OPENTECHZ

A dream to realize your dream

Core Java
with
Project Development

By

Parthasarathi Swain

Keywords

➢ A predefined identifier that has special meaning in java program outside comments and string is called a keyword.

➢ A keyword is a reserved word in java language inside compiler and jvm that performs a unique and special operation.

Rules:

- We can't use a keyword as user defined identifier.
- We must use keyword with all characters in lower case.

Keywords

keyword vs reserved word

- A word that is created as part as compiler and jvm software to perform one operation is called keyword .
  EX : public , static ..etc

- A word that is created as part of compiler and jvm software to represent a value is called reserved word.
  EX : true , false , null

Keywords

keyword operations

1. Package Creation.
2. Class Creation.
3. Data type and return type.
4. Memory Allocation.
5. Control Flow.
6. Accessibility Modifiers.
7. Execution Level Modifiers.
8. Establishing Relation.
9. Object and Member.
10. Exception Handling.

Keywords

To Perform Those 10 Operation java Supports 51 keywords.

Among of those 51 keywords

1. **Jdk .1.0 we have 47 keywords .**
2. **Jdk .1.2 we got a new keyword Strctfp .**
3. **Jdk .1.4 we got a new keyword assert .**
4. **Java 5 we got a new keyword enum .**
5. **Java 9 we got a new keyword '_'.**

Keywords

1. Package Creation
   - package
   - import
2. Class Creation.
   - class
   - interface
   - enum
3. Data type and return type.
   - byte
   - short
   - int
   - long
   - double
   - boolean
   - char
   - void (return type)
4. Memory Allocation.
   - static
   - new
5. Control Flow.
   - Conditional : if ,else ,switch ,case ,default
   - Loop : do , while , for
   - Branching : break ,continue ,return
6. Accessibility Modifiers.
   - private
   - public
   - protected
7. Execution Level Modifiers.
   - static
   - final
   - abstract
   - native
   - volatile
   - synchronised
   - strictfp

Keywords

8. Establishing Relation.

- Extends
- implements

9. Object and Member.

- this
- super
- instanceOf

10. Exception Handling.

- throws
- throw
- try
- catch
- finally
- assert

Unimplemented Keywords

- goto
- const

Java 9 new keyword ('_')

Escape Sequence

- A character with a backslash (\) just before it is an escape sequence or escape character.
- We use escape characters to perform some specific task.
- The total number of escape sequences or escape characters in Java is 8.

List of Escape Sequence

- \f represent a symbol within partha and swain.
  Ex : System.out.println("Partha\fSwain");

- \b means, it deletes \b before character.
  Ex : System.out.println("Partha\bSwain");

- \n refers for create a new line between partha and swain.
  Ex : System.out.println("Partha\nSwain");

- \t it creates a tab space .
  Ex : System.out.println("Partha\tSwain");

- \r means it replace at starting point(for example Swain placed instead of Parth)
  Ex : System.out.println("Partha\rSwain");

List of Escape Sequence

- System.out.println("Partha\'Swain");
- System.out.println("Partha\"Swain");
- System.out.println("Partha\\Swain");
- System.out.println("Partha\0Swain");
- System.out.println("Partha\1Swain");
- System.out.println("Partha\2Swain");
- System.out.println("Partha\3Swain");
- System.out.println("Partha\4Swain");
- System.out.println("Partha\5Swain");
- System.out.println("Partha\6Swain");
- System.out.println("Partha\7Swain");

Data Types

Need:

- Data types are used to store data temporarily in computer through a program.
- In real world we have different type of data like integer, floating- point, character, boolean and String..etc.
- To store all these data in program to perform business required calculation and validation we must use data types.

Def :

Data types is something which gives information about -size of the memory location and range of data that can be accommodation inside that location

Data Types

Type:

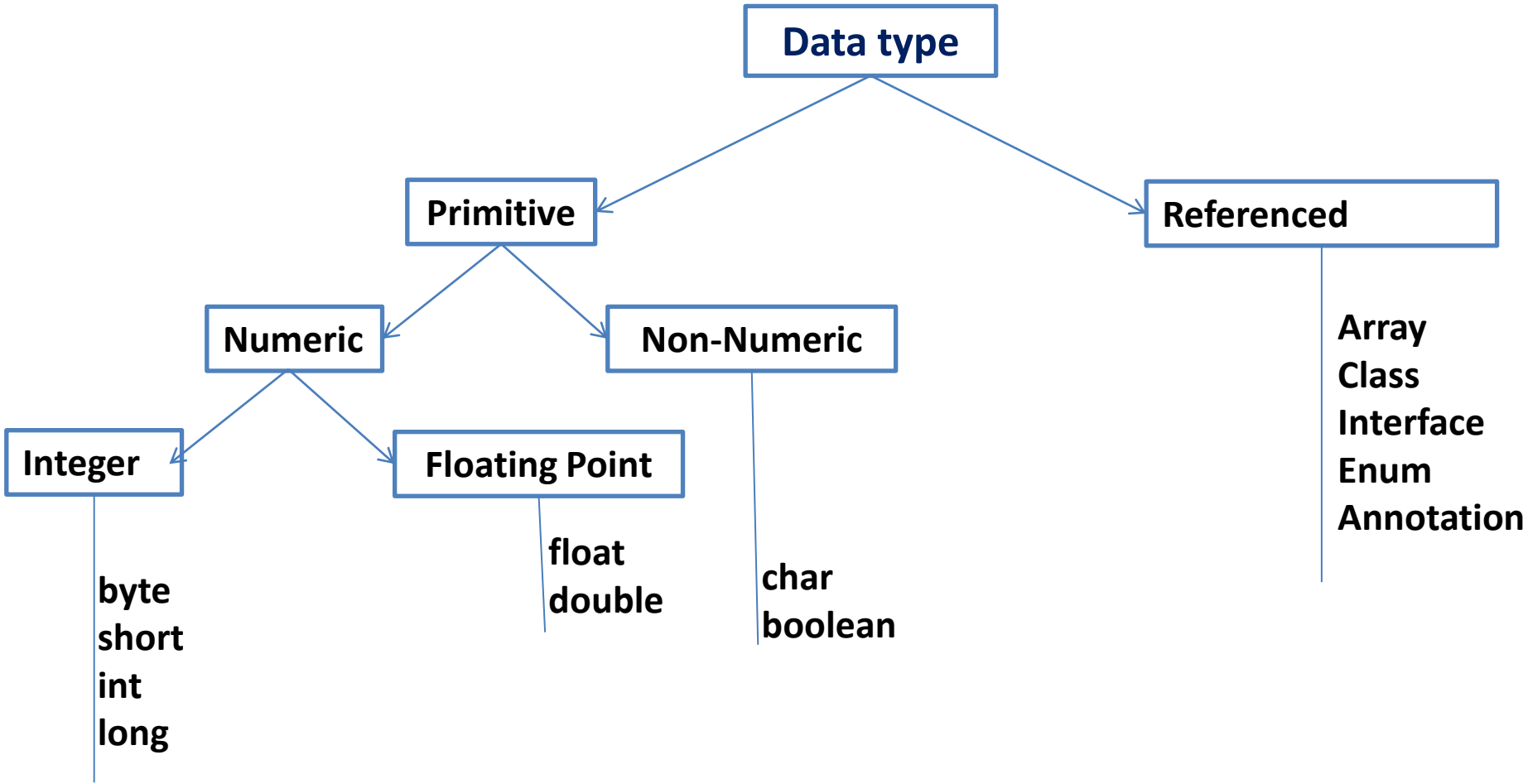
in java mainly we have two types of data types.

1. Primitive types (8) -used for storing single value at a time.

2. Referenced types (5)-used for storing multiple values of primitive types.

Primitive	Referenced
byte	Array
short	Class
int	Interface
long	Enum
char	Annotation
float	
double	
boolean	

Data Types


Data type
Primitive
Referenced
Numeric
Non-Numeric
Integer
Floating Point
byte
short
int
long
float
double
char
boolean
Array
Class
Interface
Enum
Annotation

Data Types

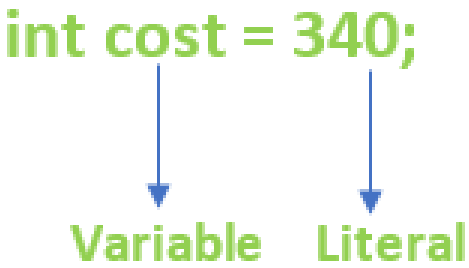
data types name	size(byte)	range	default
byte	1	-128 to +127	0
short	2	-32,768 to +32,767	0
int	4	-2 ,147,483,648 to +2 ,147,483,647	0
long	8	-9,223,372,036854775808 to +9,223,372,036854775807	0
float	4	1.40239846e-45f to 3.40282347e+38f	0.0f
double	8	4.94065645841246544e-324 to 1.79769313486231570e+308	0.0d
char	2	0 to 65,535	1 white space
boolean	1	true and false	false
Reference	depends on PDT	depends on PDT	null

Literals

Literals are data used for representing fixed values. They can be used directly in the code.

For example:

```
int a = 1;
float b = 2.5f;
char c = 'F‘;
```



Here, 1, 2.5f, and 'F' are literals.

Literals

Types of Literals in Java

There are the majorly **5** types of literals in Java:

1. **Integer Literal**
2. **Floating point Literal**
3. **Character Literal**
4. **Boolean Literal**
5. **String Literal**

Literals

1)Integral Literals:

- All integer type literals are called integral literals.

- By default they are of type int. If we want to represent them as long we must suffix literal with ‘l’ or 'L'.

- We do not have byte or short type literals.

EX:

```
Int  x=70;
```

Literals

2)Floating-point Literals:

- All floating point literals are of type double.

- If we want to represent them as float we must suffix literal with 'f or 'F'.

- double literals can also be suffixed with 'd' or 'D'.

Ex:

```
Float f=3.4f;
Double d=3.4d;
```

Literals

3)Character literals:

Character literals are Unicode character enclosed inside single quotes.
For example,
char letter = 'a';

Literals

4) Boolean Literals:

In Java, boolean literals are used to initialize boolean data types. They can store two values: true and false.

For example,

```
boolean flag1 = false;
boolean flag2 = true;
```

Here, false and true are two boolean literals.

Literals

5) String literals

A string literal is a sequence of characters enclosed inside double-quotes.

For example,

```
String  str1 = "Java Programming";
String  str2 = “OTZ";
```

Literals

Identify valid literals from the below list

- 10
- 10.345
- 53.67f
- 2345L
- 3.45d
- 45D
- 20b
- 34s
- 'a'
- a
- '#'
- '1'
- '10'
- ''
- ""
- "abc"
- "10"
- "1"
- "a"
- hi

Way of assigning one type of primitive to other primitive type is classified as 2 categories

1.Widening(Auto or implicit)
2.Narrowing(Explicit)

1.Widening

In this type, we will **assign smaller type to larger type**.

Example:

short=byte
int=short
long=int
float=long
double=float

```
public class Autowidening{

    public static void main(String[] args) {

        int i = 100;
        long l = i; // no explicit type casting is required
        float f = l; // no explicit type casting required
        System.out.println("i= " + i);
        System.out.println("l= " + l);
        System.out.println("f= " + f);

    }

}
```

2.Narrowing(Explicit)

In this type, we will **assign a larger type value to a variable of smaller type**.

Example:

byte=short

short=int

int=long

long=float

float=double

```
public class ExplicitNarrowing {

    public static void main(String[] args) {

        double d = 100.01;
            // long l=d;  compile time error as we are assigning larger type to smaller type
                                                                                    without casting
        long l = (long) d; // explicit type casting is required
        int i = (int) l; // explicit type casting is required

        System.out.println("i= " + i);
        System.out.println("l= " + l);
        System.out.println("d= " + d);

    }

}
```

```
int a = 10;
byte b = a;
byte b = (byte)a;
boolean  bo = a;
boolean  bo = (boolean)a;
boolean  bo = (byte)a;
byte b = (short) a;
short s = (byte)a;
byte b = (short)(byte)a;
byte b = (byte)(short)a;
```

```
int a = 254;

byte b1 = (byte) a;
short s1 = (short) a;

short s2 = (short)(byte) a;

System.out.println(a);
System.out.println(b1);
System.out.println(s1);
System.out.println(s2);
```

Literals

Types of Literals in Java

There are the majorly **5** types of literals in Java:

1. **Integer Literal**
2. **Floating point Literal**
3. **Character Literal**
4. **Boolean Literal**
5. **String Literal**

Literals

1)Integral Literals:

- All integer type literals are called integral literals.

- By default they are of type int. If we want to represent them as long we must suffix literal with ‘l’ or 'L'.

- We do not have byte or short type literals.

EX:

```
Int  x=70;
```

Literals

2)Floating-point Literals:

- All floating point literals are of type double.

- If we want to represent them as float we must suffix literal with 'f or 'F'.

- double literals can also be suffixed with 'd' or 'D'.

Ex:

```
Float f=3.4f;
Double d=3.4d;
```

Literals

3)Character literals:

Character literals are Unicode character enclosed inside single quotes.
For example,
char letter = 'a';

Literals

4) Boolean Literals:

In Java, boolean literals are used to initialize boolean data types. They can store two values: true and false.

For example,

```
boolean flag1 = false;
boolean flag2 = true;
```

Here, false and true are two boolean literals.

Literals

5) String literals

A string literal is a sequence of characters enclosed inside double-quotes.

For example,

```
String  str1 = "Java Programming";
String  str2 = “OTZ";
```

Literals

Identify valid literals from the below list

- 10
- 10.345
- 53.67f
- 2345L
- 3.45d
- 45D
- 20b
- 34s
- 'a'
- a
- '#'
- '1'
- '10'
- ''
- ""
- "abc"
- "10"
- "1"
- "a"
- hi

Way of assigning one type of primitive to other primitive type is classified as 2 categories

1.Widening(Auto or implicit)
2.Narrowing(Explicit)

1.Widening

In this type, we will **assign smaller type to larger type**.

Example:

short=byte
int=short
long=int
float=long
double=float

```
public class Autowidening{

    public static void main(String[] args) {

        int i = 100;
        long l = i; // no explicit type casting is required
        float f = l; // no explicit type casting required
        System.out.println("i= " + i);
        System.out.println("l= " + l);
        System.out.println("f= " + f);

    }

}
```

2.Narrowing(Explicit)

In this type, we will **assign a larger type value to a variable of smaller type**.

Example:

byte=short

short=int

int=long

long=float

float=double

```
public class ExplicitNarrowing {

    public static void main(String[] args) {

        double d = 100.01;
            // long l=d;  compile time error as we are assigning larger type to smaller type
                                                                                    without casting
        long l = (long) d; // explicit type casting is required
        int i = (int) l; // explicit type casting is required

        System.out.println("i= " + i);
        System.out.println("l= " + l);
        System.out.println("d= " + d);

    }

}
```

```
int a = 10;
byte b = a;
byte b = (byte)a;
boolean  bo = a;
boolean  bo = (boolean)a;
boolean  bo = (byte)a;
byte b = (short) a;
short s = (byte)a;
byte b = (short)(byte)a;
byte b = (byte)(short)a;
```

```
int a = 254;

byte b1 = (byte) a;
short s1 = (short) a;

short s2 = (short)(byte) a;

System.out.println(a);
System.out.println(b1);
System.out.println(s1);
System.out.println(s2);
```

Operators

What are the different operation perform by using operator ?

1. Assignment
2. Calculation
3. Validation

What is an operator ?

Operator in Java is a symbol that is used to perform calculation , validation ,assignment operation is called operator .

```
Int  x=10;                ( = is an assignment operator )
If(x>0){                  ( > is  a validation operator  )
  Sopln(x+20);            ( +  is a calculation operator  )
}
```

Operators

What is an Operand ?

A value for giving inputs to an operator to perform vals ,Cals ,Assigns

```
x=30;
y=90;
int  z=  x  +  y   ;
```

What is an expression?

Combination of operator and operands .

```
int  z=   x   +  y    ;
```

Operators

How many operators avilable in java...?
Ans- 40 .

Types
---
based on operation performing
we have 3 types of operation
1) validation operators --> < ,>,<=,>=,==,!= ,&&, || ...etc
2) Calculation operators --> +,-,*,/,%...etc
3) Assignment operators --> =

Based on operands
1) Unary operators(11)
2) Binary operators (27)
3) Ternary operators (2)

Operators

a)Assignment operators	(1)	=
b)Arithmetic operators	(5)	+ - * / %
c)increment decrement	(2)	++ --
d)Relation operators	(5)	< > <= >= instanceof(object type checking operator)
e)Logical operators	(3)	&& || !
f)Bitwise operators	(4)	& | ^ ~
g)Shift operators	(3)	<< >> >>>
h)ternary operator	(2)	? :
i)Compound Assignment	(11)	+= -= /= %= &= |= ^= <<= >>= >>>=
j)object creation	(1)	new
k)Lambda operator	(1)	->
l) Equality operator	(2)	== !=
	40	

Operators

Separator:
(one of)

```
(   )   {   }   [   ]   ;   ,   .   ...   @   ::
```

Operators

Operators	Precedence
postfix, prefix, unary	`expr++ expr-- ++expr --expr +expr -expr ~ ! new`
multiplicative	`* / %`
additive	`+ -`
shift	`<< >> >>>`
relational	`< > <= >= instanceof`
equality	`== !=`
bitwise AND	`&`
bitwise exclusive OR	`^`
bitwise inclusive OR	`|`
logical AND	`&&`
logical OR	`||`
ternary	`? :`
Lamdba Expression	`->`
assignment	`= += -= *= /= %= &= ^= |= <<= >>= >>>=`

Operators

Arithmetic operator

how many
we have 5 AO's --> + - * / %
Used
for performing calculation addition,substration, multiplication and division(finding quotient and reminder).
Type
binary operators --> takes two operands
rules--> allows only numeric data types--> byte,short,int,long,float,double,char.
--> we can't use boolean data type as operand

result type --> numeric --> based on the data types we used in calculation
--> always we will get highest range data type as result.

Operators

Arithmetic operator result type

byte+byte -->int
short+short -->int
char+char -->int
int+int -->int
long+long -->long
float+float -->float
double+double -->double

Operators

Arithmetic operator Expression result type

byte + short + char ===> int
int + short + byte + char ===>int
int + long ===> long
float + long ===> float
double +long ===> double
float + int + long + char ===> float
float + int + double + long + char ===> double

Operators

Compiler algorithm in an expression evaluation W.r.t literals and Variables

We have Two types of expression.

1) Content expression-->if we use only literals or only final variables or both is called constant expression .

2) Variables expression->if we use at least one variable or non void method call we call this expression as variable expression.

Operators

Content expression

if we use only literals or only final variables or both,
compiler directly uses value and generates final results
and verifies is this value i the range of destination
variable.

-if Yes, compiled fine.
-if No, it throws possible lossy conversion.

Operators

+ operator special case

==========

+ operator is a overloaded operator.

1)Addition operation --> if both operand are numeric or char, it add and join value

2)Concatenation operation -->at least one operand is String , then it act as concatenation operation. (it will join/append both operand values and generates new value)

Operators

Division operator special point

> int/int ==> int
> integer/0 ==> AE
> floating point num/0 ==> infinity
> -floating point num/0 ==> -infinity
> 0/0 --> AE
> 0.0/0 --> NaN
> -0.0/0 --> NaN

NaN-: Not a number

Operators

Increment and Decrement

Post Inc (exp++)

First use
Then inc

Pre Inc(++exp)

First inc
Then use

Post Dec (exp--)

First use
Then dec

Pre Dec (--exp)

First dec
Then use

Operators

```
int x=1;
int y= x++   +  ++y ;
```

```
int p= 34;
int q= p++ + ++p + p++ + p++ + --p  +  p -- ;
```

```
int a=78;
int b= ++a  + --a  -  a--  + a++  -  ++a ;
```

```
int s=65;
int r= s++ + ++s + s++ + s++ + --s  -  ++s ;
```

```
int o=0;
o++;
int i=  o++ + --o + o--  +  o++ + o++ + ++o ;
```

```
int  c=0+4;
++c;
int d= ++c  -  --c  +  ++c  +  ++c  + c++ ;
```

Operators

Java Unary Operator Example: ~ and !

```
public class OperatorExample{
public static void main(String args[]){
int a=10;
int b=-10;
boolean c=true;
boolean d=false;
System.out.println(~a);//-11 (minus of total positive value which starts from 0)
System.out.println(~b);//9 (positive of total minus, positive starts from 0)
System.out.println(!c);//false (opposite of boolean value)
System.out.println(!d);//true
}}
```

Operators

Java Left Shift and Right Shift Operator Example

```
public class OperatorExample{
public static void main(String args[]){
System.out.println(10<<2);//10*2^2=10*4=40
System.out.println(10<<3);//10*2^3=10*8=80
System.out.println(20<<2);//20*2^2=20*4=80
System.out.println(15<<4);//15*2^4=15*16=240
}}
```

```
public OperatorExample{
public static void main(String args[]){
System.out.println(10>>2);//10/2^2=10/4=2
System.out.println(20>>2);//20/2^2=20/4=5
System.out.println(20>>3);//20/2^3=20/8=2
}}
```

Operators

Java Left Shift and Right Shift Operator Example

```
public class OperatorExample{
public static void main(String args[]){
System.out.println(10<<2);//10*2^2=10*4=40
System.out.println(10<<3);//10*2^3=10*8=80
System.out.println(20<<2);//20*2^2=20*4=80
System.out.println(15<<4);//15*2^4=15*16=240
}}
```

```
public OperatorExample{
public static void main(String args[]){
System.out.println(10>>2);//10/2^2=10/4=2
System.out.println(20>>2);//20/2^2=20/4=5
System.out.println(20>>3);//20/2^3=20/8=2
}}
```

Operators

Java Ternary Operator Example

```
public class OperatorExample{
public static void main(String args[]){
int a=2;
int b=5;
int min=(a<b)?a:b;
System.out.println(min);
}}
```