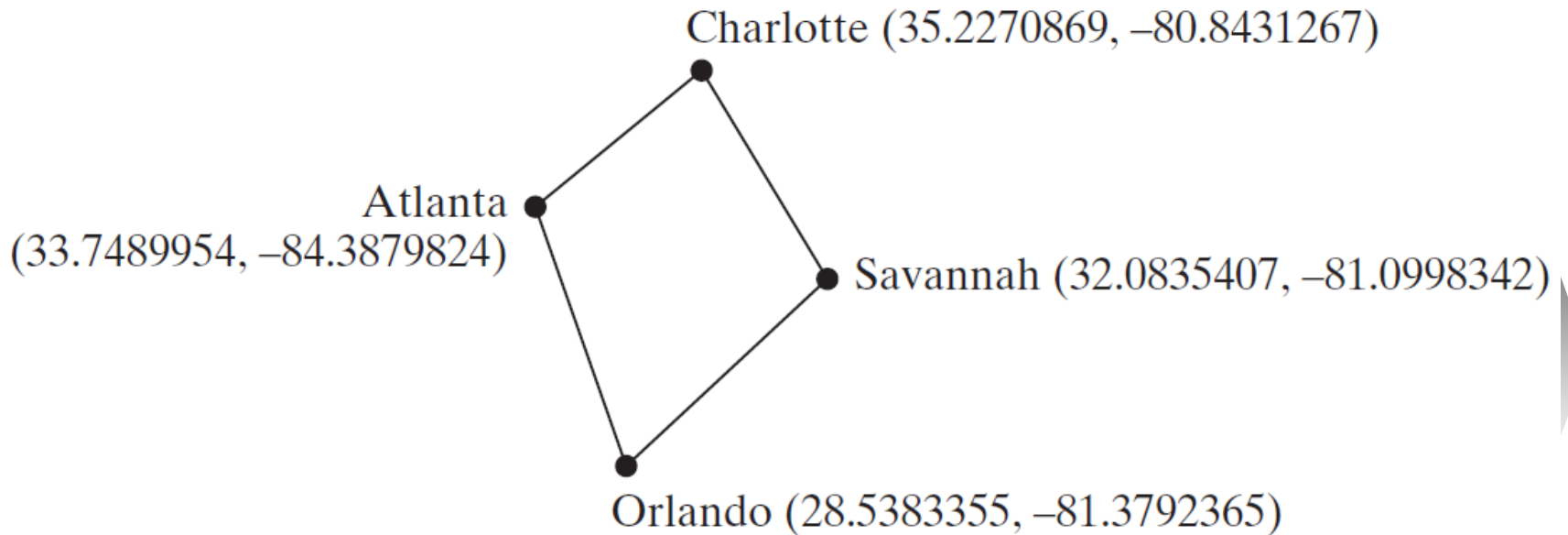


04 Mathematical Functions, Characters, and Strings



Motivations

Suppose you need to estimate the area enclosed by four cities, given the GPS locations (latitude and longitude) of these cities, as shown in the following diagram. How would you write a program to solve this problem? You will be able to write such a program after completing this chapter.



Objectives

- To solve mathematics problems by using the methods in the **Math** class (§4.2).
- To represent characters using the **char** type (§4.3).
- To encode characters using ASCII and Unicode (§4.3.1).
- To represent special characters using the escape sequences (§4.4.2).
- To cast a numeric value to a character and cast a character to an integer (§4.3.3).
- To compare and test characters using the static methods in the **Character** class (§4.3.4).
- To introduce objects and instance methods (§4.4).
- To represent strings using the **String** objects (§4.4).
- To return the string length using the **length()** method (§4.4.1).
- To return a character in the string using the **charAt(i)** method (§4.4.2).
- To use the + operator to concatenate strings (§4.4.3).
- To read strings from the console (§4.4.4).
- To read a character from the console (§4.4.5).
- To compare strings using the **equals** method and the **compareTo** methods (§4.4.6).
- To obtain substrings (§4.4.7).
- To find a character or a substring in a string using the **indexOf** method (§4.4.8).
- To program using characters and strings (**GuessBirthday**) (§4.5.1).
- To convert a hexadecimal character to a decimal value (**HexDigit2Dec**) (§4.5.2).
- To revise the lottery program using strings (**LotteryUsingStrings**) (§4.5.3).
- To format output using the **System.out.printf** method (§4.6).



Mathematical Functions

Java provides many useful methods in the **Math** class for performing common mathematical functions.



The Math Class

- Class constants:
 - π
 - e
- Class methods:
 - Trigonometric Methods
 - Exponent Methods
 - Rounding Methods
 - min, max, abs, and random Methods



Trigonometric Methods

`sin(double a)`

`cos(double a)`

`tan(double a)`

`acos(double a)`

`asin(double a)`

`atan(double a)`

Radians

`toRadians(90)`

Examples:

`Math.sin(0)` returns 0.0

`Math.sin(Math.PI / 6)`
returns 0.5

`Math.sin(Math.PI / 2)`
returns 1.0

`Math.cos(0)` returns 1.0

`Math.cos(Math.PI / 6)`
returns 0.866

`Math.cos(Math.PI / 2)`
returns 0



Exponent Methods

- **`exp(double a)`**
Returns e raised to the power of a .
- **`log(double a)`**
Returns the natural logarithm of a .
- **`log10(double a)`**
Returns the 10-based logarithm of a .
- **`pow(double a, double b)`**
Returns a raised to the power of b .
- **`sqrt(double a)`**
Returns the square root of a .

Examples:

`Math.exp(1)` returns 2.71

`Math.log(2.71)` returns 1.0

`Math.pow(2, 3)` returns 8.0

`Math.pow(3, 2)` returns 9.0

`Math.pow(3.5, 2.5)` returns
22.91765

`Math.sqrt(4)` returns 2.0

`Math.sqrt(10.5)` returns 3.24



Rounding Methods

- **double ceil(double x)**
x rounded up to its nearest integer. This integer is returned as a double value.
- **double floor(double x)**
x is rounded down to its nearest integer. This integer is returned as a double value.
- **double rint(double x)**
x is rounded to its nearest integer. If x is equally close to two integers, the **even one is returned** as a double.

- **int round(float x)**
Return (int)Math.floor(x+0.5).
- **long round(double x)**
Return (long)Math.floor(x+0.5).



Rounding Methods Examples

```
Math.ceil(2.1) returns 3.0
Math.ceil(2.0) returns 2.0
Math.ceil(-2.0) returns -2.0
Math.ceil(-2.1) returns -2.0
Math.floor(2.1) returns 2.0
Math.floor(2.0) returns 2.0
Math.floor(-2.0) returns -2.0
Math.floor(-2.1) returns -3.0
Math rint(2.1) returns 2.0
Math rint(2.0) returns 2.0
Math rint(-2.0) returns -2.0
Math rint(-2.1) returns -2.0
Math rint(2.5) returns 2.0
Math rint(-2.5) returns -2.0
Math.round(2.6f) returns 3
Math.round(2.0) returns 2
Math.round(-2.0f) returns -2
Math.round(-2.6) returns -3
```



Which of the following will output -3.0 (2分)

- ☐ A. `System.out.println(Math.floor(-3.7));`
- ☐ B. `System.out.println(Math.round(-3.7));`
- ☐ C. `System.out.println(Math.ceil(-3.7));`
- ☐ D. `System.out.println(Math.min(-3.7));`



min, max, and abs

- `max(a, b)` and `min(a, b)`
Returns the maximum or minimum of two parameters.
- `abs(a)`
Returns the absolute value of the parameter.
- `random()`
Returns a random double value in the range [0.0, 1.0).

Examples:

`Math.max(2, 3)` returns 3

`Math.max(2.5, 3)` returns 3.0

`Math.min(2.5, 3.6)` returns 2.5

`Math.abs(-2)` returns 2

`Math.abs(-2.1)` returns 2.1



The random Method

Generates a random double value greater than or equal to 0.0 and less than 1.0 ($0 \leq \text{Math.random()} < 1.0$).

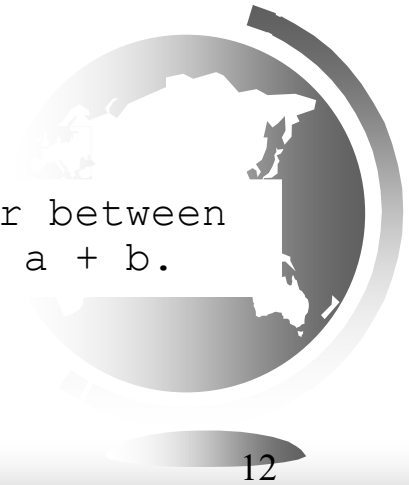
Examples:

`(int) (Math.random() * 10)` → Returns a random integer between 0 and 9.

`50 + (int) (Math.random() * 50)` → Returns a random integer between 50 and 99.

In general,

`a + Math.random() * b` → Returns a random number between a and a + b, excluding a + b.



Which expression below is for generating a random number of [20 , 999]?

A. `(int) (20+Math.random()*97)`

B. `20+(int) (Math.random()*980)`

C. `(int)Math.random()*999`

D. `20+(int)Math.random()*980`



Character Data Type

`char letter = 'A'; (ASCII)`

`char numChar = '4'; (ASCII)`

`char letter = '\u0041'; (Unicode)`

`char numChar = '\u0034'; (Unicode)`

Four hexadecimal digits.

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';
```

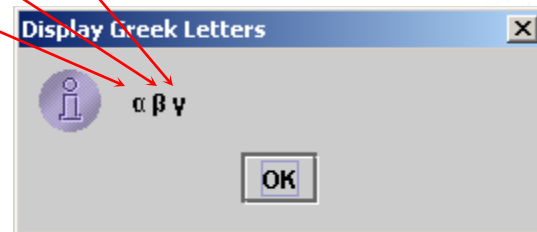
```
System.out.println(++ch);
```



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 65535 + 1 characters.

Unicode \u03b1 \u03b2 \u03b3 for three Greek letters



ASCII Code for Commonly Used 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



Escape Sequences for Special Characters

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



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

	<i>0</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>7</i>	<i>8</i>	<i>9</i>
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

需要注意注释中的“\u”，例如：// \u000A is a newline

// Look inside c:\users

会出现语法错误，因为\u后并不是4个十六进制数

Casting between char and Numeric Types

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

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



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
<code>isDigit(ch)</code>	Returns true if the specified character is a digit.
<code>isLetter(ch)</code>	Returns true if the specified character is a letter.
<code>isLetterOfDigit(ch)</code>	Returns true if the specified character is a letter or digit.
<code>isLowerCase(ch)</code>	Returns true if the specified character is a lowercase letter.
<code>isUpperCase(ch)</code>	Returns true if the specified character is an uppercase letter.
<code>toLowerCase(ch)</code>	Returns the lowercase of the specified character.
<code>toUpperCase(ch)</code>	Returns the uppercase of the specified character.



The String Type

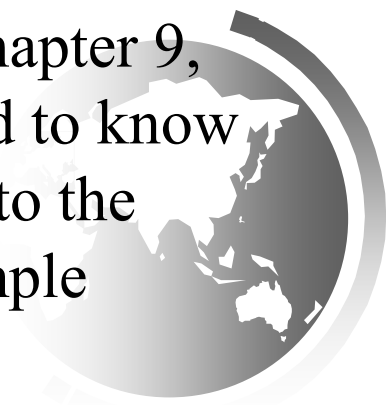
The char type only represents one character. To represent a string of characters, use the data type called String. For example,

```
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.

Reference data types will be thoroughly discussed in Chapter 9, “Objects and Classes.” 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
<code>length()</code>	Returns the number of characters in this string.
<code>charAt(index)</code>	Returns the character at the specified index from this string.
<code>concat(s1)</code>	Returns a new string that concatenates this string with string <code>s1</code> .
<code>toUpperCase()</code>	Returns a new string with all letters in uppercase.
<code>toLowerCase()</code>	Returns a new string with all letters in lowercase.
<code>trim()</code>	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).

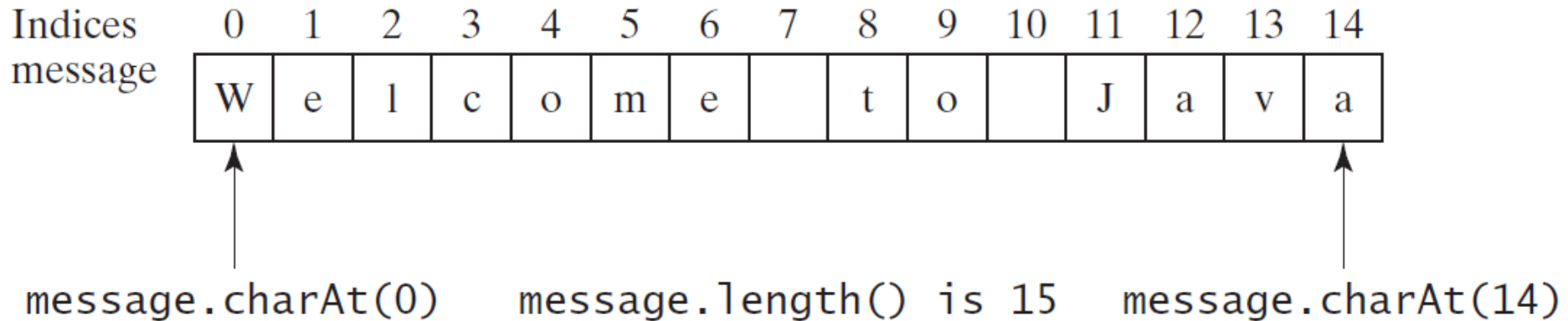


Getting String Length

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



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);
```



```

1 package cn.dx;
2
3 import java.util.Scanner;
4
5 public class ScannerTest {
6
7     public static void main(String[] args) {
8         Scanner in = new Scanner(System.in);
9         System.out.println("请输入一个整数");
10        while(in.hasNextInt()){
11            int num = in.nextInt();
12            System.out.println("请输入一个字符串");
13            String str = in.nextLine();
14            System.out.println("num="+num+",str="+str);
15            System.out.println("请输入一个整数");
16        }
17    }
18 }

```

运行结果为：

```

请输入一个整数
1231
请输入一个字符串
num=1231,str=
请输入一个整数

```

第二个String类型的参数没有读取进来。

- `nextInt()`方法会读取下一个int型标志的token.但是焦点不会移动到下一行，仍然处在这一行上。当使用`nextLine()`方法时会读取该行剩余的所有的内容，包括换行符，然后把焦点移动到下一行的开头。所以这样就无法接收到下一行输入的String类型的变量。



- `next()`从遇到第一个有效字符（非空格、换行符）开始扫描，遇到第一个分隔符或结束符（空格 ‘ ’ 或者换行符 ‘\n’）时结束。
- `nextLine()`则是扫描剩下的所有字符串直到遇到回车为止。（可以从空格开始）

```
Scanner sc = new Scanner(System.in);  
//加入输入的是:aaa bbb ccc  
str1=sc.next();  
//str1="aaa"  
str2=sc.nextLine();  
//str2=" bbb ccc"
```

`nextLine`是直接开始扫描，不管是什么；而`next()`是先要到有效字符开始



Comparing Strings

Method	Description
<code>equals(s1)</code>	Returns true if this string is equal to string <code>s1</code> .
<code>equalsIgnoreCase(s1)</code>	Returns true if this string is equal to string <code>s1</code> ; it is case insensitive.
<code>compareTo(s1)</code>	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 <code>s1</code> .
<code>compareToIgnoreCase(s1)</code>	Same as <code>compareTo</code> except that the comparison is case insensitive.
<code>startsWith(prefix)</code>	Returns true if this string starts with the specified prefix.
<code>endsWith(suffix)</code>	Returns true if this string ends with the specified suffix.

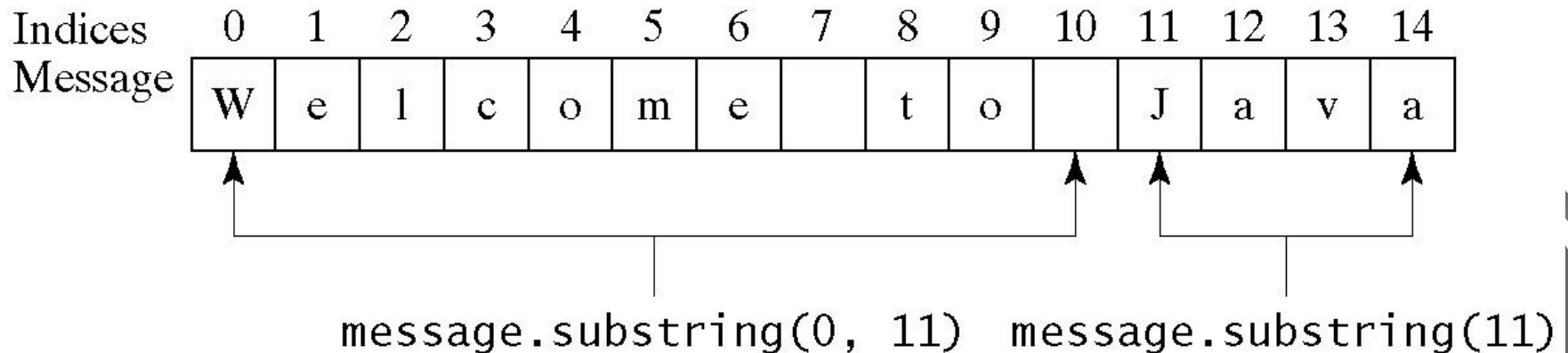


OrderTwoCities

Run

Obtaining Substrings

Method	Description
<code>substring(beginIndex)</code>	Returns this string's substring that begins with the character at the specified <code>beginIndex</code> and extends to the end of the string, as shown in Figure 4.2.
<code>substring(beginIndex, endIndex)</code>	Returns this string's substring that begins at the specified <code>beginIndex</code> and <u>extends to the character at index <code>endIndex - 1</code></u> , as shown in Figure 9.6. Note that the character at <code>endIndex</code> is not part of the substring.

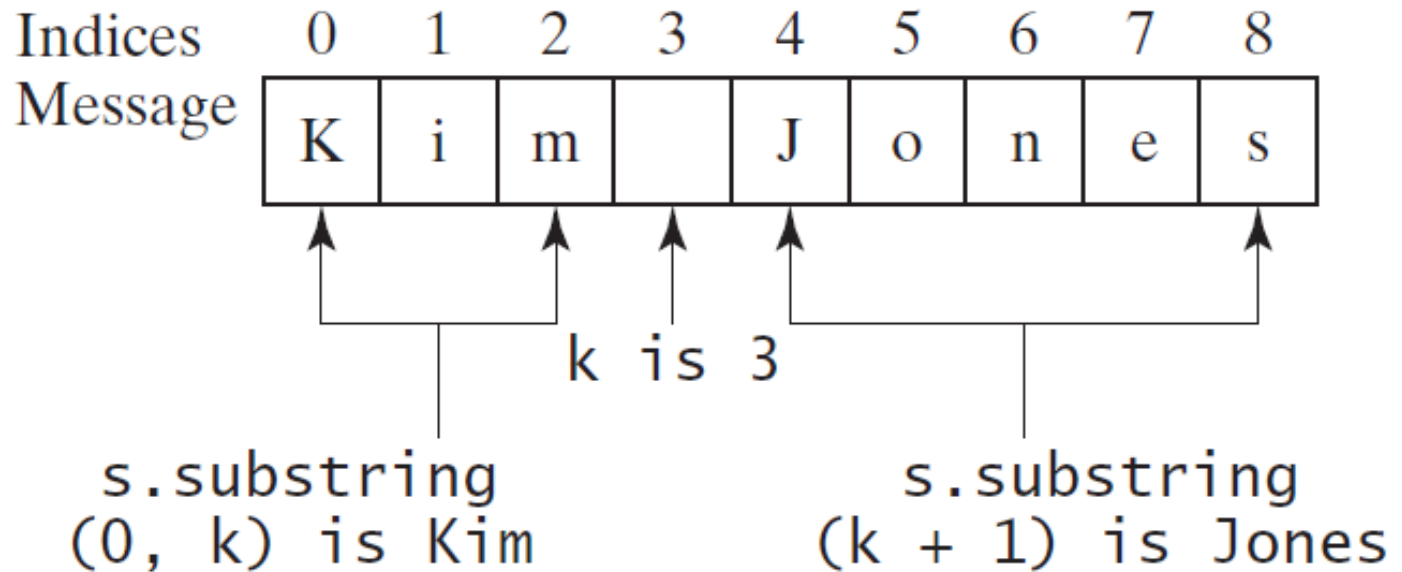


Finding a Character or a Substring in a String

Method	Description
<code>indexOf(ch)</code>	Returns the index of the first occurrence of <code>ch</code> in the string. Returns <code>-1</code> if not matched.
<code>indexOf(ch, fromIndex)</code>	Returns the index of the first occurrence of <code>ch</code> after <code>fromIndex</code> in the string. Returns <code>-1</code> if not matched.
<code>indexOf(s)</code>	Returns the index of the first occurrence of string <code>s</code> in this string. Returns <code>-1</code> if not matched.
<code>indexOf(s, fromIndex)</code>	Returns the index of the first occurrence of string <code>s</code> in this string after <code>fromIndex</code> . Returns <code>-1</code> if not matched.
<code>lastIndexOf(ch)</code>	Returns the index of the last occurrence of <code>ch</code> in the string. Returns <code>-1</code> if not matched.
<code>lastIndexOf(ch, fromIndex)</code>	Returns the index of the last occurrence of <code>ch</code> before <code>fromIndex</code> in this string. Returns <code>-1</code> if not matched.
<code>lastIndexOf(s)</code>	Returns the index of the last occurrence of string <code>s</code> . Returns <code>-1</code> if not matched.
<code>lastIndexOf(s, fromIndex)</code>	Returns the index of the last occurrence of string <code>s</code> before <code>fromIndex</code> . Returns <code>-1</code> 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 + "";
```



请写出以下程序运行结果：

```
class Main {  
    public static void main(String[] args) {  
        String s1 = "Zhejiang University";  
        String s2 = s1.substring(0, 7);  
        s2.toUpperCase();  
        System.out.println(s2+s1.substring(8));  
    }  
}
```

Zhejian University



Formatting Output

Use the printf statement.

```
System.out.printf(format, items);
```

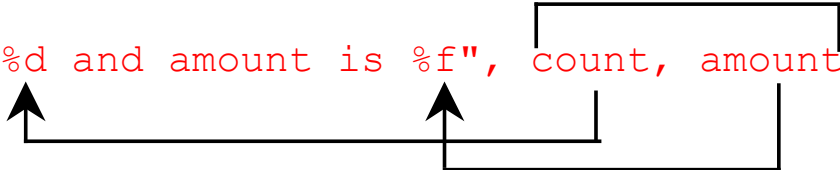
Where format is a string that may consist of substrings and format specifiers. A **format specifier** specifies how an item should be displayed. An **item** may be a numeric value, character, boolean value, or a string. **Each specifier begins with a percent sign.**



Frequently-Used Specifiers

Specifier	Output	Example
<code>%b</code>	a boolean value	true or false
<code>%c</code>	a character	'a'
<code>%d</code>	a decimal integer	200
<code>%f</code>	a floating-point number	45.460000
<code>%e</code>	a number in standard scientific notation	4.556000e+01
<code>%s</code>	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

占位符完整格式为：`%[index$][标识]*[最小宽度][.精度]转换符`。

针对不同数据类型的格式化，占位符的格式将有所裁剪。

`%`，占位符的其实字符，若要在占位符内部使用`%`，则需要写成`%%`。

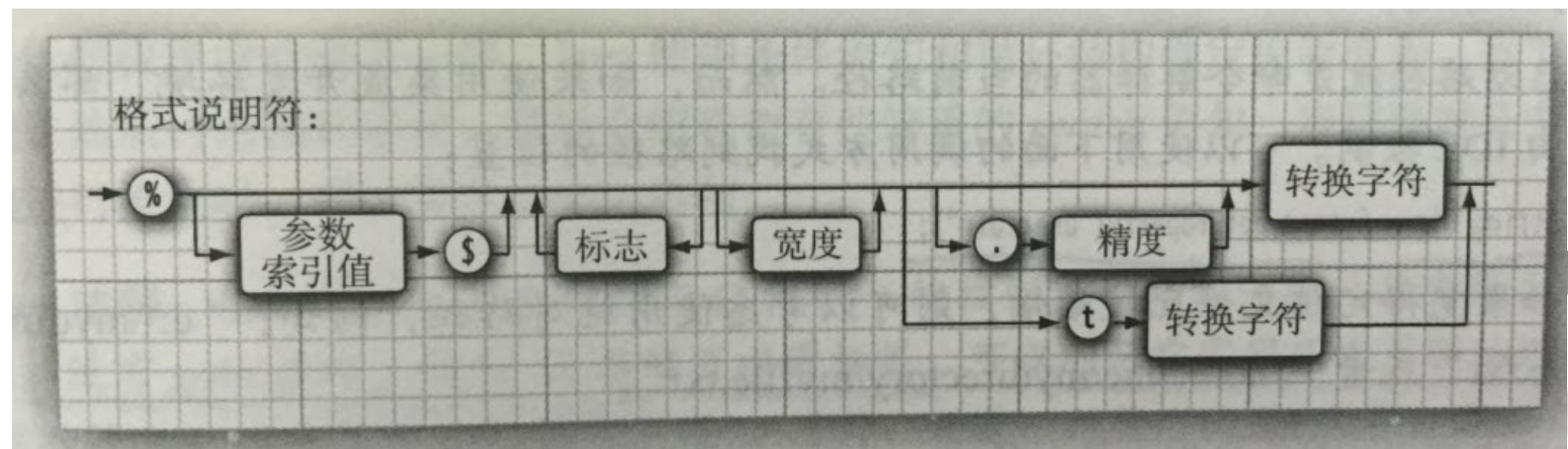
`[index$]`，位置索引从1开始计算，用于指定对索引相应的实参进行格式化并替换掉该占位符。

`[标识]`，用于增强格式化能力，可同时使用多个`[标识]`，但某些标识是不能同时使用的。

`[最小宽度]`，用于设置格式化后的字符串最小长度，若使用`[最小宽度]`而无设置`[标识]`，那么当字符串长度小于最小宽度时，则以左边补空格的方式凑够最小宽度。

`[.精度]`，对于浮点数类型格式化使用，设置保留小数点后多少位。

`转换符`，用于指定格式化的样式，和限制对应入参的数据类型。



四、对字符、字符串进行格式化

占位符格式为：`%[index$][标识][最小宽度]转换符`

示例——将"hello"格式化为" hello"

```
String raw = "hello";  
String str = String.format("%1$7s", raw);  
// 简化  
//String str = String.format("%7s", raw);
```

示例——将"hello"格式化为"hello "

```
String raw = "hello";  
String str = String.format("%1$-7s", raw);  
// 简化  
//String str = String.format("%-7s", raw);
```

可用标识：

`-`，在最小宽度内左对齐，右边用空格补上。

可用转换符：

`s`，字符串类型。

`c`，字符类型，实参必须为char或int、short等可转换为char类型的数据类型，否则抛IllegalFormatConversionException异常。

`b`，布尔类型，只要实参为非false的布尔类型，均格式化为字符串true，否则为字符串false。

`n`，平台独立的换行符（与通过 `System.getProperty("line.separator")` 是一样的）

五、对整数进行格式化

占位符格式为： `%[index$][标识]*[最小宽度]转换符`

示例——将1显示为0001

```
int num = 1;
String str = String.format("%04d", num)
```

示例——将-1000显示为(1,000)

```
int num = -1000;
String str = String.format("%(,d", num)
```

可用标识：



- ，在最小宽度内左对齐，不可以与0标识一起使用。
- 0，若内容长度不足最小宽度，则在左边用0来填充。
- #，对8进制和16进制，8进制前添加一个0，16进制前添加0x。
- ，结果总包含一个+或-号。
- 空格，正数前加空格，负数前加-号。
- ，，只用与十进制，每3位数字间用，分隔。
- （，若结果为负数，则用括号括住，且不显示符号。



可用转换符：

- b，布尔类型，只要实参为非false的布尔类型，均格式化为字符串true，否则为字符串false。
- d，整数类型（十进制）。
- x，整数类型（十六进制）。
- o，整数类型（八进制）。
- n，平台独立的换行符，也可通过System.getProperty("line.separator")获取



六、对浮点数进行格式化

占位符格式为：`%[index$][标识]*[最小宽度][.精度]转换符`

示例：

```
double num = 123.4567899;
System.out.print(String.format("%f %n", num)); // 123.456790
System.out.print(String.format("%a %n", num)); // 0x1.edd3c0bb46929p6
System.out.print(String.format("%g %n", num)); // 123.457
```

可用标识：



- ，在最小宽度内左对齐，不可以与0标识一起使用。
- 0，若内容长度不足最小宽度，则在左边用0来填充。
- #，对8进制和16进制，8进制前添加一个0，16进制前添加0x。
- ，结果总包含一个+或-号。
- 空格，正数前加空格，负数前加-号。
- ，，只用于十进制，每3位数字间用，分隔。
- (，若结果为负数，则用括号括住，且不显示符号。



可用转换符：



- b，布尔类型，只要实参为非false的布尔类型，均格式化为字符串true，否则为字符串false。
- n，平台独立的换行符，也可通过System.getProperty("line.separator")获取。
- f，浮点数值（十进制）。显示9位有效数字，且会进行四舍五入。如99.99。
- a，浮点数值（十六进制）。
- e，指数类型。如9.38e+5。
- g，浮点数值（比%f，%a长度短些，显示6位有效数字，且会进行四舍五入）



七、对日期时间进行格式化

占位符格式为：`%[index$]t转换符`

示例：

```
Date now = new Date();  
String str = String.format("%tF", now); // 2014-10-12
```

可用转换符

1. 日期的转换符



`c`, 星期六 十月 27 14:21:20 CST 2007

`F`, 2007-10-27

`D`, 10/27/07

`r`, 02:25:51 下午

`T`, 14:28:16

`R`, 14:28

`b`, 月份简称

`B`, 月份全称

`a`, 星期简称

`A`, 星期全称

`C`, 年前两位 (不足两位补零)

`y`, 年后两位 (不足两位补零)

`j`, 当年的第几天

`m`, 月份 (不足两位补零)

`d`, 日期 (不足两位补零)

`e`, 日期 (不足两位不补零)

2. 时间的转换符



`H`, 24小时制的小时 (不足两位补零)

`k`, 24小时制的小时 (不足两位不补零)

`I`, 12小时制的小时 (不足两位补零)

`i`, 12小时制的小时 (不足两位不补零)

`M`, 分钟 (不足两位补零)

`S`, 秒 (不足两位补零)

`L`, 毫秒 (不足三位补零)

`N`, 毫秒 (不足9位补零)

`p`, 小写字母的上午或下午标记, 如中文为“下午”, 英文为pm

`z`, 相对于GMT的时区偏移量, 如+0800

`Z`, 时区缩写, 如CST

`s`, 自1970-1-1 00:00:00起经过的秒数

`Q`, 自1970-1-1 00:00:00起经过的毫秒



对于参数索引index的说明

- 索引必须紧跟在%后面，以\$终止
- `System.out.printf(“%1$s %2$tB %2$te,%2$tY”, “Due date:”, new Date());`
- 输出为:
- Due date: February 9, 2015
- 还可以用<标志，表明前面格式说明中的参数被再次使用
- `System.out.printf(“%s %tB %<te, %<tY”, “Due date:”, new Date());`



FormatDemo

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

```
public class FormatDemo {  
    public static void main(String[] args) {  
        // Display the header of the table  
        System.out.printf("%-10s%-10s%-10s%-10s%-10s\n", "Degrees",  
            "Radians", "Sine", "Cosine", "Tangent");  
  
        // Display values for 30 degrees  
        int degrees = 30;  
        double radians = Math.toRadians(degrees);  
        System.out.printf("%-10d%-10.4f%-10.4f%-10.4f%-10.4f\n", degrees,  
            radians, Math.sin(radians), Math.cos(radians),  
            Math.tan(radians));  
  
        // Display values for 60 degrees  
        degrees = 60;  
        radians = Math.toRadians(degrees);  
        System.out.printf("%-10d%-10.4f%-10.4f%-10.4f%-10.4f\n", degrees,  
            radians, Math.sin(radians), Math.cos(radians),  
            Math.tan(radians));  
    }  
}
```



FormatDemo

Run

FormatDemo

Problems	Javadoc	Declaration	Console	
<terminated> FormatDemo [Java Application] C:\Program Files\Java\jc				
Degrees	Radians	Sine	Cosine	Tangent
30	0.5236	0.5000	0.8660	0.5774
60	1.0472	0.8660	0.5000	1.7321

