**Operators**

Primitives and their default value.

**boolean bool**; *// true or false***byte bite**; *// 8 bit integral value***short sort**; *// 16 bit integral value***char car** ; *// 16 bit Unicode value***int inty**; *// 32 bit integral value***float froat**; *// 32 bit floating point value // 123.f;***long lawng**; *// 64 bit integral value // 123L or 123;***double dobble**; *//64 bit floating point value;*@Test  
**public void** testPrimitiveDefaultValues(){  
 *assertTrue*(**bool** == **false**);  
 *assertTrue*(**bite** == 0);  
 *assertTrue*(**sort** == 0);  
 *assertEquals*(**car**, **'\u0000'**);  
 *assertTrue*(**car**==0);  
 *assertTrue*(**inty** == 0);  
 *assertTrue*(**froat** == 0.0);  
 *assertTrue*(**lawng** == 0.0);  
 *assertTrue*(**dobble** == 0);  
}

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

* Int Y = 4;

Double x = 3 + 2 \* --y;

The prefix operator would run first setting y to 3 and then 3 would be multiplied with 2 and then 6 added with 3, the result would be **9.0.** Don’t forget Decimal and Floats have a default value of 0.0.

Now if 3+2 was in a parenthesis, then that would take precedence and the result would be 5 \* 4= **20.0**

* If two values have different data types, java will automatically promote one of the variable to the larger of the two-data type.

5 == 5.0 // this is true because java promotes the left hand to float automatically.

* Smaller data types **byte, short and char** are automatically converted to integer
* Short s1 = 4, s2=3;

Short s = s1+s2; **// !!COMPILER error** because the result is an integer. However, s += s1; would work because that would automatically cast the result to short.

Long x = 10; int y =5;

Y = x \* y; **// !! Compiler error** because the result is upcasted to long and the variable is assigned to int.

* Floats are tricky, if you have a float and it has a decimal you must have the word f in the end.

**float fl** = 1.0f; Valid

**float fl = 1.0; // ERROR**

float fl = 1l; // Valid

**Float f1 = 1/ // ERROR**, for Wrapper type Float you need “f” at the end.

Float f1 = 1f// Valid

* Java will throw compiler exception if it detects trying to convert from larger to smaller data types.
* For Wrappers

**For Primitives float, long , double you could just assign them to an integer value, but if they are Wrappers, then you cant simply assign them a primitive int or short, byte char .**

**float** f = 1;  
*// Float F = 1; !! Compile error , found int* Float F = **new** Float(1.0f);  
 Float F2 = **new** Float(1);  
 Float F1 = **new** Float (**"1F"**);

**But you can do the following**

Float F = 1.0f;

Even though you could do

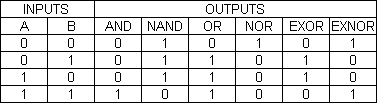
**int** i = **'C'**; with primitives you can’t do the same with wrappers

ex : *// Integer I = 'C'; !! Compile error*

But you can do this Integer I = **new** Integer(**'C'**);

For Character wrapper, you can however do the following

Character c = 1; // This compiles without any issue

* You can use Wrapper class name for primitives and object references (variable names) in Java

Ex . int Integer = 0; , boolean Boolean = false;

Wrappers and Primitives

**Integer**

**Integer(int value) :**Constructs a newly allocated Integer object that represents the specified int value.

**Integer(String s) :**Constructs a newly allocated Integer object that represents the int value indicated by the String parameter.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method | Parameters | Return Type | Description | Examples |
| intValue | None | Int | Returns the value of this Integer as an int |  |
| **Static** parseInt | String | Int | Parses the string argument as a signed decimal integer |  |
| **Static** valueOf | Int | Integer | Returns an Integer instance representing the specified int value. |  |
| **Static** valueOf | String | Integer | Returns an Integer instance representing the specified int value. |  |

**Boolean(boolean value) :** Allocates a Boolean object representing the value argument.

**Boolean(String s):** Allocates a Boolean object representing the value true if the string argument is not null and is equal, ignoring case, to the string "true".

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| booleanValue() | None | boolean | Returns the value of this Boolean object as a boolean primitive. |  |
| **Static** parseBoolean | String | boolean | Parses the string argument as Boolean |  |
| **Static** valueOf() | boolean | Boolean | Returns a Boolean instance representing the specified boolean value. |  |

**Character(char value)**

Constructs a newly allocated Character object that represents the specified char value.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| charValue | None | Char | Returns the value of this Character object. |  |
| **Static** getNumericValue | Char | Int |  |  |
| **Static** valueOf | Char | Character |  |  |

**Double(double value)**

**Double(String str)**

The Double class wraps a value of the primitive type double in an object. An object of type Double contains a single field whose type is double.

In addition, this class provides several methods for converting a double to a String and a String to a double, as well as other constants and methods useful when dealing with a double

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| [**isNaN**](https://docs.oracle.com/javase/8/docs/api/java/lang/Double.html#isNaN--)() | None | Boolean | Returns true if this Double value is a Not-a-Number (NaN), false otherwise. |  |
| **Static** [**isNaN**](https://docs.oracle.com/javase/8/docs/api/java/lang/Double.html#isNaN--)**(double)** | Double | Boolean | Returns true if the specified number is a Not-a-Number (NaN) value, false otherwise. |  |
|  |  |  |  |  |