Java Basics

Data Types

- Java is a strongly typed language.
- Every variable and expression has a type.
- All types are well defined.
- Java strictly checks type compatibility while doing assignments.

Data Types

- Java defines eight primitive types - byte, short, int, long, float, double, char, boolean
- These can be put in four groups:

  Integers : byte, short, int, long

  Floating points numbers : float, double

  Characters : char

  Boolean : boolean

Data Types

- The primitive type represents single values and not complex object.

- Unlike c/c++ (which depends upon the execution environment), Java has strictly defined range for all primitive types. For example size of an int type data in C/C++ may vary but not in Java to achieve portability.

Data Types

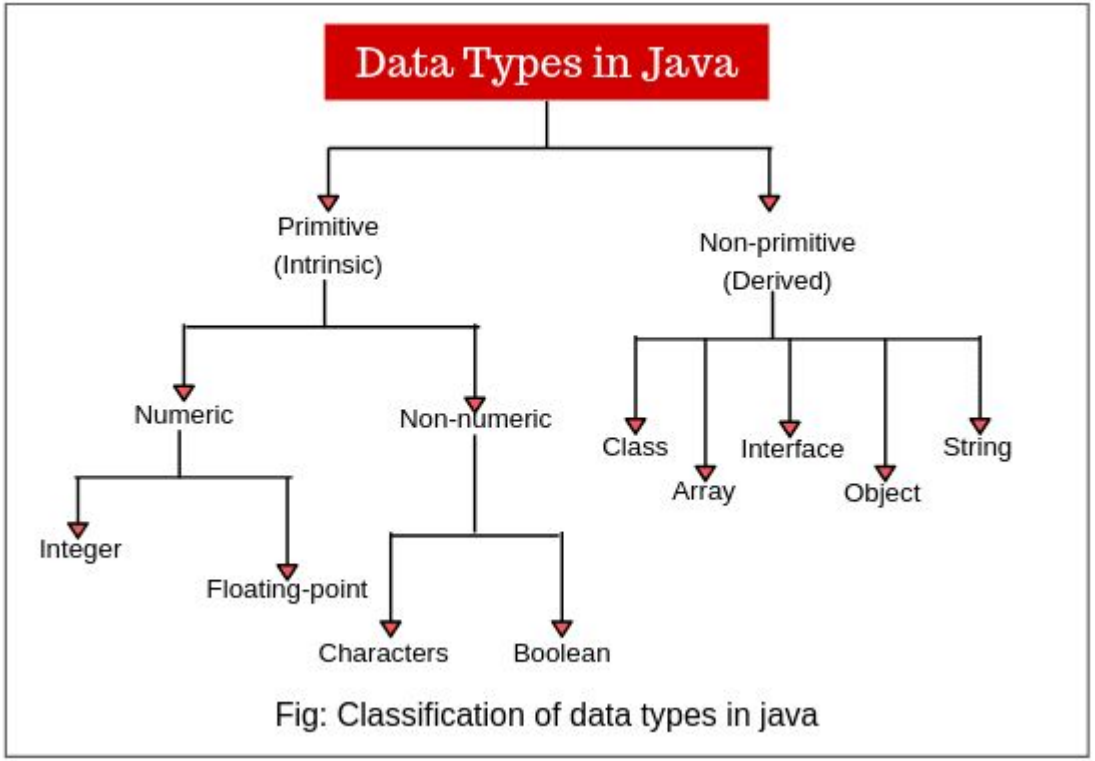

Fig: Classification of data types in java

Data Types

Data Type	Size	Range of values that can be stored	Default value
byte	1 byte	−128 to 127	0
short	2 bytes	−32768 to 32767	0
int	4 bytes	−2,147,483,648 to 2,147,483,647	0
long	8 bytes	-9,223,372,036,854,775,808 to 9223372036854750000	0
float	4 bytes	3.4e−038 to 3.4e+038	0.0f
double	8 bytes	1.7e−308 to 1.7e+038	0.0d
boolean	1 bit	true or false	false
char	2 bytes		\u0000

Data Types

- Java does not support unsigned positive-only integers as many other languages do. Ex. in C signed int, unsigned int etc.

- The java run-time environment is free to use whatever size it wants, as long as the types behave as you declared them. Infact one implementation stores bytes and shorts as 32 bit (rather than 8-bit and 16-bit) values to improve performance because that is the word size of most computers currently.

Data Types

Use of byte

- Stream of data over network
- Working with raw binary data not compatible with other built-in types.

Use of short

- It is probably the least-used java type, mostly useful in 16 bit computers.

Data Types

Use of int

- It may seem that using short or byte will save space, but there is no guarantee that Java won’t promote those types to int internally anyway. Remember, type determines behavior, not size.
- The only exception is arrays, where byte is guaranteed to use only one byte per array element, short will use two bytes, and int will use four.

Data Types

Use of long

- When range is out of int type, use long. Ex. Calculate distance travelled by light in a given number of days (e.g. 10000). Please note that light travels 186000 miles approx in a second.

Use of float

- To store the real numbers that require fractional precision. Type float specifies a single-precision value that uses 32 bits of storage.
- Single precision will become imprecise when the values are either very large or very small.
- Use float when you don’t require a large degree of precision.

Data Types

Use of double

- It specifies a double-precision value.
- Use double when you need many iterative calculations (like sqrt, sin, cos etc.) or manipulate large valued numbers.
- The extra bits in double, increase not only the precision but also the range of magnitudes that can be represented.

Data Types

Use of char

- Java uses Unichar to represent characters which is an international character set including all the characters found in all human languages.
- Standard set of characters ASCII range : 0 to 127
- Extended 8-bit character set, ISO-Latin-1 range : 0 to 255
- First 127 values in the Unicode character set is same as ASCII set.

Data Types

- Java allows to add two characters and increment the value of character.

  Ex: char ch = ‘J’;

  ch ++;

  S.o.p(ch); //Will print K

Data Types

Use of boolean

- To store logical values (true or false)

  Ex. boolean b = false;

- Boolean is also the type required by the conditional expressions that govern the control statements used in if and for.
- Outcome of a relational operator is a boolean value.

  Ex. boolean res = (a > b);

Variables

Variables

- Basic unit of storage in a program.
- Basic form of variable

  type identifier [= value];

  Where the identifier is the name of the variable.

- Ex. int num = 100, char ch, String name etc.
- We can not use Java keywords as a variable name.

Java Keywords

abstarct	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

Variables

Scope of a variable

- Scope - Begins with opening curly bracket and ends with closing curly bracket.
- The variables declared in the outer scope will be visible to code within inner scope. However the reverse is not true.
- Within a block, variables can be declared at any point.
- Variables are created when their scope is entered and destroyed when their scope is left.

Variables

Scope of a variable

Although blocks can be nested, we can not declare a variable to have the same name as one in an outer scope.

```
Class ScopeDemo
{
P.s.v.m. ( String args [ ] )
{       int num = 10;
        {
                int num = 20; // Compile-time error, num is already defined.
        }
}
}
```

Variables

Scope of a variable

- A variable must be initialized (not necessarily at the time of declaration) before its use in the code otherwise the compiler will give an error “variable might not have been initialized”.

Literals

Literals

- Literals are numbers, text, or anything that represent a value. In other words, Literals in Java are the constant values assigned to the variable. Ex. 235, “Hello”, ‘Y’ etc.

Literals

Integer literals

- Integer literals in java can be represented in any of the following form.

  Decimal (Base 10) - Ex . 93

  Octal (Base 8) - Ex. **0**135 (Leading with zero)

  Hexadecimal (Base 16) - Ex. **0x**5D (Leading with zero and small/capital X)

Literals

Integer literals Contd..

- To assign a long value you need to explicitly tell the compiler that it is a long value by appending lowercase or uppercase L. ex. long p = 0xffffffffL or long p = 23654777788L
- By **default** each integer literal type is treated as **int**.

Literals

Floating point literals

Floating point literal in java default to double precision.

To specify a float literal, you must append f or F to the constant.

Boolean literals:

In java true does not mean 1 and false does not mean 0.

Literals

Character literals:

- A character literal is represented by using single quote. Ex. ‘A’, ‘a’ etc.
- Using escape sequence to represent special characters like ‘\’’ to represent single quote.
- \n for newline character
- Representing character using octal or hexadecimal values. Ex. ‘\141’ (three digit Octal for ‘a’), ‘\u0061 (4 digit hexadecimal for ‘a’, see \u for hexadecimal).

Literals

String literals

- Enclosing sequence of characters between pair of double quotes.
- Ex. “Hello World”, “Line\nBreak” , “\”Quotes\”” etc.In java String must begin and end on the same line.
- There is no line-continuation escape sequence.

  Ex.

  “Hello

  World”

  Is wrong.

Literals

Escape Sequences

\ddd - Octal character

\uxxxx - Hexadecimal unicode character

\’ - single quote

\” - double quote

\\ - backslash

\n - new line (line feed)

Literals

Escape Sequences

\t - Tab

\b - backspace

Type Conversion

Type Conversion and Casting

- It is common to assign value of one type to another type. Ex. assigning int value to a long variable.
- If the two types are compatible, java will perform automatic conversion. Ex. int to long (Automatic type conversion by java - implicit)
- It is possible to obtain conversion between incompatible types. Ex. double to byte (Typecasting - explicit)

Type Conversion

Automatic type conversion (Implicit Conversion) conditions

- The two types are compatible.
- Destination type is larger than the source type. (Also called, Widening Conversion). Ex. Assigning byte to int as int type is always large enough to hold byte values.
- All numeric types (int, float, double) are type compatible with each other for widening conversion.
- Ex.
  int a ; byte b = 10 ; a = b;

Type Conversion

Explicit Conversion (Typecasting)

Syntax:
(Target-Type) value

Assigning larger type to smaller type. (Also called, Narrowing Conversion). Ex. Assigning int to byte, float to int.

Ex. int a = 10;

byte b;

b = (byte) a;

Type Conversion

- If the size of the whole number component (Integer values long, int, short) is too large to fit into the target integer type, then that value will be the reduced modulo the target type's range .

  Ex.

  int i = 258;

  byte b ;

  b = (byte) i ;

  s.o.p(i); // Prints 258

  s.o.p(b); // Prints 2 (remainder) i.e. result of modulo operation 258 % 256 = 2

Type Conversion

- When a double value is converted to int, its fractional component is lost.

  Ex.

  int i ;

  double d = 258.73;

  i = d;

  S.o.p (i); // Prints 258

Type Conversion

- When a double value is converted to byte, its fractional component is lost and value is reduced modulo 256.

  Ex.

  byte b ;

  double d = 258.73;

  b = (byte) d;

  S.o.p (b); // Prints 2 (remainder) i.e. result of modulo operation 258 % 256 = 2

Type Conversion

- When a double value is converted to float

float f = 85.0;

Error: incompatible types: possible lossy conversion from double to float.

float f = 85.0f;

Type Conversion

- In the widening conversion, high order bits of target type is filled up with the highest bit of the given value. This way, the value's sign is retained.

  byte (2) => 00000010
  Short (2) => 00000000 000000010

- Exception is widening of char. As char are unsigned, high order bits of target type is filled up with 0.

  (char) \uFFFF => 11111111 11111111
  (int) 65535 => 00000000 00000000 11111111 11111111

Type Conversion

- In the narrowing conversion, where the original type has more bits than the target type, the additional bits are merely cut off.

  Short (2) => 0000001 00000100 // 260
  byte (1) => 00000100 // 4

  (int) 65535 => 00000000 00000000 11111111 11111111
  (char) \uFFFF => 11111111 11111111

Type Promotion

- If an expression contains multiple type variables, the type of each variable is first promoted to the largest type variable in the expression and also evaluates the result to that largest type.

- Java promotes each byte or short operand to int when evaluating the expression.

  Ex. 1.

  byte a = 20, b = 30, c = 100;

  int res = a * b * c; // a, b, c promoted to int during evaluation

Identifiers

Identifiers

A name in the program is an identifier it may be class name or method name, variable name or label name.

Example

```
Class Demo{

public static void main(String[] args) {

int a = 5;

System.out.println(a);

}

}
```

Identifiers

Rules for defining Identifiers

- Identifiers cannot start with a number.
- Identifiers must start with a letter, a currency character ($), or a connecting character such as the underscore (_)
- After the first character, identifiers can contain any combination of letters, currency characters, connecting characters, or numbers.
- In practice, there is no limit to the number of characters an identifier can contain.
- You can't use a Java keyword as an identifier.
- Identifiers in Java are case-sensitive; num and NUM are two different identifiers.
- Upper camel case ex: MyClass
- Lower camel case ex: myClassObj

Identifiers

Name	Convention
class name	should start with uppercase letter and be a noun e.g. String, Color, Button, System, Thread etc.
interface name	should start with uppercase letter and be an adjective e.g. Runnable, Remote, ActionListener etc.
method name	should start with lowercase letter and be a verb e.g. actionPerformed(), main(), print(), println() etc.
variable name	should start with lowercase letter e.g. firstName, orderNumber etc.
package name	should be in lowercase letter e.g. java, lang, sql, util etc.
constants name	should be in uppercase letter. e.g. RED, YELLOW, MAX_PRIORITY etc.