Java Quick Reference Guide

Arithmetic Operators

+	Addition
-	Subtraction
/	Division (int / floating-point)
	2/3 = 0, 2.0/3.0 =.666667
*	Multiplication
%	Modulus (integer remainder)

Relational/Equality Operators

<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to
!=	Not equal to

Logical Operators

!	NOT
&&	AND
||	OR

Assignment Operators

=	simple assignment
+=	addition/assignment
-=	subtraction/assignment
*=	multiplication/assignment
/=	division/assignment
%=	modulus/assignment

Remember to use the methods **equals( )** or **compareTo( )** when comparing Strings rather than relational comparison operators.

String s1 = "abc", s2 = "def";

String Comparisons:

Compare for equality:

- s1.equals(s2) or
- s1.compareTo(s2) == 0

Remember the compareTo( ) method returns one of 3 values:

- neg number, pos number, 0

Compare for lexical order:

- s1.compareTo(s2) < 0 (s1 before s2)
- s1.compareTo(s2) > 0 (s1 after s2)

Remember to distinguish between integers and real numbers (called floating-point in Java). These are stored differently in memory and have different ranges of values that may be stored.

- integer: 2, 3, -5, 0, 8
- floating-point: 2.0, 0.5, -3., 4.653

Object Creation: (new) new int[10], new GradeBook("CIS 182")

The **new** operator creates an object and returns a reference (address of an object)

Java Types [value/reference]

A value type stores a value of a primitive type **int x = 3;**
A reference type stores the address of an object **Circle c = new Circle(2);**
A reference variable is created using a class name: **GradeBook myGradeBook;**

Primitive Data Types (Java value types) Remember: **String** is a reference type

boolean	flag / logical	true, **false**	[**boolean** literals]
char	character	'A', 'n', '!'	[**char** literals]
byte, short, **int**, long	integral	2, 3, 5000, 0	[**int** literals]
float, **double**	floating-point	123.456, .93	[**double** literals]

Default numeric literal types:

integral:	**int**	int x = 3;	//3 is an int literal
floating-point:	**double**	double y = 2.5;	//2.5 is a double literal

Most commonly used reference type in Java is **String**. String name = "Jack";

Forms of the if Statement

Simple if

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

Example

```
if (x < y)
    x++;
```

if/else

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

Example

```
if (x < y)
    x++;
else
    x--;
```

if/else if (nested if)

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

Example

```
if (x < y)
    x++;
else
    if (x < z)
        x--;
    else
        y++;
```

The **"expression"** in the parentheses for an

if statement

or

loop

is often also referred to as a **"condition"**

To conditionally execute more than one statement, you must create a **compound statement** (block) by enclosing the statements in braces (this is true for loops as well):

Form

```
if (expression)
{
   statement;
   statement;
}
```

Example

```
if (x < y)
{
   x++;
   System.out.println( x );
}
```

Input using Scanner class

Scanner input = new Scanner (System.in); //keyboard input
input methods: next(), nextLine(), nextInt(), nextDouble()

Output methods for System.out or PrintWriter objects

print(), println(), printf() [formatted output]

Conversion from a String to a number using Wrapper Classes

```
double d = Double.parseDouble(dString);
float  f = Float.parseFloat(fString);
int    j = Integer.parseInt(jString);
```

Java Numeric Conversions and Casts:

Widening conversions are done implicitly.

```
double x;    int y = 100;
x = y;       // value of y implicitly converted to a double.
```

Narrowing conversions must be done explicitly using a cast.

```
double x = 100;    int y;
y = (int) x;    // value of x explicitly cast to an int
```

In mixed expressions, numeric conversion happens implicitly. double is the "highest" primitive data type, byte is the "lowest".

Java Quick Reference Guide

The while Loop (pre-test loop)

Form:

```
init;
while (test)
{
   statement;
   update;
}
```

Example:

```
x = 0;
while (x < 10)
{
   sum += x;
   x++;
}
```

The do-while Loop (post-test loop)

Form:

```
init;
do
{
   statement;
   update;
} while (test);
```

Example:

```
x = 0;
do
{
   sum += x;
   x++;
} while (x < 10);
```

The for Loop (pre-test loop)

Form:

```
for (init; test; update)
{
    statement;
}
```

Example:

```
for (int count=1; count<=10; count++)
{
    System.out.println( count );
}
```

Enhanced for loop:

```
for (parameter : collection)
     statement;
```

```
int scores[ ] = {85, 92, 76, 66, 94};   //collection is the array scores
for ( int number : scores )             //parameter is the variable number
   System.out.println(number);
```

Escape Sequences

Special characters in Java

\n	newline character	'\n'
\t	tab character	'\t'
\"	double quote	'\"'
\'	single quote	'\''
\\	backslash	'\\'

Operator Precedence

```
    (  )
  ----------
   *, /, %      [ mathematical ]
  ----------
    +, -
```

Logical operators: `!, &&, ||`

(1) mathematical (2) relational (3) logical

Selection and Loop Structures :

Looping:

- Java **Pre-test** loops
- Test precedes loop body
  - while
  - for

Loop Control:

- 3 types of expressions that are used to control loops:
  - initialization (init)
  - test
  - update
- Counter-controlled loops, aka definite loops, work with a **loop control variable** (lcv)
- Sentinel-controlled loops, aka indefinite loops, work with a **sentinel value**
- Java Loop Early Exit:
  - **break** statement

Note: The **break** statement can be used with a **loop** in Java. **Loops** may also use a **continue** statement.

Java Arrays: Create an array (2 ways)

```
1. <type> <array-name>[ ] = new <type>[size];
2. <type> <array-name>[ ] = { <initializer-list> };
```

```
//create an array of 20 elements.
       int    myArray[ ] = new int[20];

//create an array of 3 elements set to the values in the initializer list.
       int    myArray[ ] = { 1, 2, 3 };
       String stooges[ ] = { "Moe", "Larry", "Curly" };

//assign value of first element in myArray to the integer variable x.
       int x = myArray[0];

//assign value of the last element in myArray to the integer variable y.
       int y = myArray[ myArray.length-1 ];
```

All arrays have a public field named `length` which holds the number of elements in the array.

Given this declaration: `int x[][][];`

`x.length`	is the number of elements in the array in the first dimension.
`x[m].length`	is the number of elements for a specific array in the second dimension.
`x[m][n].length`	is the number of elements for a specific array in the third dimension.

Use the ArrayList class to create a dynamically resizable array.

The Arrays class has static methods that can be used with arrays and ArrayLists to search, sort, copy, compare for equality, etc.

int num[ ]; … <stmts> ….

Create a new initialized array and assign to num.
num = new int[ ]{1,2,3,4,5};

Java Methods: <type> <method-name> ([<type> parameter1, [<type parameter2, …]])

Methods that will not return a value will have the return type **void** in the method header.

```
void printHeadings( )    //no parameters, return type is void
{ <method body> }

void printDetailLine( String name, int number, double gpa )   //3 parameters, return type is void
{ <method body> }

int getCount( )    //no parameters, return type is int
{ <method body> }

double max( double x, double y )   //2 parameters, return type is double
{ <method body> }
```

When a method is called, the data is passed to the parameters (if any) using arguments

//Arguments: "Jack Wilson", 100, 3.50 passed to Parameters: name, number, gpa for Method: printDetailLine (see method header above) : **`printDetailLine( "Jack Wilson", 100, 3.50);`**

A method may be declared with one variable length parameter. It must be the last parameter declared. The syntax of the declaration is <type> **...** <parameter-name>.

Examples: `int... numbers,  double ... values, String ...names` //implicit **array** creation