Key point: *A character data type represents a single character.*

4.3.1 Unicode and ASCII code

- A 16-bit Unicode takes two bytes, preceded by **\u**, expressed in four hexadecimal digits that run from **\u0000** to **\uFFFF**.
- Most computers use *ASCII* (*American Standard Code for Information Interchange*), an 8-bit encoding scheme, for representing all uppercase and lowercase letters, digits, punctuationmarks, and control characters. Unicode includes ASCII code with **\u0000** to **\u007F** corresponding to the 128 ASCII characters.

Table 4.4 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**

Examples:

```
char letter = 'A';         //Equivalent to '\u0041'
char letter = '\u0041';    //Equivalent to 'A'
```

4.3.2 Escape Sequences for Special Characters

```
System.out.println("He said "Java is fun"");     //Compilation error
System.out.println("He said \"Java is fun\"");   //Correction with escape sequences
```

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

```
System.out.println("\\t is a tab character");    //Displays \t is a tab character
```

4.3.3 Casting between char an Numeric Types

```
// Note a hex integer is written using prefix 0X
char ch = (char)0XAB0041;      //The lower 16 bits hex code 0041 is
                               //assigned to ch
System.out.println(ch);        //ch is character A
```

When a floating-point value is cast into a **char**, the floating-point value is first cast into an **int**, which is then cast into a **char**.

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

When a **char** is cast into a numeric type, the character’s Unicode is cast into the specified numeric type.

```
int i = (int)’A’;              //The Unicode of character A is assigned to variable i
System.out.println(i);         //i is character 65
```

Implicit casting can be used if the result of a casting fits into the target variable. Otherwise, explicit casting must be used. For example, since the Unicode of **‘a’** is **97**, which is within the range of a byte, these implicit casting are fine:

```
byte b = ‘a’;
int i = ‘a’;
```

But the following statement is incorrect, because the Unicode **\uFFF4** cannot fit into a byte:

```
byte b = (byte)’\uFFF4’;
```

Any positive integer between **0** and **FFFF** in hexadecimal can be cast into a character implicitly. Any number not in this range must be cast into **char** explicitly.

All numeric operators can be applied to **char** operands. A **char** operand is automatically cast into a number if the other operand is a number or a character. If the other operand is a string, the character is concatenated with string. For example, the following statements

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

int j = 2 + ‘a’;                  //(int)’a’ is 97
System.out.println(“j is “ + j);  // j is 99
//99 is the Unicode for character c
System.out.println(j + “ is the Unicode for character “ + (char)j);
System.out.println(“Chapter “ + ‘2’);
```

display

```
i is 101
j is 99
99 is the Unicode for character c
Chapter 2
```

4.3.4 Comparing and Testing Characters

Two characters can be compared using the relational operators just like comparing two numbers. This is done by comparing the Unicodes of the two characters.

‘a’ < ‘b’ is true because the Unicode for **‘a’** (**97**) is less than the Unicode for **‘b’** (**98**).
‘a’ > ‘A’ is true because the Unicode for **‘a’** (**97**) is greater than the Unicode for **‘A’** (**65**).
‘1’ < ‘8’ is true because the Unicode for **‘1’** (**49**) is greater than the Unicode for **‘8’** (**56**).

Often in a program, you need to test whether a character is a number, a letter, an uppercase letter, or a lowercase letter.

```
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”);
```

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.
isLetterOrDigit(ch)	Returns true if the specified character is a letter or a 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.

Examples:

```
System.out.println(“isDigit(‘a’) is “ + Character.isDigit(‘a’));
System.out.println(“isDigit(‘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’));
```

Displays

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