

Programming with Java - An Algorithmic Introduction

Appendix 1 - Recommended further reading

There are now hundreds of text books that describe Java, provide insight into advanced features of Java, or claim to be suitable for learning to program from scratch in Java. Of the latter there are several that teach "objects first" and others (like the present book) that teach "objects later". Among all these, the following may be worth drawing to your attention. Several of these have been through several editions, trying to track changes in Java.

David Flanagan: *Java in a Nutshell* (O'Reilly, 2005) (5th Edition)

An excellent reference, highly recommended, but not a beginner's book.

Bruce Eckel: *Thinking in Java* (Prentice Hall, 2006) (4th Edition)

Another really excellent book with a mass of insight, but it is not a beginner's book. You have to know Java to be able to benefit.

Kathy Sierra and Bert Bates: *Head First Java* (O'Reilly, 2005) (2nd Edition)

A whimsical and amusing book, full of insight, but it is not really a first time text. Once you have learned a quite a bit about Java you can learn a lot more from this one.

Kim King: *Java Programming from the beginning* (Norton, 2000)

Very clearly written and a joy to read. Unfortunately it does not describe the latest features of Java at all, but it is an excellent text for the rest!

Cay Horstmann: *Java Concepts* (Wiley, 2005) (4th Edition)

Much better than average book, with some very clear explanations, and quite accessible to beginners.

Samuel Kamin, Dennis Mickunas, Edward Reingold: *An Introduction to Computer Science using Java* (McGraw-Hill, 2002) (2nd Edition)

A fairly well structured text, although objects are introduced quite early on before many fundamental algorithmic concepts.

Barry Cornelius: *Understanding Java* (Addison Wesley, 2001)

An interesting and sometimes controversial book from someone with a perceptive outlook on programming language design

John Lewis and William Loftus: *Java Software Solutions: Foundations of Program Design* (Pearson, 2005) (4th Edition)

This was formerly used as a text at Rhodes. The many editions have chopped and changed, and it is now quite heavy reading.

Tony Gaddis: *Starting Out with Java: Control Structures through Objects* (Addison Wesley, 2008) (3rd Edition)

Formerly used as a text at Rhodes. Gaddis has a whole series of books on Java from various perspectives. They give the impression of being badly edited - concepts are covered in a confusing order.

Ralph Bravaco and Shai Simonson: *Java Programming: From The Ground Up* (McGraw Hill, 2010)

A recent "fundamentals first" book which I found refreshing, even though, as with many others, there is a huge amount to wade through!

Appendix 2 - Java reserved words

Java key words cannot be used as identifiers. Note that `const` and `goto` are reserved for possible future use, even though they are not currently used. `true`, `false`, and `null` are actually reserved literals but cannot be used as identifiers either.

<code>abstract</code>	<code>assert</code>	<code>boolean</code>	<code>break</code>	<code>byte</code>	<code>case</code>	<code>catch</code>
<code>char</code>	<code>class</code>	<code>(const)</code>	<code>continue</code>	<code>default</code>	<code>do</code>	<code>double</code>
<code>else</code>	<code>enum</code>	<code>extends</code>	<code>(false)</code>	<code>final</code>	<code>finally</code>	<code>float</code>
<code>for</code>	<code>(goto)</code>	<code>if</code>	<code>implements</code>	<code>import</code>	<code>instanceof</code>	<code>int</code>
<code>interface</code>	<code>long</code>	<code>native</code>	<code>new</code>	<code>(null)</code>	<code>package</code>	<code>private</code>
<code>protected</code>	<code>public</code>	<code>return</code>	<code>short</code>	<code>static</code>	<code>strictfp</code>	<code>super</code>
<code>switch</code>	<code>synchronized</code>	<code>this</code>	<code>throw</code>	<code>throws</code>	<code>transient</code>	<code>(true)</code>
<code>try</code>	<code>void</code>	<code>volatile</code>	<code>while</code>			

Appendix 3 - The ASCII character set

Java uses the Unicode character set, but the first 256 characters of this are the same as the very widely used ASCII set which includes the alphabet, digit, and familiar punctuation characters.

00	0	<NUL>	10	16	<DLE>	20	32	SP	30	48	0	40	64	@	50	80	P	60	96	`	70	112	p
01	1	<SOH>	11	17	<DC1>	21	33	!	31	49	1	41	65	A	51	81	Q	61	97	a	71	113	q
02	2	<STX>	12	18	<DC2>	22	34	"	32	50	2	42	66	B	52	82	R	62	98	b	72	114	r
03	3	<ETX>	13	19	<DC3>	23	35	#	33	51	3	43	67	C	53	83	S	63	99	c	73	115	s
04	4	<EOT>	14	20	<DC4>	24	36	\$	34	52	4	44	68	D	54	84	T	64	100	d	74	116	t
05	5	<ENQ>	15	21	<NAK>	25	37	%	35	53	5	45	69	E	55	85	U	65	101	e	75	117	u
06	6	<ACK>	16	22	<SYN>	26	38	&	36	54	6	46	70	F	56	86	V	66	102	f	76	118	v
07	7	<BEL>	17	23	<ETB>	27	39	'	37	55	7	47	71	G	57	87	W	67	103	g	77	119	w
08	8	<BS>	18	24	<CAN>	28	40	(38	56	8	48	72	H	58	88	X	68	104	h	78	120	x
09	9	<HT>	19	25		29	41)	39	57	9	49	73	I	59	89	Y	69	105	i	79	121	y
0A	10	<LF>	1A	26	<SUB>	2A	42	*	3A	58	:	4A	74	J	5A	90	Z	6A	106	j	7A	122	z
0B	11	<VT>	1B	27	<ESC>	2B	43	+	3B	59	:	4B	75	K	5B	91	[6B	107	k	7B	123	{
0C	12	<FF>	1C	28	<FS>	2C	44	,	3C	60	<	4C	76	L	5C	92	\	6C	108	l	7C	124	
0D	13	<CR>	1D	29	<GS>	2D	45	-	3D	61	=	4D	77	M	5D	93]	6D	109	m	7D	125	}
0E	14	<SO>	1E	30	<RS>	2E	46	.	3E	62	>	4E	78	N	5E	94	^	6E	110	n	7E	126	~
0F	15	<SI>	1F	31	<US>	2F	47	/	3F	63	?	4F	79	O	5F	95	_	6F	111	o	7F	127	

Appendix 4 - Escape sequences for character specification

Character literals are written between single quotes, as exemplified by `'a'`, `'x'`, `'9'`.

Not all character can be represented by themselves in that way. Java allows awkward characters to be represented by so-called escape sequences.

Unicode characters may be specified by `'\uNNNN'` where `NNNN` is the hexadecimal representation of the position in the Unicode character set. For example `'\u004D'` is equivalent to `'M'`, while `'\u001B'` represents `<ESC>`, the character corresponding to pressing the ESC key on a keyboard

Unsigned short hexadecimal values may be represented by `'\x00NN'` where `NNNN` is the hexadecimal representation of the short value. Values of the `short` and `char` types are both represented by 16 bit numbers.

There are some special values that are represented by sequences `'\X'` where `X` is itself a simple character. These include

<code>'\a'</code>	Alert	<code><BEL></code>	<code>0x0007</code>	7
<code>'\b'</code>	Backspace	<code><BS></code>	<code>0x0008</code>	8
<code>'\t'</code>	Horizontal tab	<code><HT></code>	<code>0x0009</code>	9
<code>'\n'</code>	Newline	<code><LF></code>	<code>0x000A</code>	10
<code>'\v'</code>	Vertical tab	<code><VT></code>	<code>0x000B</code>	11
<code>'\f'</code>	Formfeed (page)	<code><FF></code>	<code>0x000C</code>	12
<code>'\r'</code>	Carriage return	<code><CR></code>	<code>0x000D</code>	13
<code>'\"'</code>	Double quote	<code>"</code>	<code>0x0022</code>	34
<code>'\''</code>	Single quote	<code>'</code>	<code>0x0027</code>	39
<code>'\\'</code>	Backslash	<code>\</code>	<code>0x005C</code>	92

Other sequences of the form `'\X'` simply represent `'X'`

Appendix 5 - A screen handling library

Input and output in Java programs is usually achieved in one of two ways

- Simple console based I/O using facilities like those provided in the IO library that has been used extensively in this book
- GUI based IO, using multiple windows and event driven software

While the first of these is adequate for introductory programs, the GUI based approach requires the programmer to have a fairly deep understanding of multiple advanced concepts.

The library described here has been developed from sources available on the Internet. It provides the ability to regard a console single window as an (x,y) addressable area, and has facilities for writing text into this window at specified points, and for reading values in response to prompts placed at appropriate points. Typically a window can have at least 80 columns and 25 rows, but the exact size will be determined by the size of the console window in which the application is executed.

This library affords the opportunity of writing fairly sophisticated applications, like those set as exercises from chapter 15 onwards (or even before!)

```
import jcurses.system.*;

public class Screen {
    // Simple screen addressed I/O like that in TurboPascal
    // Built on the jcurses library found at
    // http://sourceforge.net/projects/javacurses/
    // (Alexei Chmelev)
    // The screen is addressed as (0,0) in the top left, with x increasing to the
    // right and y increasing downwards. The limits are imposed by the size of
    // the DOS window in which the client program is run.
    // P.D. Terry, Rhodes University, 2009

    public static void init()
    public static void init(boolean pauseOnExit) {
        // Initialize internal variables and hidden handlers
        // if pauseOnExit is true or omitted a user must type ESC to release the
        // screen when the client program terminates.

        // Definitions of colours - magic numbers

        public static final short BLACK = CharColor.BLACK; // 0
        public static final short RED = CharColor.RED; // 1
        public static final short GREEN = CharColor.GREEN; // 2
        public static final short YELLOW = CharColor.YELLOW; // 3
        public static final short BLUE = CharColor.BLUE; // 4
        public static final short MAGENTA = CharColor.MAGENTA; // 5
        public static final short CYAN = CharColor.CYAN; // 6
        public static final short WHITE = CharColor.WHITE; // 7

        // Getting and Setting colors and background
        public static void setColors(short backgroundColor, short foregroundColor)
        public static void invertColors()
        public static void setBackgroundColor(short backgroundColor)
        public static void setForegroundColor(short foregroundColor)
        public static short getBackgroundColor()
        public static short getForegroundColor()

        // Get details of screen
        public static int getScreenHeight()
        public static int getScreenWidth()

        // Screen and Line clearing routines
        public static void clearScreen()
        public static void clearScreen(short backgroundColor)
        public static void clearEol(int x, int y)

        // Single key press routines - no need for ENTER to terminate

        public static int getKey()

        // Waits for keypress and returns key code. Most are ASCII but there are some
        // function key mappings as shown here - more magic numbers
    }
}
```

```

public static final int KEY_DOWN      = 258;
public static final int KEY_UP        = 259;
public static final int KEY_LEFT      = 260;
public static final int KEY_RIGHT     = 261;
public static final int KEY_HOME      = 262;
public static final int KEY_BACKSPACE = 263;
public static final int KEY_F1        = 265;
public static final int KEY_F2        = 266;
public static final int KEY_F3        = 267;
public static final int KEY_F4        = 268;
public static final int KEY_F5        = 269;
public static final int KEY_F6        = 270;
public static final int KEY_F7        = 271;
public static final int KEY_F8        = 272;
public static final int KEY_F9        = 273;
public static final int KEY_F10       = 274;
public static final int KEY_F11       = 275;
public static final int KEY_F12       = 276;
public static final int KEY_DELETE    = 330;
public static final int KEY_INSERT    = 331;
public static final int KEY_PAGEDOWN  = 338;
public static final int KEY_PAGEUP    = 339;
public static final int KEY_PRINT     = 346;
public static final int KEY_END       = 360;

// Input routines - display prompt at (x, y) and obtain token

public static String  readLine  (int x, int y, String prompt)
public static byte    readByte  (int x, int y, String prompt)
public static short   readShort (int x, int y, String prompt)
public static int     readInt   (int x, int y, String prompt)
public static long    readLong  (int x, int y, String prompt)
public static boolean readBoolean(int x, int y, String prompt)
public static char    readChar  (int x, int y, String prompt)
public static int     readKey   (int x, int y, String prompt)
public static float   readFloat (int x, int y, String prompt)
public static double  readDouble(int x, int y, String prompt)

// Input errors can be trapped and then abort the program or
// they may be ignored when numerical values are returned as 0

public static void ignoreErrors()
public static void trapErrors()

// Output routines for almost any value at (x, y)

public static void write(int x, int y, String s)
public static void write(int x, int y, Object o)
public static void write(int x, int y, byte o)
public static void write(int x, int y, short o)
public static void write(int x, int y, int o)
public static void write(int x, int y, long o)
public static void write(int x, int y, boolean o)
public static void write(int x, int y, char o)
public static void write(int x, int y, char[] o)
public static void write(int x, int y, float o)
public static void write(int x, int y, double o)

// Various utility routines
public static void drawRectangle(int x, int y, int width, int height)
public static void drawHorizontalLine(int x, int y, int x2)
public static void drawVerticalLine(int x, int y, int y2)
} // Screen

```

To make use of this package you must ensure that you have copied the files `jcurses.jar` and `libjcurses.dll` into a directory that is listed in the `CLASSPATH` for your system (see section 10.7).

A very simple demonstration program showing how input may be requested from various points on the screen, and output written elsewhere, is as follows:

```
import library.Screen;

public class ScreenDemo0 { // ScreenDemo0.java
// Very basic demonstration of use of Screen library
// P.D. Terry, Rhodes University, 2009

    public static void main(String[] args) {
        Screen.init(); // always start with this
        Screen.clearScreen();
        String name = Screen.readLine(0, 2, "Name please");
        String prompt = name + ", please supply a value for ";
        int x = Screen.readInt(0, 4, prompt + "x");
        int y = Screen.readInt(0, 5, prompt + "y");
        Screen.write(x, y, "Hello " + name);
    } // main
} // ScreenDemo0
```

A second demonstration program, showing how one might navigate around an initially blank screen and substitute the characters typed at various positions, as the same time updating a matching rectangular matrix, as follows:

```
import library.Screen;

public class ScreenDemo1 { // ScreenDemo1.java
// Demonstration of use of Screen library
// Tracks cursor and initializes an array of characters
// P.D. Terry, Rhodes University, 2009

    public static void main(String[] args) {
        final int size = 20;
        Screen.init(false); // no delay on exit
        Screen.clearScreen();
        char[][] screen = new char[size][size];
        for (int x = 0; x < size; x++)
            for (int y = 0; y < size; y++)
                screen[x][y] = (char) ' ';
        int key;
        int x = 0;
        int y = 0;
        do {
            Screen.write(0, size + 5, x + "," + y
                + " Use the arrow keys to navigate, ESC to quit");
            Screen.invertColors();
            Screen.write(x, y, screen[x][y]); // display original value, highlighted
            key = Screen.getKey();
            Screen.invertColors();
            Screen.write(x, y, screen[x][y]); // restore original value, normal
            switch (key) {
                case Screen.KEY_UP : y = (y - 1 + size) % size; break;
                case Screen.KEY_DOWN : y = (y + 1) % size; break;
                case Screen.KEY_LEFT : x = (x - 1 + size) % size; break;
                case Screen.KEY_RIGHT : x = (x + 1) % size; break;
                default: if (key >= ' ' && key <= 255) { // ignore other control characters
                    screen[x][y] = (char) key; Screen.write(x, y, screen[x][y]);
                } break;
            } // switch
        } while (key != Screen.KEY_ESCAPE);

    } // main
} // ScreenDemo1
```

Appendix 6 - A rudimentary GUI for simple IO

The class below will allow I/O to be performed in very simple pop-up windows. Essentially all this buys you is the ability to avoid using the console I/O of the IO library.

```
public class GIO {
// Provide simple facilities for text I/O
// (standard input and output streams in dialogue boxes)

// Input routines

// Unlabelled

public static String  readString()
public static char    readChar()
public static boolean readBoolean()
public static int     readInt()
public static short   readShort()
public static long    readLong()
public static float   readFloat()
public static double  readDouble()

// Labelled

public static String  readString(String prompt)
public static char    readChar(String prompt)
public static boolean readBoolean(String prompt)
public static int     readInt(String prompt)
public static short   readShort(String prompt)
public static long    readLong(String prompt)
public static float   readFloat(String prompt)
public static double  readDouble(String prompt)

// Formatters

public static String fixedRep(double d, int fractionDigits)
public static String fixedRep(float f, int fractionDigits)
public static String floatingRep(double d, int fractionDigits)
public static String floatingRep(float f, int fractionDigits)

// Output routines

// Unlabelled

public static void write(Object o)
public static void write(double d)
public static void write(int i)
public static void write(char c)
public static void write(long i)
public static void write(boolean b)
public static void writeFixed(double d, int fractionDigits)
public static void writeFloating(double d, int fractionDigits)
public static void write(float f)
public static void writeFixed(float d, int fractionDigits)
public static void writeFloating(float d, int fractionDigits)

// Labelled

public static void write(String label, Object o)
public static void write(String label, double d)
public static void write(String label, char c)
public static void write(String label, int i)
public static void write(String label, long i)
public static void write(String label, boolean b)
public static void writeFixed(String label, double d, int fractionDigits)
public static void writeFloating(String label, double d, int fractionDigits)
public static void write(String label, float f)
public static void writeFixed(String label, float d, int fractionDigits)
public static void writeFloating(String label, float d, int fractionDigits)

} // GIO
```