**CHAPTER 4: MATHEMATICAL FUNCTIONS, CHARACTERS, AND STRINGS**

* 1. **Common Mathematical Functions**

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

A method is a group of statements that performs a specific task.

They can be categorized as **trigonometric methods**, **exponent methods**, and **service methods**.

Trigonometric Methods in the Math Class

**TRIGONOMETRIC METHODS**

|  |  |
| --- | --- |
| 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 degrees. |
| 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. |

**EXPONENT METHODS**

There are five methods related to exponents in the **Math** class,

|  |  |
| --- | --- |
| Methods | Description |
| exp (x) | Returns e raised to power of |
| log (x) | Returns the natural logarithm of = . |
| log10 (x) | Returns the base 10 logarithm of . |
| pow (a,b) | Returns a raised to the power of |
| sqrt (x) | Returns the square root of . |

* 1. **The Rounding Methods**

The **Math** class contains four rounding methods

|  |  |
| --- | --- |
| 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 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. |

For example,

* {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(−2.1) returns −2.0 }, { Math.rint(2.5) returns 2.0 }, { Math.rint(3.5) returns 4.0 }, { 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 }, { Math.round(−2.4) returns −2 }.

**The min, max, and abs Methods**

The **min** and **max** methods return the minimum and maximum numbers of two numbers (**int**, **long**, **float**, or **double**).

For example, **max(4.4, 5.0)** returns **5.0**, and **min(3, 2)** returns **2**.

The **abs** method returns the absolute value of the number (**int**, **long**, **float**, or **double**). (**Math.abs(−2)** returns **2**).

**The random Method**

The **random()** method generates a random double value greater than or equal to 0.0 and less than 1.0 (**0 <= Math.random() < 1.0**)**.** You can use it to write a simple expression to generate random numbers in any range.

* 1. **Character Data Type and Operations**

A character data type represents a single character.

In addition to processing numeric values, you can process characters in Java.

The character data type, **char**, is used to represent a single character. A character literal is enclosed in single quotation marks. Consider the following code:

**char letter = 'A';**

**char numChar = '4';**

The first statement assigns character **A** to the **char** variable **letter**. The second statement assigns digit character **4** to the char variable **numChar**.

**Unicode and ASCII Code**

Computers use binary numbers internally. A character is stored in a computer as a sequence of 0s and 1s. Mapping a character to its binary representation is called **encoding**.

Java supports **Unicode**, an encoding scheme established by the Unicode Consortium to support the interchange, processing, and display of written texts in the world’s diverse languages.

Unicode includes ASCII code, with **\u0000** to **\u007F** corresponding to the 128 ASCII characters. Table 4.4 shows the ASCII code for some commonly used characters. Appendix B, “The ASCII Character Set,” gives a complete list of ASCII characters and their decimal and hexadecimal codes.

**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**

Java uses a special notation to represent special characters, This special notation, called an **escape sequence**, consists of a backslash (**\**) followed by a character or a combination of digits.

**Escape Sequences:**

|  |  |  |  |
| --- | --- | --- | --- |
| Escape sequence | Name | Unicode value | 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 |

**Casting between char and Numeric Types**

A **char** can be cast into any numeric type, and vice versa.

When an integer is cast into a **char**, only its lower 16 bits of data are used; the other part is ignored.

* 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**.
* When a **char** is cast into a numeric type, the character’s Unicode is cast into the specified numeric type.
* Implicit casting can be used if the result of a casting fits into the target variable. Otherwise, explicit casting must be used.
* 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 a **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 a string is concatenated with a character, the character is converted into a string.

**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 false 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 less than the Unicode for **'8' (56).**

For convenience, Java provides the following methods in the **Character** class for testing characters. The **Character** class is defined in the **java.lang** package.

**Methods in the Character Class:**

|  |  |
| --- | --- |
| METHOD | DESCRIPRION |
| 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 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. |

* 1. **The String Type**

A string is a sequence of characters.

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

**String** is a predefined class in the Java library, just like the classes **System** and **Scanner.**

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.

The variable declared by a reference type is known as a reference variable that references an object.

String methods for obtaining string length, for accessing characters in the string, for concatenating string, for converting string to uppercases or lowercases, and for trimming a string.

**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. |

**Reading a String from the Console**

To read a string from the console, invoke the **next()** method on a **Scanner** object.

For convenience, we call the input using the methods **next(),** **nextByte(), nextShort(), nextInt(), nextLong(), nextFloat(), and nextDouble()** the token-based input, because they read individual elements separated by whitespace characters rather than an entire line.

The **nextLine()** method is called a line-based input.

**Reading a Character from the Console**

To read a character from the console, use the **nextLine()** method to read a string and then invoke the **charAt(0)** method on the string to return a character.

**Comparing Strings**

The String class contains the methods, for comparing two 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. |
| contains(s1) | Returns true if s1 is a substring in this string. |

**Note:** The **compareTo** method can also be used to compare two strings.

**Obtaining Substrings**

You can obtain a single character from

a string using the **charAt** method. You can also obtain a substring from a string using the **substring** method in the **String** class.

**The String Class Contains the Methods for Obtaining Substring**

|  |  |
| --- | --- |
| 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. |
| substring(beginIndex, endIndex) | Returns this string’s substring that begins at the specified **beginIndex** and extends to the character at index **endIndex– 1**,  Note the character at **endIndex** is not part of the substring. |

**Finding a Character or a Substring in a String**

The **String** class provides several versions of **indexOf** and **lastIndexOf** methods to find a character or a substring in a string.

|  |  |
| --- | --- |
| Method | Description |
| indexOf(ch) | Returns the index of the first occurrence o**f 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. |

**Conversion between Strings and Numbers**

You can convert a numeric string into a number.

* To convert a string into an **int** value, use the **Integer.parseInt** method, as follows:

**int intValue = Integer.parseInt(intString);**

Where **intString** is a numeric string such as "123".

* To convert a string into a double value, use the **Double.parseDouble** method, as follows:

**double doubleValue = Double.parseDouble(doubleString);**

Where **doubleString** is a numeric string such as "123.45".

* If the string is not a numeric string, the conversion would cause a runtime error. The **Integer** and **Double** classes are both included in the **java.lang** package, and thus they are automatically imported.
* You can convert a number into a string; simply use the string concatenating operator as follows:

String s = number + "";

* 1. **Formatting Console Output**

You can use the **System.out.printf** method to display formatted output on the console.

Often, it is desirable to display numbers in a certain format.

The **f** in the **printf** stands for formatted, implying that the method prints an item in some format.

The syntax to invoke this method is :

**System.out.printf(format, item1, item2, ..., itemn);**

Where **format** is a string that may consist of substrings and format specifiers.

A **format specifier** specifies how an item should be formatted. An item may be a numeric value, a character, a Boolean value, or a string. A simple format specifier consists of a percent sign (**%**) followed by a conversion code.

Here is the list of some frequently used simple 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” |

The **%** sign denotes a format specifier. To output a literal **%** in the format string, use **%%**.

**NOTE:** The items must match the format specifiers in exact type. The item for the format specifier **%f** or %e must be a floating-point type value such as **40.0**, not **40**. Thus, an **int** variable cannot match **%f** or **%e**. You can use %.2f to specify a floating-point value with two digits after the decimal point. However, **%0.2f** would be incorrect

**THE END!**