);
  - double d = Double.parseDouble(“42.3d”);
- To store value of primitive type into instance of generic class, type argument must be wrapper class.
  - Stack<int> stk = new Stack<int>( ); //Not OK
  - Stack<Integer> stk = new Stack<Integer>( ); //OK

Wrapper class



Command line argument

```
class Program{
    public static void main( String[] args ){
        int num1      = Integer.parseInt(args[0]);
        float num2    = Float.parseFloat(args[1]);
        double num3   = Double.parseDouble(args[2]);
        double result = num1 + num2 + num3;
        System.out.println("Result : "+result);
    }
}
```

+ User input from terminal:
    - java Program 10 20.3f 35.2d (Press enter key)

Thank you!

Rohan Paramane

rohan.paramane@sunbeaminfo.com