Chapter 8
Characters and Strings

Objectives

- To represent characters using the **char** type
- To encode characters using ASCII and Unicode
- To represent special characters using the escape sequences
- To cast a numeric value to a character and cast a character to an integer
- To compare and test characters using the static methods in the **Character** class
- To introduce objects and instance methods
- To represent strings using the **String** objects

Character Data Type

Four hexadecimal digits.

char letter = 'A'; (ASCII)

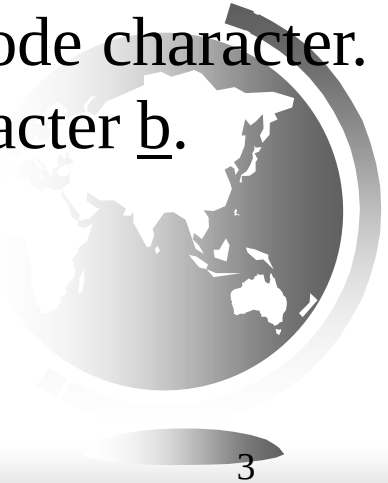
char numChar = '4'; (ASCII)

char letter = '\u0041'; (Unicode)

char numChar = '\u0034'; (Unicode)

NOTE: The increment and decrement operators can also be used on char variables to get the next or preceding Unicode character. For example, the following statements display character b.

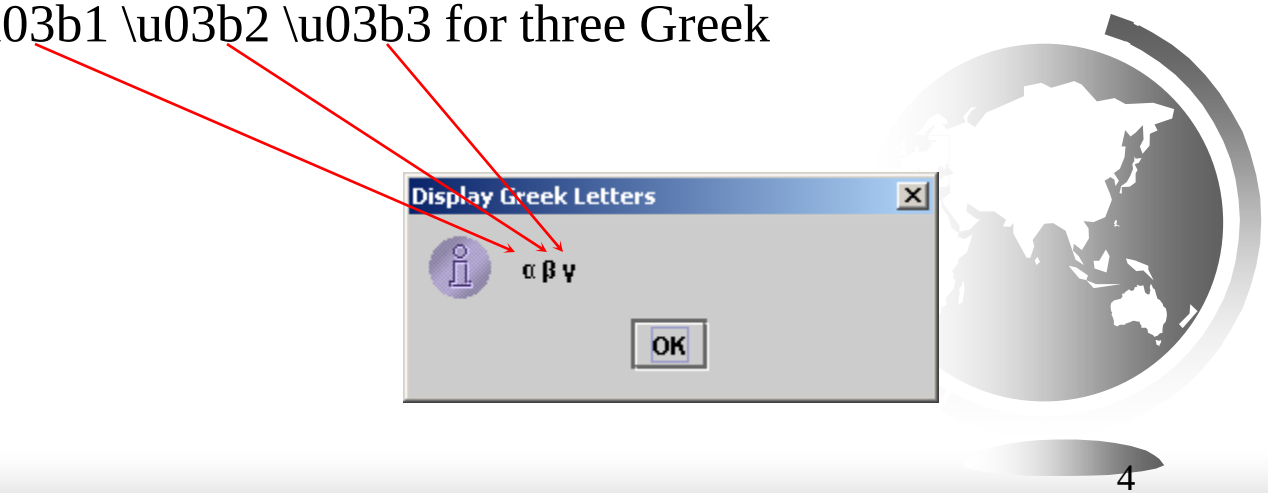
```
char ch = 'a';
System.out.println(++ch);
```



Unicode Format

Java characters use *Unicode*, a 16-bit encoding scheme established by the Unicode Consortium to support the interchange, processing, and display of written texts in the world's diverse languages. Unicode takes two bytes, preceded by \u, expressed in four hexadecimal numbers that run from '\u0000' to '\uFFFF'. So, Unicode can represent `65536 characters`.

Unicode \u03b1 \u03b2 \u03b3 for three Greek letters



ASCII Code for Commonly Used Characters

Most computers use ASCII (*American Standard Code for Information Interchange*), an 8-bit encoding scheme (128 characters), for representing all uppercase and lowercase letters, digits, punctuation marks, and control characters.

Characters	Code Value in Decimal	Unicode Value
`'0'` to `'9'`	48 to 57	\u0030 to \u0039
`'A'` to `'Z'`	65 to 90	\u0041 to \u005A
`'a'` to `'z'`	97 to 122	\u0061 to \u007A

Appendix B: ASCII Character Set

ASCII Character Set is a subset of the Unicode from \u0000 to \u007f

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		

ASCII Character Set, cont.

ASCII Character Set is a subset of the Unicode from \u0000 to \u007f

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

Escape Sequences for Special Characters

Can you write a statement like this?

System.out.println(“He said “Java is Fun” ”);

No. this statement has a compile error. The compiler thinks the second quotation character is the end of the string and does not know what to do with the rest of the characters.

Escape Sequences for Special Characters

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

```
System.out.println("He said \"Java is Fun\" ");
System.out.println("\\t is a tab character.");
```

Casting between char and Numeric Types

```
int i = 'a'; // Same as int i = (int)'a';

System.out.println(i); // 97


char c = 97; // Same as char c = (char)97;
```

Casting between char and Numeric Types

```
char ch = (char)65.25;          // Decimal 65 is assigned to ch
System.out.println(ch);         // ch is character A

int i = (int) ‘ A’ ;            // the Unicode of character A is
System.out.println(i);          assigned to i, i is 65

int i = ‘2’ + ‘3’;              // (int)’2’ is 50 and (int)’3’ is 51
System.out.println(“i is “ + i); // is is 101
```

Casting between char and Numeric Types

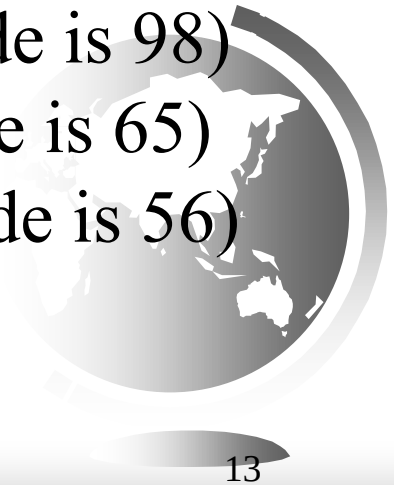
```
int j = 2 + 'a';                          // (int)'a' is 97
System.out.println("j is " + j);          // j is 99

System.out.println(j + "is the
Unicode for character " +                 // 99 is the Unicode for character c
(char)j);
System.out.println("Chapter "
+ '2');                                   // Chapter 2
```

Comparing and Testing Characters

Two characters can be compared using the relational operators.

This is done by comparing the Unicodes of the two characters.

‘a’ < ‘b’ is true: ‘a’ (Unicode is 97), ‘b’ (Unicode is 98)
‘a’ > ‘A’ is false: ‘a’ is greater than ‘A’ (Unicode is 65)
‘1’ < ‘8’ is true: ‘1’ (Unicode is 49), ‘8’ (Unicode is 56)



Comparing and Testing Characters

```
if (ch >= 'A' && ch <= 'Z')
  System.out.println(ch + " is an uppercase letter");
else if (ch >= 'a' && ch <= 'z')
  System.out.println(ch + " is a lowercase letter");
else if (ch >= '0' && ch <= '9')
  System.out.println(ch + " is a numeric character");
```

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.

Methods in the Character Class

```
System.out.println("isDigit('a') is " + Character.isDigit('a'));
System.out.println("isLetter('a') is " + Character.isLetter('a'));
System.out.println("isLowerCase('a') is " + Character.isLowerCase('a'));
System.out.println("isUpperCase('a') is " + Character.isUpperCase('a'));
System.out.println("toLowerCase('T') is " + Character.toLowerCase('T'));
System.out.println("toUpperCase('q') is " + Character.toUpperCase('q'));
```

```
isDigit('a') is false
isLetter('a') is true
isLowerCase('a') is true
isUpperCase('a') is false
toLowerCase('T') is t
toUpperCase('q') is Q
```

The String Type

The char type only represents one character. To represent a string of characters, use the data type called String. For example,

```
String message = "Welcome to Java";
```

String is actually a predefined class in the Java library just like the System class and Scanner class. The String type is not a primitive type. It is known as a *reference type*. Any Java class can be used as a reference type for a variable. For the time being, you just need to know how to declare a String variable, how to assign a string to the variable, how to concatenate strings, and to perform simple operations for strings.

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.

Simple Methods for **String** Objects

Strings are objects in Java. The methods in the preceding table can only be invoked from a specific string instance. For this reason, these methods are called *instance methods*. A non-instance method is called a *static method*. A static method can be invoked without using an object. All the methods defined in the **Math** class are static methods. They are not tied to a specific object instance. The syntax to invoke an instance method is

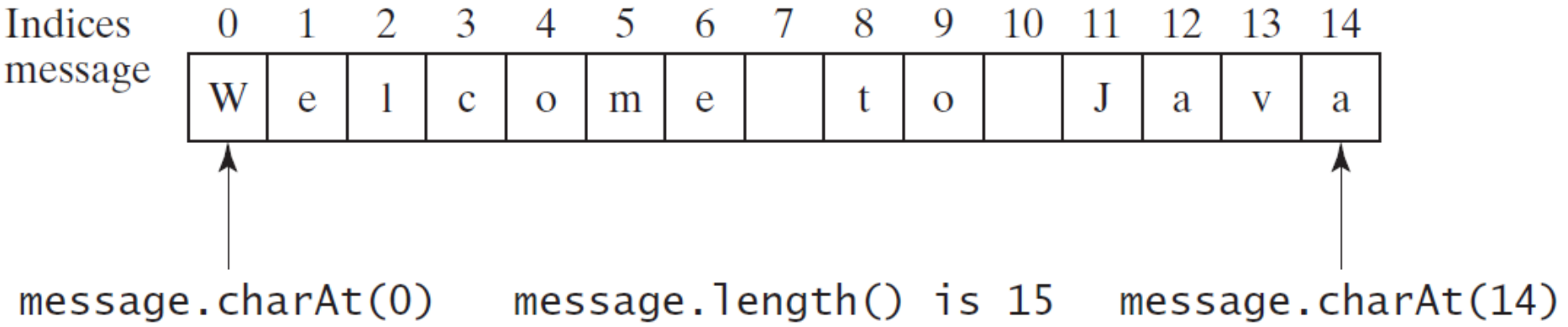
referenceVariable.methodName(arguments).

Getting String Length

```
String message = "Welcome to Java";
System.out.println("The length of " + message + " is "
  + message.length());
```

The length of Welcome to Java is 15

Getting Characters from a String



String message = **"Welcome to Java"**;

System.out.println(**"The first character in message is "**

+ message.charAt(0));

Converting Strings

"Welcome".toLowerCase() returns a new string, welcome.

"Welcome".toUpperCase() returns a new string, WELCOME.

" Welcome ".trim() returns a new string, Welcome.

String Concatenation

String s3 = s1.concat(s2); or String s3 = s1 + s2;

```
// Three strings are concatenated
String message = "Welcome " + "to " + "Java";

// String Chapter is concatenated with number 2
String s = "Chapter" + 2; // s becomes Chapter2

// String Supplement is concatenated with character B
String s1 = "Supplement" + 'B'; // s1 becomes SupplementB
```

Reading a String from the Console

```
Scanner input = new Scanner(System.in);
System.out.print("Enter three words separated by spaces: ");
String s1 = input.next();
String s2 = input.next();
String s3 = input.next();
System.out.println("s1 is " + s1);
System.out.println("s2 is " + s2);
System.out.println("s3 is " + s3);
```

Reading a Character from the Console

```
Scanner input = new Scanner(System.in);
System.out.print("Enter a character: ");
String s = input.nextLine();
char ch = s.charAt(0);
System.out.println("The character entered is " + ch);
```

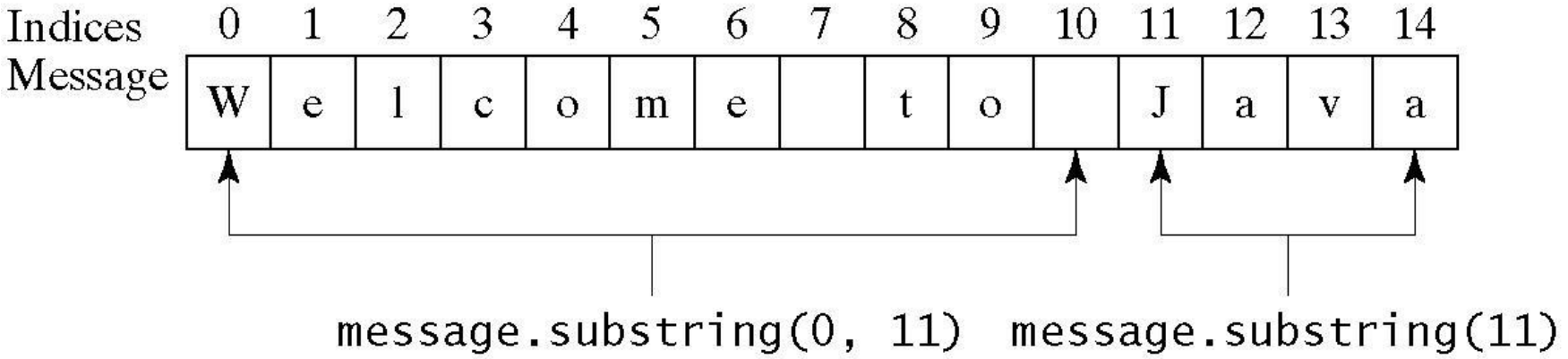
Comparing Strings

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.

OrderTwoCities

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 9.6. Note that the character at `endIndex` is not part of the substring.

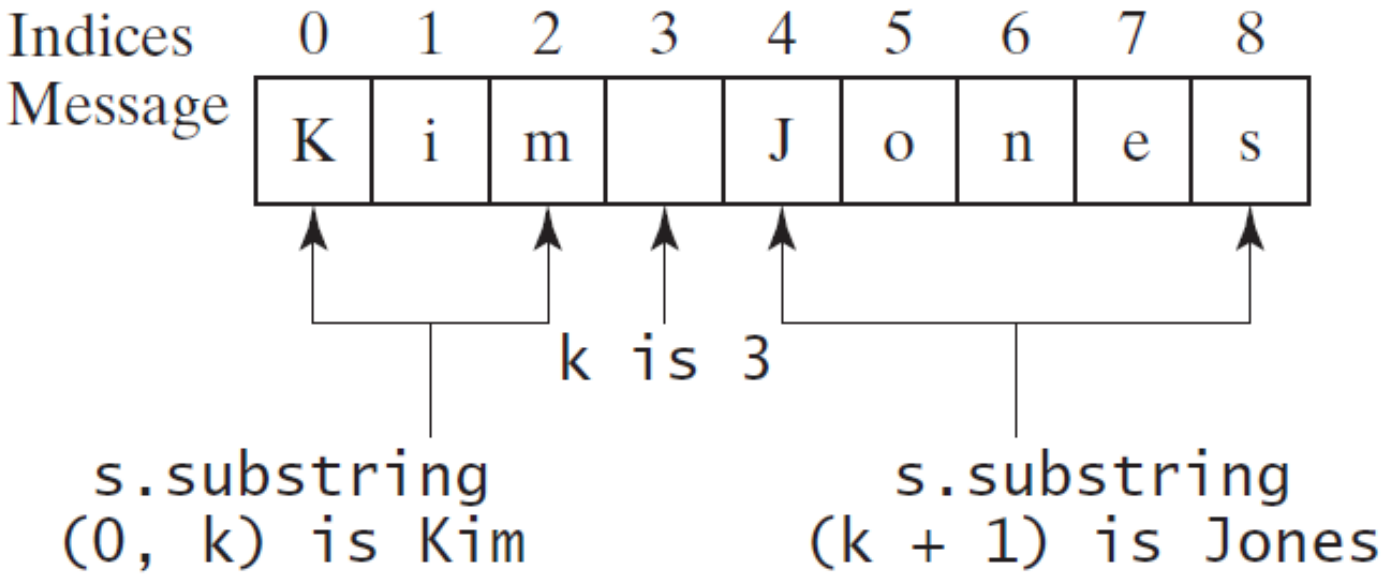


Finding a Character or a Substring in a String

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.

Finding a Character or a Substring in a String

```
int k = s.indexOf(' ');
String firstName = s.substring(0, k);
String lastName = s.substring(k + 1);
```

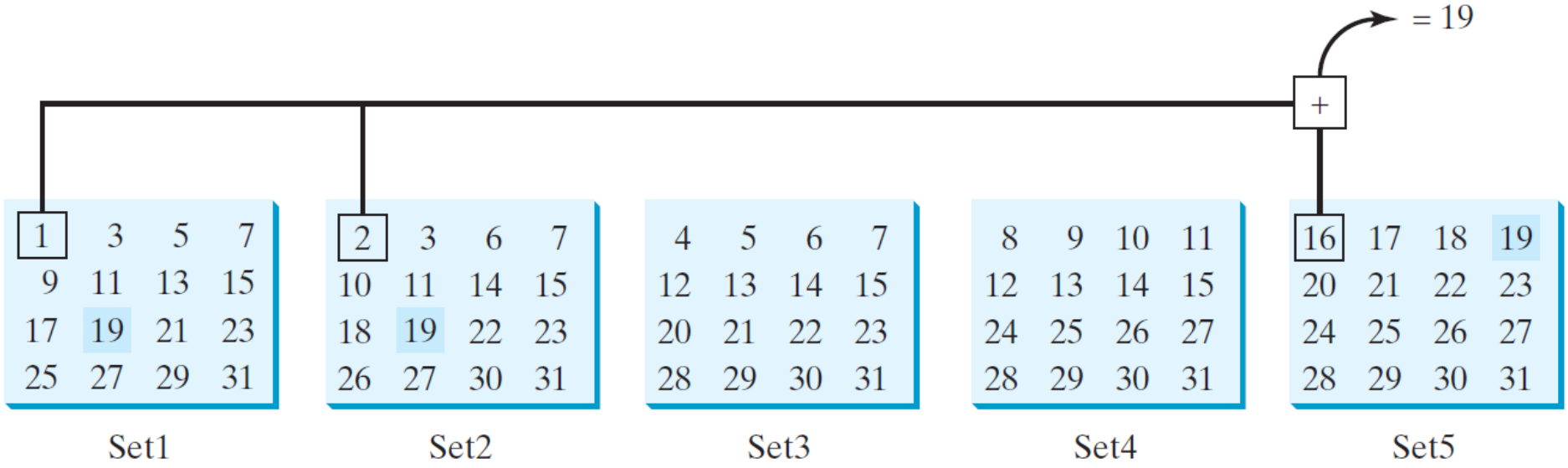


Conversion between Strings and Numbers

```
int intValue = Integer.parseInt(intString);
double doubleValue = Double.parseDouble(doubleString);

String s = number + "";
```

Problem: Guessing Birthday

The program can guess your birth date. Run to see how it works.



GuessBirthday

Mathematics Basis for the Game

19 is 10011 in binary. 7 is 111 in binary. 23 is 11101 in binary

```
  10000          00110          10000
     10             10           1000
+     1        +     1            100
  -----          -----      +     1
  10011          00111        -----
                              11101
```

19 7 23

Decimal	Binary
1	00001
2	00010
3	00011
...	
19	10011
...	
31	11111

$$
\begin{array}{r}
b_5\ 0\ 0\ 0\ 0 \\
b_4\ 0\ 0\ 0 \\
b_3\ 0\ 0 \\
b_2\ 0 \\
+\qquad b_1 \\
\hline
b_5\, b_4\, b_3\, b_2\, b_1
\end{array}
\qquad
\begin{array}{r}
10000 \\
10 \\
+\quad 1 \\
\hline
10011 \\
19
\end{array}
\qquad
\begin{array}{r}
10000 \\
1000 \\
100 \\
10 \\
+\quad 1 \\
\hline
11111 \\
31
\end{array}
$$

Case Study: Converting a Hexadecimal Digit to a Decimal Value

Write a program that converts a hexadecimal digit into a decimal value.

HexDigit2Dec

Case Study: Revising the Lottery Program Using Strings

A problem can be solved using many different approaches. This section rewrites the lottery program in Listing 3.7 using strings. Using strings simplifies this program.

LotteryUsingStrings

Formatting Output

Use the printf statement.

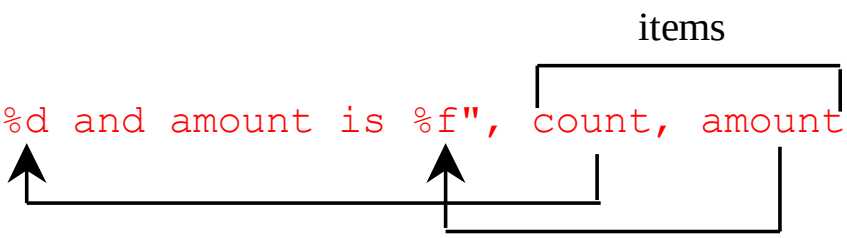
System.out.printf(format, items);

Where format is a string that may consist of substrings and format specifiers. A format specifier specifies how an item should be displayed. An item may be a numeric value, character, boolean value, or a string. Each specifier begins with a percent sign.

Frequently-Used Specifiers

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"

```
int count = 5;
double amount = 45.56;
System.out.printf("count is %d and amount is %f", count, amount);
```




```
display          count is 5 and amount is 45.560000
```

FormatDemo

The example gives a program that uses **printf** to display a table.

FormatDemo

Tutorial 1

Suppose the s1, s2, and s3 are three strings, given as follows:

String s1 = “Malaysia Holiday”;

String s2 = “Malaysia holiday”;

String s3 = new String (“Holiday”);

What are the results of the following expressions?

s3.equals(s1.substring(9));

s1.length();

s1.indexOf('e');

s3.concat(s1)

Tutorial 1

Suppose the s1, s2, and s3 are three strings, given as follows:

String s1 = “Malaysia Holiday”;

String s2 = “Malaysia holiday”;

String s3 = new String (“Holiday”);

What are the results of the following expressions?

s3.equals(s1.substring(9));	true
s1.length();	16
s1.indexOf('e');	-1
s3.concat(s1)	Holiday Malaysia Holiday

Tutorial 2

Suppose the s1, s2, and s3 are three strings, given as follows:

String s1 = “Welcome to Java Programming”;

String s2 = s1;

String s3 = new String (“Welcome to Java Programming”);

What are the results of the following expressions?

s2 == s3;

s1.length();

s1.substring (5,11)

s1.charAt(9);

s3.replace('a', 's')

Tutorial 2

Suppose the s1, s2, and s3 are three strings, given as follows:

String s1 = “Welcome to Java Programming”;

String s2 = s1;

String s3 = new String (“Welcome to Java Programming”);

What are the results of the following expressions?

s2 == s3;	false
s1.length();	27
s1.substring (5,11)	me to
s1.charAt(9);	o
s3.replace('a', 's')	Welcome to Jsvs Progrsmming