**Cheat Sheet – Java Programming**

CREDIT: Based on a similar cheat-sheet from the book, “Introduction to Programming in Java, An Interdisciplinary Approach” by Robert Sedgewick, Kevin Wayne. See URL: <http://introcs.cs.princton.edu/java/11cheatsheet/>

***Basic Reference:*** An initial list of useful statements, operators, and misc. info to get you started with Java. You should become familiar with types, statements, operators, and precedence.

***Hello, World*:**

|  |
| --- |
|  |

***Editing, compiling, and executing*:**

|  |
| --- |
|  |

***Declaration and Assignment Statements*:**

|  |
| --- |
|  |

***Conditional Statements*:**

|  |
| --- |
| **// if then else** statement  **if** (test) {  // Code path "A", execute if test == true  }  **else** {  // Code path "B", execute if test == false  } |
| // **Nested** if-then-else statements  **if** (income < 0) rate = .0;  **else if** (income < 47450) rate = .22;  **else if** (income < 114650) rate = .25;  **else if** (income < 174700) rate = .28;  **else if** (income < 311950) rate = .33;  **else** rate = .35; |
| // **Switch** statement  **switch** (day) { // Display Day of Week  **case** 0: System.out.println("Sun"); **break**;  **case** 1: System.out.println("Mon"); **break**;  **case** 2: System.out.println("Tue"); **break**;  **case** 3: System.out.println("Wed"); **break**;  **case** 4: System.out.println("Thu"); **break**;  **case** 5: System.out.println("Fri"); **break**;  **case** 6: System.out.println("Sat"); **break**;  **default**: System.out.println("Error"); **break**;  } |

***Iteration Statements*:**

|  |
| --- |
| // **While** (repeatedly execute “loop body” while test is true) |
| // **For** (execute “loop body” specified number of times) |
| // **do** … **while** loop (execute “loop body” at least once)  **do** { // Generate a random 2D point inside circle  x = 2.0\*Math.random() – 1.0;  y = 2.0\*Math.random() – 1.0;  } **while** (Math.sqrt(x\*x + y\*y) > 1.0); |
| // **Break** statement (exit a loop early)  **for** (int i = 2; i <= N/i; i++) // Test for prime number  **if** (N % i == 0) **break**;  **if** (i > N/i) System.out.printLn(N + " is prime"); |

***Functions*:**

|  |
| --- |
| // Function (method) implementation |
| // Calling a function (method), AKA method invocation  double val = Math.**sqrt**( 25.0 ); |

***Built-in Data Types*:**

|  |  |  |
| --- | --- | --- |
| ***Type*** | ***Values*** | ***Literals*** |
| int  long | integers (32-bit)  (64-bit) | 9, -12,  2^24-1 |
| float  double | floating-point numbers | 3.14, -2.5,  6.022e23 |
| boolean | boolean values | true, false |
| char | individual  characters | 'A', '1',  '%', '\n' |
| string | sequences of characters | "AB","Hello",  "2.5" |

***Scientific Notation*:**

|  |  |  |
| --- | --- | --- |
| ***Full Number*** | ***Sci. Not.*** | ***Java*** |
| 5723000000 | 5.723×109 | 5.723e9 |
| -0.000000689 | -6.89×10-7 | -6.123e-7 |

***Parse strings into numbers*:**

|  |  |
| --- | --- |
| ***Character*** | ***Description*** |
| int Integer.parseInt(String s);  // Converts s into an int value | |
| int Double.parseDouble(String s);  // Converts s into a double value | |
| long Long.parseLong(String s);  // Converts s into a long value | |

***Special Characters*:**

|  |  |
| --- | --- |
| ***Character*** | ***Description*** |
| () | Prioritize operators (precedence)  Pass method arguments  Invoke a method  Cast operator |
| [] | Access array element |
| {} | Specify initializer list |
| % | Modulus operator (remainder) |
| . | Decimal point  Access object member |
| , | Separate method arguments |
| ; | End of statement |
| ' ' | Single Quote, Construct character |
| " " | Double Quote, Construct string |
| //  /\*…\*/ | Single line comment  Multi-line comment |
| \ | Escape character  Separate locations (in File or URL) |

***Basic Operators*:**

|  |  |  |
| --- | --- | --- |
| ***Name*** |  | ***Examples*** |
| ***Unary Operators*** | | |
| Plus | + | +5 |
| Minus | - | –2 |
| Increment (add 1) | ++ | i++ or ++i |
| Decrement (minus 1) | -- | i-- or --i |
| Logical NOT | ! | !done |
| ***Arithmetic Operators (Binary)*** | | |
| Addition | + | 2+5 // =7 |
| Subtraction | - | 9-2 // =7 |
| Multiplication | \* | 5\*8 // = 40 |
| Division | / | 5/4 // = 1.25 |
| Remainder (modulus) | % | 5%3 // = 2 |
| Exponentiation |  | Does not exist in Java.  Use Math.pow() instead. |
| ***Assignment Operator*** | | |
| Assignment | = | a = (2\*b)+c; |

***Advanced Operators*:**

|  |  |  |  |
| --- | --- | --- | --- |
| ***Name*** | |  | ***Examples*** |
| **Conditional Operators** | | | |
| conditional AND | && | | test = (a && b); |
| conditional OR | || | | test = (a || b); |
| if-then-else (ternary) | ?: | | c = test ? a:b; |
| **Comparison Operators (Relational)** | | | |
| Equality | == | | (a == b) |
| Inequality | != | | (a != b) |
| Greater than | > | | (a > b) |
| Greater than (or equal) | >= | | (a >= b) |
| Less than | < | | (a < b) |
| Less than (or equal) | <= | | (a <= b) |
| **Bitwise Operators** | | | |
| Bitwise NOT | ~ | | inverse = ~A; |
| Bitwise AND | & | | C = (A & B); |
| Bitwise OR | | | | C = (A | B); |
| Bitwise exclusive OR | ^ | | C = (A ^ B); |
| Signed Left Shift | << | | C = (A << 2); |
| Signed Right Shift | >> | | C = (A >> 2); |
| Unsigned Right shift | >>> | | C = (A >>> 2); |
| **More Assignment Operators** | | | |
| =, +=, -=, \*=, /=, %=, &=, ^=, |=, >>=, <<=, >>>= | | | |

***Precedence Rules*:**

|  |  |
| --- | --- |
| ***Level*** | ***Operators / Descriptions*** |
| a) **If in doubt use parentheses to make intent clear.**  b) **Associativity:** assume “*Left to Right”* (LtoR)  Unless noted as “*Right to Left*” (RtoL)  c) Precedence is from Highest (1) to Lowest (15) | |
| 1 | [] access array element (LtoR)  . access object member (LtoR)  () invoke a method (LtoR)  ++ (post-increment) (LtoR)  -- (post-decrement) (LtoR) |
| 2 | ++ (pre-increment) (RtoL)  -- (pre-decrement) (RtoL)  + Unary Plus (RtoL)  - (Unary Minus) (RtoL)  ! logical NOT (RtoL)  ~ bitwise NOT (RtoL) |
| 3 | (<type>) cast operator (RtoL)  new object creation (RtoL) |
| 4 | \*, /, % (**Multiplicative**) (LtoR) |
| 5 | +, - (**Additive**) (LtoR)  **+** (string concatenation) (LtoR) |
| 6 | <<, >>, >>> (**Shifts**) (LtoR) |
| 7 | <, <=, >, >= (**Comparisons**) (LtoR) |
| 8 | ==, != (**Equality**) (LtoR) |
| 9 | & Bitwise AND (LtoR) |
| 10 | ^ Bitwise XOR (LtoR) |
| 11 | | Bitwise OR (LtoR) |
| 12 | && Conditional AND (LtoR) |
| 13 | || Conditional OR (LtoR) |
| 14 | ?: (Conditional if-then-else) (LtoR) |
| 15 | **Assignment** (RtoL)  =, +=, -=, \*=, /=, %=, &=, ^=, |=, >>=, <<=, >>>= |

***Math Library*:**

|  |  |
| --- | --- |
| ***Method/Example*** | ***Description*** |
| a.) *Some* methods frompublic class [Math](http://docs.oracle.com/javase/6/docs/api/java/lang/Math.html)  b.) Almost all return types and method arguments are of type double. | |
| c = **abs**(a); | Absolute value ofa |
| c = **max**(a,b); | maximum of a and b |
| c = **min**(a,b); | minimum of a and b |
| c = **sin**(a); | sine function |
| c = **cos**(a); | cosine function |
| c = **tan**(a); | tangent function |
| c = **exp**(a); | Exponential (*ea*) |
| c = **log**(a); | Natural log (log*ea*, or ln *a*) |
| c = **pow**(a,b); | Raise *a* to *b*th power (*ab*) |
| c = **round**(a); | Round to the nearest integer  **Note:** c is of type long. |
| c = **random**(); | random number in [0, 1) |
| c = **sqrt**(a); | Square root of *a*. |
| double E; | value of *e* (constant) |
| double PI; | value of π (constant) |

***Type Conversion*:**

|  |  |  |
| --- | --- | --- |
| ***Expression*** | ***Type*** | ***Value*** |
| *Upcasting*: Converts a smaller type to a larger type, which can safely hold all values. Java will implicitly perform “upcasts” on your behalf. Also known as (AKA) “wide casting”.  *Downcasting*: Converts a larger type to a smaller type, which means information may be lost. The programmer must explicitly perform “downcasts”. AKA “narrow casting”. | | |
| "1234" + 99 | String | "123499" |
| Integer.parseInt("123") | int | 123 |
| (int)2.71828 | int | 2 |
| Math.round(2.71828) | long | 3 |
| (int)Math.round(2.71828) | int | 3 |
| (int)Math.round(3.14159) | int | 3 |
| 11 \* 0.3 | double | 3.3 |
| (int)11\*0.3 | double | 3.3 |
| 11\*(int)0.3 | int | 0 |
| (int)(11\*0.3) | int | 3 |

**Simple Exception Handling:**

|  |
| --- |
| FileInputStream in = null;  FileOutputStream out = null;  int c;  **try** {  // Allocate resources  in = new FileInputStream("in.txt");  out = new FileOutputStream("out.txt");  // Code to error protect  **while** (c = in.read() != -1) {  out.write(c);  }  } **catch** (Exception e) {  // Handle specific exception  } **catch** (…) {  // Handle all exceptions  } **finally** {  // Cleanup resources  if (in != null) in.close();  if (out != null) out.close();  } |

**Simple I/O:**

|  |  |
| --- | --- |
| ***Class or Method*** | ***Desc./Notes*** |
| System.in | Standard input |
| System.out | Standard output |
| System.err | Standard error |
| Console c = new Console(); | JVM Console |
| in.read() | Reads one byte |
| in.read(byte[] b) | Reads byte array |
| out.format(…)  out.printf(…); | Writes formatted string |
| out.print(val) | Writes value of specified type |
| out.println(val); | Writes value then terminates line |