CS-1150

Exam #1 Reference Material

NAME _____

USER ID _____

TURN IN THIS REFERENCE PACK WITH YOUR EXAM (It will be returned).

CONTENTS

CS-1150

Exam #1 Reference Material

Java Quick Reference

Console Input

```
Scanner input = new Scanner(System.in);
int intValue = input.nextInt();
long longValue = input.nextLong();
double doubleValue = input.nextDouble();
float floatValue = input.nextFloat();
String string = input.next();
String line = input.nextLine();
```

Console Output

```
System.out.println(anyValue);
```

Conditional Expression

```
boolean-expression ? expression1 :
  expression2

y = (x > 0) ? 1 : -1

System.out.println(number % 2 == 0 ?
  "number is even" : "number is odd");
```

Primitive Data Types

byte	8 bits
short	16 bits
int	32 bits
long	64 bits
float	32 bits
double	64 bits
char	16 bits
boolean	true/false

Arithmetic Operators

+	addition
-	subtraction
*	multiplication
/	division
%	remainder
++var	preincrement
--var	predecrement
var++	postincrement
var--	postdecrement

Assignment Operators

=	assignment
+=	addition assignment
-=	subtraction assignment
*=	multiplication assignment
/=	division assignment
%=	remainder assignment

Relational Operators

<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	equal to
!=	not equal

Logical Operators

&&	short circuit AND
||	short circuit OR
!	NOT
^	exclusive OR

if Statements

```
if (condition) {
  statements;
}

if (condition) {
  statements;
}
else {
  statements;
}

if (condition1) {
  statements;
}
else if (condition2) {
  statements;
}
else {
  statements;
}
```

switch Statements

```
switch (intExpression) {
  case value1:
    statements;
    break;
  ...
  case valuen:
    statements;
    break;
  default:
    statements;
}
```

loop Statements

```
while (condition) {
  statements;
}

do {
  statements;
} while (condition);

for (init; condition;
  adjustment) {
  statements;
}
```

Companion Web site: www.pearsonhighered.com/liang

CS-1150

Exam #1 Reference Material

Java Quick Reference

Frequently Used Static Constants/Methods

```
Math.PI
Math.random()
Math.pow(a, b)
Math.abs(a)
Math.max(a, b)
Math.min(a, b)
Math.sqrt(a)
Math.sin(radians)
Math.asin(a)
Math.toRadians(degrees)
Math.toDegress(radians)
System.currentTimeMillis()
Integer.parseInt(string)
Integer.parseInt(string, radix)
Double.parseDouble(string)
Arrays.sort(type[] list)
Arrays.binarySearch(type[] list, type key)
```

Array/Length/Initializer

```
int[] list = new int[10];
list.length;
int[] list = {1, 2, 3, 4};
```

Multidimensional Array/Length/Initializer

```
int[][] list = new int[10][10];
list.length;
list[0].length;
int[][] list = {{1, 2}, {3, 4}};
```

Ragged Array

```
int[][] m = {{1, 2, 3, 4},
             {1, 2, 3},
             {1, 2},
             {1}};
```

Text File Output

```
PrintWriter output =
  new PrintWriter(filename);
output.print(...);
output.println(...);
output.printf(...);
```

Text File Input

```
Scanner input = new Scanner(
  new File(filename));
```

File Class

```
File file =
  new File(filename);
file.exists()
file.renameTo(File)
file.delete()
```

Object Class

```
Object o = new Object();
o.toString();
o.equals(o1);
```

Comparable Interface

```
c.compareTo(Comparable)
c is a Comparable object
```

String Class

```
String s = "Welcome";
String s = new String(char[]);
int length = s.length();
char ch = s.charAt(index);
int d = s.compareTo(s1);
boolean b = s.equals(s1);
boolean b = s.startsWith(s1);
boolean b = s.endsWith(s1);
boolean b = s.contains(s1);
String s1 = s.trim();
String s1 = s.toUpperCase();
String s1 = s.toLowerCase();
int index = s.indexOf(ch);
int index = s.lastIndexOf(ch);
String s1 = s.substring(ch);
String s1 = s.substring(i,j);
char[] chs = s.toCharArray();
boolean b = s.matches(regex);
String s1 = s.replaceAll(regex,repl);
String[] tokens = s.split(regex);
```

ArrayList Class

```
ArrayList<E> list = new ArrayList<>();
list.add(object);
list.add(index, object);
list.clear();
Object o = list.get(index);
boolean b = list.isEmpty();
boolean b = list.contains(object);
int i = list.size();
list.remove(index);
list.set(index, object);
int i = list.indexOf(object);
int i = list.lastIndexOf(object);
```

printf Method

```
System.out.printf("%b %c %d %f %e %s",
  true, 'A', 45, 45.5, 45.5, "Welcome");
System.out.printf("%-5d %10.2f %10.2e %8s",
  45, 45.5, 45.5, "Welcome");
```

Companion Web site: www.pearsonhighered.com/liang

CS-1150

Exam #1 Reference Material

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

CS-1150

Exam #1 Reference Material

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

TABLE 2.1 Numeric Data Types

Name	*Range*	*Storage Size*
`byte`	-2^7 to $2^7 - 1$ (-128 to 127)	8-bit signed
`short`	-2^{15} to $2^{15} - 1$ (-32768 to 32767)	16-bit signed
`int`	-2^{31} to $2^{31} - 1$ (-2147483648 to 2147483647)	32-bit signed
`long`	-2^{63} to $2^{63} - 1$	64-bit signed
	(i.e., -9223372036854775808 to 9223372036854775807)	
`float`	Negative range: $-3.4028235E + 38$ to $-1.4E - 45$	32-bit IEEE 754
	Positive range: $1.4E - 45$ to $3.4028235E + 38$	
`double`	Negative range: $-1.7976931348623157E + 308$ to $-4.9E - 324$	64-bit IEEE 754
	Positive range: $4.9E - 324$ to $1.7976931348623157E + 308$	

CS-1150

Exam #1 Reference Material

TABLE 2.2 Methods for `Scanner` Objects

Method	*Description*
`nextByte()`	reads an integer of the `byte` type.
`nextShort()`	reads an integer of the `short` type.
`nextInt()`	reads an integer of the `int` type.
`nextLong()`	reads an integer of the `long` type.
`nextFloat()`	reads a number of the `float` type.
`nextDouble()`	reads a number of the `double` type.

TABLE 4.1 Trigonometric Methods in the Math Class

Method	*Description*
`sin(radians)`	Returns the trigonometric sine of an angle in radians.
`cos(radians)`	Returns the trigonometric cosine of an angle in radians.
`tan(radians)`	Returns the trigonometric tangent of an angle in radians.
`toRadians(degree)`	Returns the angle in radians for the angle in degree.
`toDegree(radians)`	Returns the angle in degrees for the angle in radians.
`asin(a)`	Returns the angle in radians for the inverse of sine.
`acos(a)`	Returns the angle in radians for the inverse of cosine.
`atan(a)`	Returns the angle in radians for the inverse of tangent.

TABLE 4.2 Exponent Methods in the Math Class

Method	*Description*
`exp(x)`	Returns e raised to power of x (e^x).
`log(x)`	Returns the natural logarithm of x ($\ln(x) = \log_e(x)$).
`log10(x)`	Returns the base 10 logarithm of x ($\log_{10}(x)$).
`pow(a, b)`	Returns a raised to the power of b (a^b).
`sqrt(x)`	Returns the square root of x ($\sqrt{x}$) for x >= 0.

TABLE 4.3 Rounding Methods in the Math Class

Method	*Description*
`ceil(x)`	x is rounded up to its nearest integer. This integer is returned as a double value.
`floor(x)`	x is rounded down to its nearest integer. This integer is returned as a double value.
`rint(x)`	x is rounded up to its nearest integer. If x is equally close to two integers, the even one is returned as a double value.
`round(x)`	Returns (int)Math.floor(x + 0.5) if x is a float and returns (long)Math.floor(x + 0.5) if x is a double.

CS-1150

Exam #1 Reference Material

TABLE 4.6 Methods in the Character Class

Method	*Description*
`isDigit(ch)`	Returns true if the specified character is a digit.
`isLetter(ch)`	Returns true if the specified character is a letter.
`isLetterOfDigit(ch)`	Returns true if the specified character is a letter or digit.
`isLowerCase(ch)`	Returns true if the specified character is a lowercase letter.
`isUpperCase(ch)`	Returns true if the specified character is an uppercase letter.
`toLowerCase(ch)`	Returns the lowercase of the specified character.
`toUpperCase(ch)`	Returns the uppercase of the specified character.

TABLE 4.7 Simple Methods for `String` Objects

Method	*Description*
`length()`	Returns the number of characters in this string.
`charAt(index)`	Returns the character at the specified index from this string.
`concat(s1)`	Returns a new string that concatenates this string with string s1.
`toUpperCase()`	Returns a new string with all letters in uppercase.
`toLowerCase()`	Returns a new string with all letters in lowercase
`trim()`	Returns a new string with whitespace characters trimmed on both sides.

TABLE 4.8 Comparison Methods for `String` Objects

Method	*Description*
`equals(s1)`	Returns true if this string is equal to string `s1`.
`equalsIgnoreCase(s1)`	Returns true if this string is equal to string `s1`; it is case insensitive.
`compareTo(s1)`	Returns an integer greater than 0, equal to 0, or less than 0 to indicate whether this string is greater than, equal to, or less than `s1`.
`compareToIgnoreCase(s1)`	Same as `compareTo` except that the comparison is case insensitive.
`startsWith(prefix)`	Returns true if this string starts with the specified prefix.
`endsWith(suffix)`	Returns true if this string ends with the specified suffix.
`contains(s1)`	Returns true if s1 is a substring in this string.

CS-1150

Exam #1 Reference Material

TABLE 4.9 The `String` class contains the methods for obtaining substrings.

Method	*Description*
`substring(beginIndex)`	Returns this string's substring that begins with the character at the specified `beginIndex` and extends to the end of the string, as shown in Figure 4.2.
`substring(beginIndex, endIndex)`	Returns this string's substring that begins at the specified `beginIndex` and extends to the character at index `endIndex - 1`, as shown in Figure 4.2. Note that the character at `endIndex` is not part of the substring.

TABLE 4.10 The `String` class contains the methods for finding substrings.

Method	*Description*
`indexOf(ch)`	Returns the index of the first occurrence of `ch` in the string. Returns `-1` if not matched.
`indexOf(ch, fromIndex)`	Returns the index of the first occurrence of `ch` after `fromIndex` in the string. Returns `-1` if not matched.
`indexOf(s)`	Returns the index of the first occurrence of string `s` in this string. Returns `-1` if not matched.
`indexOf(s, fromIndex)`	Returns the index of the first occurrence of string `s` in this string after `fromIndex`. Returns `-1` if not matched.
`lastIndexOf(ch)`	Returns the index of the last occurrence of `ch` in the string. Returns `-1` if not matched.
`lastIndexOf(ch, fromIndex)`	Returns the index of the last occurrence of `ch` before `fromIndex` in this string. Returns `-1` if not matched.
`lastIndexOf(s)`	Returns the index of the last occurrence of string `s`. Returns `-1` if not matched.
`lastIndexOf(s, fromIndex)`	Returns the index of the last occurrence of string `s` before `fromIndex`. Returns `-1` if not matched.

CS-1150

Exam #1 Reference Material

TABLE 4.5 Escape Sequences

Escape Sequence	*Name*	*Unicode Code*	*Decimal Value*
`\b`	Backspace	`\u0008`	8
`\t`	Tab	`\u0009`	9
`\n`	Linefeed	`\u000A`	10
`\f`	Formfeed	`\u000C`	12
`\r`	Carriage Return	`\u000D`	13
`\\`	Backslash	`\u005C`	92
`\"`	Double Quote	`\u0022`	34

TABLE 4.11 Frequently Used Format Specifiers

Format Specifier	*Output*	*Example*
`%b`	a Boolean value	true or false
`%c`	a character	'a'
`%d`	a decimal integer	200
`%f`	a floating-point number	45.460000
`%e`	a number in standard scientific notation	4.556000e+01
`%s`	a string	"Java is cool"

TABLE 4.12 Examples of Specifying Width and Precision

Example	*Output*
`%5c`	Output the character and add four spaces before the character item, because the width is 5.
`%6b`	Output the Boolean value and add one space before the false value and two spaces before the true value.
`%5d`	Output the integer item with width at least 5. If the number of digits in the item is < 5, add spaces before the number. If the number of digits in the item is > 5, the width is automatically increased.
`%10.2f`	Output the floating-point item with width at least 10 including a decimal point and two digits after the point. Thus, there are 7 digits allocated before the decimal point. If the number of digits before the decimal point in the item is < 7, add spaces before the number. If the number of digits before the decimal point in the item is > 7, the width is automatically increased.
`%10.2e`	Output the floating-point item with width at least 10 including a decimal point, two digits after the point and the exponent part. If the displayed number in scientific notation has width less than 10, add spaces before the number.
`%12s`	Output the string with width at least 12 characters. If the string item has fewer than 12 characters, add spaces before the string. If the string item has more than 12 characters, the width is automatically increased.

CS-1150

Exam #1 Reference Material

TABLE B.1 ASCII Character Set in the Decimal Index

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
0	nul	soh	stx	etx	eot	enq	ack	bel	bs	ht
1	nl	vt	ff	cr	so	si	dle	dcl	dc2	dc3
2	dc4	nak	syn	etb	can	em	sub	esc	fs	gs
3	rs	us	sp	!	”	#	$	%	&	’
4	(	)	*	+	,	–	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	–	’	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~	del		

TABLE B.2 ASCII Character Set in the Hexadecimal Index

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*A*	*B*	*C*	*D*	*E*	*F*
0	nul	soh	stx	etx	eot	enq	ack	bel	bs	ht	nl	vt	ff	cr	so	si
1	dle	dcl	dc2	dc3	dc4	nak	syn	etb	can	em	sub	esc	fs	gs	rs	us
2	sp	!	”	#	$	%	&	’	(	)	*	+	,	–	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	–
6	’	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	del