CS3231
Object Oriented Programming I

Name: _____ ( ) Date: _____

Appendix: Java Keywords, Operator Precedence Chart and ASCII Chart

1 Java Keywords

The following fifty keywords are reserved for use by the Java language:

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

2 Operator Precedence Chart

The operators are shown in decreasing order of precedence from top to bottom. Operators in the same group have the same precedence, and their associativity is shown in the table.

Operator	*Name*	*Associativity*
`()`	Parentheses	Left to right
`()`	Function call	Left to right
`[]`	Array subscript	Left to right
`.`	Object member access	Left to right
`++`	Postincrement	Left to right
`--`	Postdecrement	Left to right
`++`	Preincrement	Right to left
`--`	Predecrement	Right to left
`+`	Unary plus	Right to left
`-`	Unary minus	Right to left
`!`	Unary logical negation	Right to left
`(type)`	Unary casting	Right to left
`new`	Creating object	Right to left
`*`	Multiplication	Left to right
`/`	Division	Left to right
`%`	Remainder	Left to right
`+`	Addition	Left to right
`-`	Subtraction	Left to right
`<<`	Left shift	Left to right
`>>`	Right shift with sign extension	Left to right
`>>>`	Right shift with zero extension	Left to right
`<`	Less than	Left to right
`<=`	Less than or equal to	Left to right
`>`	Greater than	Left to right
`>=`	Greater than or equal to	Left to right
`instanceof`	Checking object type	Left to right

Operator	*Name*	*Associativity*
==	Equal comparison	Left to right
!=	Not equal	Left to right
&	(Unconditional AND)	Left to right
^	(Exclusive OR)	Left to right
|	(Unconditional OR)	Left to right
&&	Conditional AND	Left to right
||	Conditional OR	Left to right
?:	Ternary condition	Right to left
=	Assignment	Right to left
+=	Addition assignment	Right to left
-=	Subtraction assignment	Right to left
*=	Multiplication assignment	Right to left
/=	Division assignment	Right to left
%=	Remainder assignment	Right to left

3 ASCII Table - Standard and Extended ASCII Table

The ASCII character set is a subset of the Unicode character set used by Java to represent characters from most of the world's languages.

Regular ASCII Chart (character codes 0 - 127)

000	(nul)	016 ►	(dle)	032	sp	048	0	064	@	080	P	096	`	112	p
001 ☺	(soh)	017 ◄	(dc1)	033	!	049	1	065	A	081	Q	097	a	113	q
002 ☻	(stx)	018 ↕	(dc2)	034	"	050	2	066	B	082	R	098	b	114	r
003 ♥	(etx)	019 ‼	(dc3)	035	#	051	3	067	C	083	S	099	c	115	s
004 ♦	(eot)	020 ¶	(dc4)	036	$	052	4	068	D	084	T	100	d	116	t
005 ♣	(enq)	021 §	(nak)	037	%	053	5	069	E	085	U	101	e	117	u
006 ♠	(ack)	022 ▬	(syn)	038	&	054	6	070	F	086	V	102	f	118	v
007 •	(bel)	023 ↨	(etb)	039	'	055	7	071	G	087	W	103	g	119	w
008 ◘	(bs)	024 ↑	(can)	040	(	056	8	072	H	088	X	104	h	120	x
009	(tab)	025 ↓	(em)	041	)	057	9	073	I	089	Y	105	i	121	y
010	(lf)	026	(eof)	042	*	058	:	074	J	090	Z	106	j	122	z
011 ♂	(vt)	027 ←	(esc)	043	+	059	;	075	K	091	[	107	k	123	{
012 ♀	(np)	028 ∟	(fs)	044	,	060	<	076	L	092	\	108	l	124	|
013	(cr)	029 ↔	(gs)	045	-	061	=	077	M	093	]	109	m	125	}
014 ♫	(so)	030 ▲	(rs)	046	.	062	>	078	N	094	^	110	n	126	~
015 ☼	(si)	031 ▼	(us)	047	/	063	?	079	O	095	_	111	o	127	⌂

Extended ASCII Chart (character codes 128 - 255)

128	Ç	143	Å	158	₧	172	¼	186	‖	200	╚	214	╓	228	Σ	242	≥
129	ü	144	É	159	ƒ	173	¡	187	╗	201	╔	215	╫	229	σ	243	≤
130	é	145	æ	160	á	174	«	188	╝	202	╩	216	╪	230	µ	244	⌠
131	â	146	Æ	161	í	175	»	189	╜	203	╦	217	┘	231	τ	245	⌡
132	ä	147	ô	162	ó	176	░	190	╛	204	╠	218	┌	232	Φ	246	÷
133	à	148	ö	163	ú	177	▒	191	┐	205	═	219	█	233	Θ	247	≈
134	å	149	ò	164	ñ	178	▓	192	└	206	╬	220	▄	234	Ω	248	°
135	ç	150	û	165	Ñ	179	│	193	┴	207	╧	221	▌	235	δ	249	∙
136	ê	151	ù	166	ª	180	┤	194	┬	208	╨	222	▐	236	∞	250	·
137	ë	152	ÿ	167	º	181	╡	195	├	209	╤	223	▀	237	φ	251	√
138	è	153	Ö	168	¿	182	╢	196	─	210	╥	224	α	238	ε	252	ⁿ
139	ï	154	Ü	169	⌐	183	╖	197	┼	211	╙	225	ß	239	∩	253	²
140	î	155	¢	170	¬	184	╕	198	╞	212	╘	226	Γ	240	≡	254	■
141	ì	156	£	171	½	185	╣	199	╟	213	╒	227	π	241	±	255	
142	Ä	157	¥														

4 Java Quick Reference Guide

Class Declaration

```
public class CashRegister
{
   private int itemCount;
   private double totalPrice;

   public void addItem(double price)
   {
      itemCount++;
      totalPrice = totalPrice + price;
   }
   . . .
}
```

Instance variables

Method

Selected Operators and Their Precedence

(See Appendix B for the complete list.)

`[]`	Array element access
`++ -- !`	Increment, decrement, Boolean *not*
`* / %`	Multiplication, division, remainder
`+ -`	Addition, subtraction
`< <= > >=`	Comparisons
`== !=`	Equal, not equal
`&&`	Boolean *and*
`||`	Boolean *or*
`=`	Assignment

Conditional Statement

```
if (floor >= 13)
{
   actualFloor = floor - 1;
}
else if (floor >= 0)
{
   actualFloor = floor;
}
else
{
   System.out.println("Floor negative");
}
```

Condition

Executed when condition is true

Second condition (optional)

Executed when all conditions are false (optional)

Variable and Constant Declarations

```
int cansPerPack = 6;

final double CAN_VOLUME = 0.335;
```

Type, Name, Initial value

Method Declaration

```
public static double cubeVolume(double sideLength)
{
   double volume = sideLength * sideLength * sideLength;
   return volume;
}
```

Modifiers, Return type, Parameter type and name

Exits method and returns result.

Mathematical Operations

`Math.pow(x, y)`	Raising to a power x^y
`Math.sqrt(x)`	Square root $\sqrt{x}$
`Math.log10(x)`	Decimal log $\log_{10}(x)$
`Math.abs(x)`	Absolute value $\|x\|$

`Math.sin(x)`
`Math.cos(x)`
`Math.tan(x)`
	Sine, cosine, tangent of x (x in radians)

String Operations

```
String s = "Hello";
int n = s.length(); // 5
char ch = s.charAt(1); // 'e'
String t = s.substring(1, 4); // "ell"
String u = s.toUpperCase(); // "HELLO"
if (u.equals("HELLO")) ... // Use equals, not ==
for (int i = 0; i < s.length(); i++)
{
   char ch = s.charAt(i);
   Process ch
}
```

Loop Statements

```
while (balance < TARGET)
{
   year++;
   balance = balance * (1 + rate / 100);
}
```

Condition

Executed while condition is true

```
do
{
   System.out.print("Enter a positive integer: ");
   input = in.nextInt();
}
while (input <= 0);
```

Loop body executed at least once

```
for (int i = 0; i < 10; i++)
{
   System.out.println(i);
}
```

Initialization, Condition, Update

```
for (double value : values)
{
   sum = sum + value;
}
```

Set to a new element in each iteration

An array or collection

Executed for each element

Input

```
Scanner in = new Scanner(System.in);
  // Can also use new Scanner(new File("input.txt"));

int n = in.nextInt();
double x = in.nextDouble();
String word = in.next();
String line = in.nextLine();

while (in.hasNextDouble())
{
   double x = in.nextDouble();
   Process x
}
```

Output

Does not advance to new line.

```
System.out.print("Enter a value: ");
```

Use + to concatenate values.

```
System.out.println("Volume: " + volume);
```

Field width, Precision

```
System.out.printf("%-10s %10d %10.2f", name, qty, price);
```

Left-justified string, Integer, Floating-point number

```
try (PrintWriter out = new PrintWriter("output.txt"))
{
   Write to out
}
```

Use the print/println/printf methods.

The output is closed at the end of the try-with-resources statement.

Arrays

Element type, Element type, Length

All elements are zero.

```
int[] numbers = new int[5];
int[] squares = { 0, 1, 4, 9, 16 };
int[][] magicSquare =
   {
      { 16,  3,  2, 13},
      {  5, 10, 11,  8},
      {  9,  6,  7, 12},
      {  4, 15, 14,  1}
   };

for (int i = 0; i < numbers.length; i++)
{
   numbers[i] = i * i;
}

for (int element : numbers)
{
   Process element
}

System.out.println(Arrays.toString(numbers));
   // Prints [0, 1, 4, 9, 16]
```

Array Lists

Use wrapper type, Integer, Double, etc., for primitive types.

Initially empty

Element type (optional)

```
ArrayList<String> names = new ArrayList<String>();
```

Add elements to the end

```
names.add("Ann");
names.add("Cindy"); // [Ann, Cindy], names.size() is now 2

names.add(1, "Bob"); // [Ann, Bob, Cindy]
names.remove(2); // [Ann, Bob]
names.set(1, "Bill"); // [Ann, Bill]

String name = names.get(0); // Gets "Ann"
System.out.println(names); // Prints [Ann, Bill]
```

Linked Lists, Sets, and Iterators

```
LinkedList<String> names = new LinkedList<>();
names.add("Bob"); // Adds at end

ListIterator<String> iter = names.listIterator();
iter.add("Ann"); // Adds before current position

String name = iter.next(); // Returns "Ann"
iter.remove(); // Removes "Ann"

Set<String> names = new HashSet<>();
names.add("Ann"); // Adds to set if not present
names.remove("Bob"); // Removes if present

Iterator<String> iter = names.iterator();
while (iter.hasNext())
{
   Process iter.next()
}
```

Maps

Key type, Value type

```
Map<String, Integer> scores = new HashMap<>();

scores.put("Bob", 10);
Integer score = scores.get("Bob");
```

Returns null if key not present

```
for (String key : scores.keySet())
{
   Process key and scores.get(key)
}
```