CS3231
Object Oriented Programming I

Name: _____ ( ) Date: _____

Chapter 8: Java API

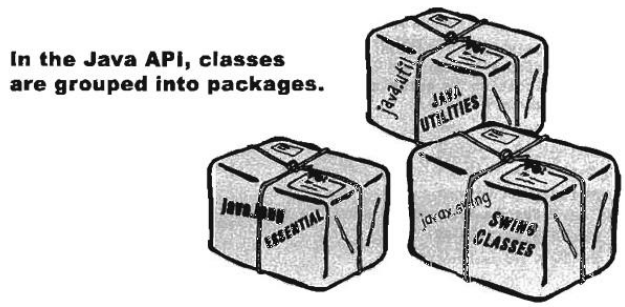
Lesson Objectives

After completing the lesson, the student will be able to

- *understand the purpose of Java Application Programming Interface (Java API)*
- *know how to use import Java API*
- *solve mathematical problems by using the methods in the Math class*
- *generate random numbers using Random Class*
- *manipulate text strings using String Class*

8.1 Java Application Programming Interface

A great strength of Java is its rich set of predefined classes that you can reuse rather than "reinventing the wheel." Java contains many predefined classes that are grouped into categories of related classes called **packages**. From JDK 9 onwards, packages are further grouped in **modules**. Every class in the Java library belongs to a package. Together, these are known as the **Java Application Programming Interface (Java API)**, or the Java class library. The package has a name like `java.lang.Math`. Related packages are then further grouped into modules.



Using a class from the API, in your own code, is simple. You just treat the class as though you wrote it yourself as though you compiled it, and there it sits waiting for you to use it. With one big difference:

Somewhere in your code you have to indicate the full name of the library class you want to use, and that means the package name with class name. Even if you did not know it, you have already been using classes from a package. `System(System.out.println)`, `String`, and `Math (Math.abs())`, all belong to the `java.lang` package.

Packages are important for three main reasons. First, they help the overall organization of a project or library. Rather than just having one horrendously large pile of classes, they are all grouped into packages for specific kinds of functionality (like GUI,or data structures, or database stuff, etc.)

Second, packages give you a namescoping, to help prevent collisions if you and 12 other programmers in your company all decide to make a class with the same name. If you have a class named Set and someone else (including the Java API) has a class named Set, you need some way to tell the JVM which Set class you're trying to use.

Third, packages provide a level of security, because you can restrict the code you write so that only other classes in the same package can access it. Table 8.1 shows a subset of commonly

use Java API packages. You are encourage to explore the rich Java API documentations on your own, refer to the following link: https://docs.oracle.com/en/java/javase/13/docs/api/index.html

Most of the important Java API packages we are discussing can be found in the java.base module, which defines the foundational APIs of the Java SE Platform.
https://docs.oracle.com/en/java/javase/13/docs/api/java.base/module-summary.html

Name	Description
java.io	The Java Input/Output Package contains classes and interfaces that enable programs to input and output data.
java.lang	The Java Language Package contains classes and interfaces that are required by many Java programs. **This package is imported by the compiler into all programs.**
java.util	The Java Utilities Package contains utility classes and interfaces that enable storing and processing of large amounts of data. Many of these classes and interfaces have been updated to support Java SE 8's new lambda capabilities.
java.time	The new Java SE 8 Date/Time API Package contains classes and interfaces for working with dates and times. These features are designed to replace the older date and time capabilities of package java.util.
javafx package	JavaFX is the preferred GUI technology for the future.

Table 8.1 Subset of Java API Packages

In each new Java version, the APIs typically contain new capabilities that fix bugs, improve performance or offer better means for accomplishing tasks. The corresponding older versions are no longer needed and should not be used. Such APIs are said to be deprecated and might be removed from later Java versions. You will often encounter deprecated APIs when browsing the online API documentation. The compiler will warn you when you compile code that uses deprecated APIs.

8.2 Using import

The Java API provides a rich collection of predefined classes that contain methods for performing common mathematical calculations, string manipulations, character manipulations, input/output operations, database operations, networking operations, file processing, error checking and more. To use these classes you need to import them into your program.

There are two types of import statements: **specific import** and **wildcard import**. The specific import specifies a single class in the `import` statement. For example, the following statement imports Scanner from the package `java.util`.

```
import java.util.Scanner;
```

The wildcard import imports all the classes in a package by using the asterisk as the wildcard. For example, the following statement imports all the classes from the package `java.util`.

```
import java.util.*;
```

The information for the classes in an imported package is not read in at compile time or runtime unless the class is used in the program. **The import statement simply tells the compiler where to locate the classes.** There is no performance difference between a specific import and a wildcard import declaration.

All import declarations must appear before the first class declaration in the file. Placing an import declaration inside or after a class declaration is a syntax error. Forgetting to include an import declaration for a class that must be imported will result in a compilation error containing a message such as "cannot find symbol." When this occurs, check that you provided the proper import declarations and that the names in them are correct, including proper capitalization.

Alternatively, you may also type the full location of the class directly in your code. For example,

```
java.util.Scanner input = new java.util.Scanner();
```

This style of coding is not efficient as it slows down the development time and it is prone to introduce bugs into the program.

All the classes in the `java.lang` package are implicitly imported in a Java program. For example, the Math and String class are used in the program, but not imported, because they are in the `java.lang` package.

8.3 Math Class

The `java.lang.Math` class contains methods for performing basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric 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. Service methods include the `rounding`, `min`, `max`, `absolute`, and `random` methods. In addition to methods, the Math class provides two useful double constants, `PI` and `E` (the base of natural logarithms). You can use these constants as `Math.PI` and `Math.E` in any program.

8.3.1 Trigonometric Methods

Name	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 8.2 Trigonometric Methods

The parameter for `sin`, `cos`, and tan is an angle in radians. The return value for `asin`, acos, and `atan` is a degree in radians in the range between - π/2 and π/2. One degree is equal to π/180 in radians, 90 degrees is equal to π/2 in radians, and 30 degrees is equal to π/6 in radians.

Example	Result
`Math.toDegrees(Math.PI / 2)`	**`90.0`**
`Math.toRadians(30)`	**`0.5236`** `(same as π/6)`
`Math.sin(Math.PI / 6)`	**`0.5`**
`Math.cos(0)`	**`1.0`**
`Math.cos(Math.PI / 6)`	**`0.866`**
`Math.asin(0.5)`	**`0.523598333`** `(same as π/6)`
`Math.acos(0.5)`	**`1.0472`** `(same as π/3)`
`Math.atan(1.0)`	**`0.785398`** `(same as π/4)`

8.3.2 Exponent Methods

Name	Description
`exp(x)`	Returns e raised to power of x (ex).
`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 $x \geq 0$.

Table 8.3 Exponent Methods

Example	Result
`Math.exp(1)`	`2.71828`
`Math.log(Math.E)`	`1.0`
`Math.log10(10)`	`1.0`
`Math.pow(2, 3)`	`8.0`
`Math.pow(3, 2)`	`9.0`
`Math.pow(4.5, 2.5)`	`22.91765`
`Math.sqrt(4)`	`2.0`
`Math.sqrt(10.5)`	`3.24`

8.3.3 Rounding Methods

Name	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.

Table 8.4 Rounding Methods

Example	Result
`Math.ceil(2.1)`	`3.0`
`Math.ceil(-2.0)`	`-2.0`
`Math.ceil(-2.1)`	`-2.0`
`Math.floor(2.1)`	`2.0`
`Math.floor(-2.0)`	`-2.0`
`Math.rint(2.1)`	`2.0`
`Math.rint(-2.0)`	`-2.0`
`Math.rint(-2.1)`	`-2.0`
`Math.rint(4.5)`	`4.0`
`Math.rint(-2.5)`	`-2.0`
`Math.round(2.6f)`	`3`
`Math.round(-2.0f)`	`-2`
`Math.round(-2.6)`	`-3`
`Math.round(-2.4)`	`-2`

8.3.4 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).

Example	Result
`Math.max(2, 3)`	`3`
`Math.max(2.5, 3)`	`3.0`
`Math.min(2.5, 4.6)`	`2.5`
`Math.abs(-2)`	`2`
`Math.abs(-2.1)`	`2.1`

8.3.5 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.

Example	Result
`(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.

8.4 Random Class

In Section 8.3.5, `Math.random()` generates a random double value between 0.0 and 1.0 (excluding 1.0). Another way to generate random numbers is to use the `java.util.Random` class, as shown in Table 8.5, which can generate a random int, long, double, float, and boolean value. The Random class of the Java library implements a random number generator that produces numbers that appear to be completely random. To generate random numbers, you construct an object of the Random class, and then apply one of the following methods: The `java.util.Random` class instance is used to generate a stream of pseudorandom numbers.

Actually, the numbers are not completely random. They are drawn from very long sequences of numbers that do not repeat for a long time. These sequences are computed from fairly simple formulas; they just behave like random numbers. For that reason, they are often called **pseudorandom numbers**.

Name	Description
`Random()`	Constructs a Random object with the current time as its seed.
`Random(seed: long)`	Constructs a Random object with a specified seed.
`nextInt(): int`	Returns a random int value.
`nextInt(n: int): int`	Returns a random int value between 0 and n (excluding n).
`nextLong(): long`	Returns a random long value.
`nextDouble(): double`	Returns a random double value between 0.0 and 1.0 (excluding 1.0).
`nextFloat(): float`	Returns a random float value between 0.0F and 1.0F (excluding 1.0F).
`nextBoolean(): boolean`	Returns a random boolean value.

Table 8.5 Random object can be used to generate random values.

When you create a Random object, you have to specify a seed or use the default seed. **A seed is a number used to initialize a random number generator. The no-arg constructor (`Random()`) creates a Random object using the current elapsed time as its seed**. **If two Random objects have the same seed, they will generate identical sequences of numbers**. For example, the following code creates two Random objects with the same seed, 3.

PROGRAM 8-1

```
1   import java.util.Random;
2   public class RandomNumber {
3    public static void main(String[] args) {
4      Random random1 = new Random(3);
5      System.out.print("From random1: ");
6      for (int i = 0; i < 10; i++)
7         System.out.print(random1.nextInt(1000) + " ");
8      Random random2 = new Random(3);
9      System.out.print("\nFrom random2: ");
10     for (int i = 0; i < 10; i++)
11        System.out.print(random2.nextInt(1000) + " "); }
12  }
```

PROGRAM OUTPUT

```
From random1: 734 660 210 581 128 202 549 564 459 961
From random2: 734 660 210 581 128 202 549 564 459 961
```

If you do not provide a seed for both line 4 and 8, instead using the default seed `Random()`. You will notice that the numbers generated will be different, such as the following:

```
From random1: 271 816 25 485 768 781 589 546 48 658
From random2: 511 31 0 922 660 675 951 779 907 256
```

To further demonstrate the use of random numbers, PROGRAM 8-2 simulates 20 rolls of a six-sided die and displays the value of each roll. Using `nextInt()` to produce random values in the range 0 – 5, as follows:

```
int face = die.nextInt(6);
```

The argument 6, called the **scaling factor**—represents the number of unique values that `nextInt()` should produce (in this case six—0, 1, 2, 3, 4 and 5). This manipulation is called scaling the range of values produced by the method `nextInt()`. A six-sided die has the numbers 1 – 6 on its faces, not 0 – 5. So we shift the range of numbers produced by adding a **shifting value** as in

```
int face = 1 + die.nextInt(6);
```

The shifting value (1) specifies the first value in the desired range of random integers. The preceding statement assigns face a random integer in the range 1 – 6.

PROGRAM 8-2

```
1   import java.util.Random;
2   public class DieSimulator {
3    public static void main(String[] args) {
4     Random die = new Random();
5     for (int counter = 1; counter <= 20; counter++){  // loop 20 times
6       int face = 1 + die.nextInt(6); // pick random integer from 1 to 6
7
8       System.out.printf("%d ", face); // display generated value
9
10      if (counter % 5 == 0) // if 5 number generated, start a new line
11        System.out.println();
      }
    }
```

PROGRAM OUTPUT

```
5 1 3 2 2
2 4 4 3 1
6 4 2 6 2
5 6 2 6 3
```

Note: The Random class produced deterministic values that could be predicted by malicious programmers. Deterministic random numbers have been the source of many software security breaches. Java Secure Random class produced non-deterministic random numbers that cannot be predicted.

Most programming languages now have library features similar to Java's Secure Random class for producing non-deterministic random numbers to help prevent such problems. For this module, we will only be focusing on Random class. If you are interested to pursue the topic on non-deterministic random numbers, refer to the following link:

https://docs.oracle.com/javase/8/docs/api/java/security/SecureRandom.html

8.5 String Class

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. For example, the following code declares message to be a string with the value "Welcome to Java".

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

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**. Here, `message` is a reference variable that references a string object with contents `Welcome to Java`.

For the time being, you need to know only how to declare a String variable, how to assign a string to the variable, and how to use the methods in the String class. Table 8.6 lists the String methods for obtaining string length, for accessing characters in the string, for concatenating strings, for converting a string to upper or lowercases, and for trimming a string

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 8.6 Some methods from the String class

Strings are objects in Java. The methods in Table 8.6 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 a static method is `ClassName.methodName(arguments)`. For example, the `pow()` method in the Math class can be invoked using `Math.pow(2, 2.5)`.
The syntax to invoke an instance method is `reference-Variable.methodName(arguments)`. A method may have many arguments or no arguments. For example, the `charAt(index)` method has one argument, but the `length()` method has no arguments.

8.5.1 The `length()` Method

You can use the `length()` method to return the number of characters in a string. PROGRAM 8-3 show the use of `length()` method.

PROGRAM 8-3

```
1  import java.util.Random;
2  public class StringLength {
3   public static void main(String[] args) {
4    String message = "Welcome to Java";
5    System.out.println("The length of " + message + " is " +
6    message.length());
7    }
8  }
```

PROGRAM OUTPUT

```
The length of Welcome to Java is 15
```

8.5.2 Getting Characters from a String

The `s.charAt(index)` method can be used to retrieve a specific character in a string s, where the index is between 0 and `s.length()-1`. For example, `message.charAt(0)` returns the character W, as shown in Figure 7.1. Note that the index for the first character in the string is 0.

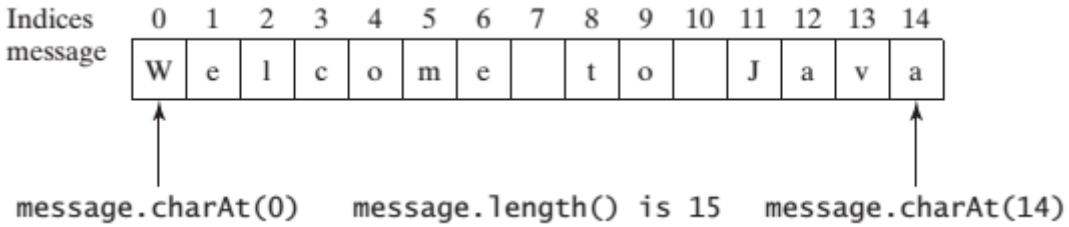


Figure 8.1 The characters in a String object can be accessed using its index.

Attempting to access characters in a string s out of bounds is a common programming error. To avoid it, make sure that you do not use an index beyond `s.length()-1`. For example, `s.charAt(s.length())` would cause a `StringIndexOutOfBoundsException`.

8.5.3 String Concatenation

You can use the `concat()` method to concatenate (join) two strings. The statement shown below, for example, concatenates strings s1 and s2 into s3:

```
String s3 = s1.concat(s2);
```

Because string concatenation is heavily used in programming, Java provides a convenient way to accomplish it. You can use the plus + operator to concatenate two strings, so the previous statement is equivalent to

```
String s3 = s1 + s2;
```

Recall that the + operator can also concatenate a number with a string. In this case, the number is converted into a string and then concatenated. Note that **at least one of the operands must be a string in order for concatenation to take place**. If one of the operands is a non-string (e.g., a number), the non-string value is converted into a string and concatenated with the other string. PROGRAM 8-4 shows some examples using the + operator.

PROGRAM 8-4

```
1  import java.util.Random;
2  public class StringConcat {
3   public static void main(String[] args) {
4      String message = "Java " + "is " + "Fun";
5      String s = "Chapter " + 7;
6      String s1 = "Grade " + 'B' ;
7   }
8  }
```

PROGRAM OUTPUT

```
Welcome to Java
Chapter 7
Grade B
```

8.5.4 Case Conversion and `trim()` Method

The `toLowerCase()` method returns a new string with all lowercase letters and the `toUpperCase()` method returns a new string with all uppercase letters. For example,

```
"Welcome".toLowerCase() returns a new string welcome.
"Welcome".toUpperCase()  returns a new string WELCOME.
```

The `trim()` method returns a new string by eliminating whitespace characters from both ends of the string. The characters `' '`, `\t`, `\f`, `\r`, or `\n` are known as whitespace characters. For example, `"\t Good Night \n".trim()` returns a new string `Good Night`.

8.5.5 Comparing Strings

The String class contains the methods as shown in Table 8.7 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.

Table 8.7 Comparison Methods for String Objects

To compare two strings you probably will use the equality == operator, as follows:

```
if (string1 == string2)
   System.out.println("string1 and string2 are the same object");
else
   System.out.println("string1 and string2 are different objects");
```

However, the **== operator checks only whether string1 and string2 refer to the same object; it does not tell you whether they have the same contents**. **Therefore, you cannot use the == operator to find out whether two string variables have the same contents**. Instead, you should use the `equals()` method. PROGRAM 8-5 demonstrate how `equals()` method is use to compare strings.

PROGRAM 8-5

```
1   import java.util.Random;
2   public class CompareString {
3    public static void main(String[] args) {
4       String s1 = "Welcome to Java";
5       String s2 = "Welcome to Java";
6       String s3 = "Welcome to C++";
7       System.out.println(s1.equals(s2));
8       System.out.println(s1.equals(s3));
9    }
10  }
```

PROGRAM OUTPUT

```
true
false
```

The `compareTo()` method can also be used to compare two strings. For example, consider the following code:

```
s1.compareTo(s2)
```

The method returns the value 0 if s1 is equal to s2, a value less than 0 if s1 is lexicographically (i.e., in terms of Unicode ordering) less than s2, and a value greater than 0 if s1 is lexicographically greater than s2. The actual value returned from the `compareTo()` method depends on the offset of the first two distinct characters in s1 and s2 from left to right.

For example, suppose s1 is `abc` and s2 is `abg`, and `s1.compareTo(s2)` returns -4. The first two characters (a vs.a) from s1 and s2 are compared. Because they are equal, the second two characters (b vs. b) are compared. Because they are also equal, the third two characters (c vs. g) are compared. Since the character c is 4 less than g, the comparison returns -4. **Syntax errors will occur if you compare strings by using relational operators >, >=, <, or <=.** Instead, you have to use `s1.compareTo(s2)`.

The `equals()` method returns true if two strings are equal and false if they are not. **The `compareTo()` method returns 0, a positive integer, or a negative integer, depending on whether one string is equal to, greater than, or less than the other string.**

The String class also provides the `equalsIgnoreCase()` and `compareToIgnoreCase()` methods for comparing strings. These methods ignore the case of the letters when comparing two strings. You can also use `str.startsWith(prefix)` to check whether string `str` starts with a specified prefix, `str.endsWith(suffix)` to check whether string `str` ends with a specified suffix, and `str.contains(s1)` to check whether string `str` contains string `s1`. PROGRAM 8-6 shows the effect of these methods.

PROGRAM 8-6

```
1   public class CompareString {
2    public static void main(String[] args) {
3      "Welcome to Java".startsWith("We")
4      "Welcome to Java".startsWith("we")
5      "Welcome to Java".endsWith("va")
6      "Welcome to Java".endsWith("v")
7      "Welcome to Java".contains("to")
8      "Welcome to Java".contains("To")
9    }
10  }
```

PROGRAM OUTPUT

```
true
false
true
false
true
false
```

8.5.6 The `substring()` Method

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, as shown in Table 8.8.

Method	Description
`substring(beginIndex)`	Returns this string's substring that begins with the character at the specified begin Index and extends to the end of the string.
`substring(beginIndex, endIndex)`	Returns this string's substring that begins at the specified begin Index and extends to the character at index end Index – 1. Note that the character at end Index is not part of the substring.

Table 8.8 The String class contains the methods for obtaining substrings.

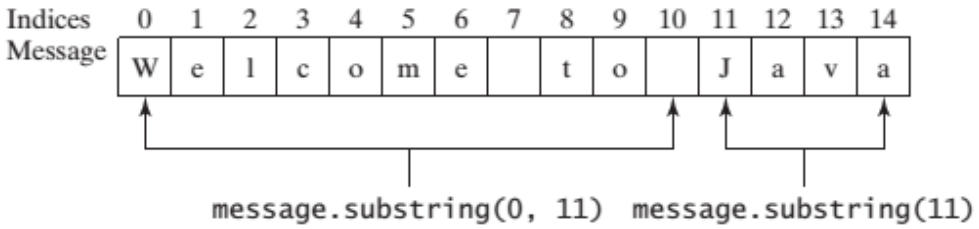


Figure 8.2 The substring method obtains a substring from a string

If `beginIndex` is `endIndex`, `substring(beginIndex, endIndex)` returns an empty string with length 0. If `beginIndex > endIndex`, it would be a runtime error.

8.5.7 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, as shown in Table 8.9.

Method	Description
`index(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 from Index 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 from Index. 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 from Index 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 from Index. Returns -1 if not matched.

Table 8.9 The String class contains the methods for finding substrings

PROGRAM 8-7 illustrates the use of `indexOf()` and `lastIndexOf()` methods.

PROGRAM 8-7

```
 1  public class indexOfDemo {
 2   public static void main(String[] args) {
 3     System.out.println("Welcome to Java".indexOf('W'));
 4     System.out.println("Welcome to Java".indexOf('o'));
 5     System.out.println("Welcome to Java".indexOf('o', 5));
 6     System.out.println("Welcome to Java".indexOf("come"));
 7     System.out.println("Welcome to Java".indexOf("Java", 5));
 8     System.out.println("Welcome to Java".indexOf("java", 5));
 9     System.out.println("Welcome to Java".lastIndexOf('W'));
10     System.out.println("Welcome to Java".lastIndexOf('o'));
11     System.out.println("Welcome to Java".lastIndexOf('o', 5));
12     System.out.println("Welcome to Java".lastIndexOf("come"));
13     System.out.println("Welcome to Java".lastIndexOf("Java", 5));
14     System.out.println("Welcome to Java".lastIndexOf("Java"));
15    }
16  }
```

PROGRAM OUTPUT

```
0
4
9
3
11
-1
0
9
4
3
-1
11
```

8.5.8 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 + ""; // number is a variable
```

Another way of converting a number into a string is to use the overloaded static `valueOf()` method. This method can also be used to convert a character or an array of characters into a string, as shown in Table 8.10.

Method	Description
`+valueOf(c: char): String`	Returns a string consisting of the character c.
`+valueOf(data: char[]): String`	Returns a string consisting of the characters in the array.
`+valueOf(d: double): String`	Returns a string representing the double value.
`+valueOf(f: float): String`	Returns a string representing the float value.
`+valueOf(i: int): String`	Returns a string representing the int value.
`+valueOf(l: long): String`	Returns a string representing the long value.
`+valueOf(b: boolean): String`	Returns a string representing the boolean value.

Table 8.10 The String class contains the static methods for creating strings from primitive type values.

For example, to convert a double value 5.44 to a string, use `String.valueOf(5.44)`. The return value is a string consisting of the characters '5' , '.' , '4' , and '4' .

8.6 Immutable Strings and Intern Strings

A String object is immutable and its contents cannot be changed. The following code does not change the contents of the string.

```
String s = "Java";
s = "HTML";
```

The first statement creates a String object with the content "Java" and assigns its reference to s. The second statement creates a new String object with the content "HTML" and assigns its reference to s. The first String object still exists after the assignment, but it can no longer be accessed, because variable s now points to the new object, as shown in Figure 8.3.

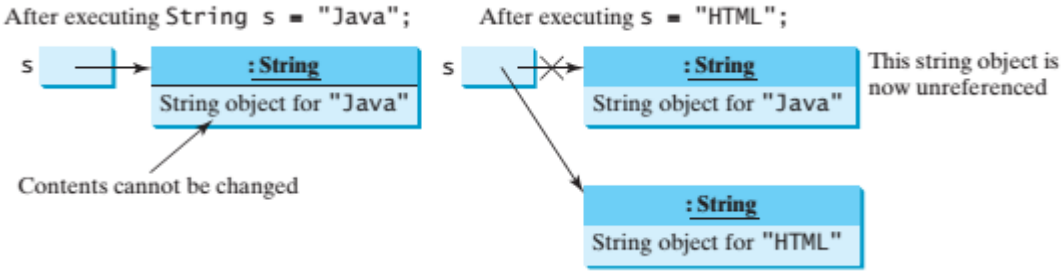
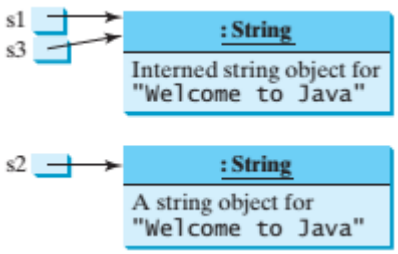


Figure 8.3 Strings are immutable; once created, their contents cannot be changed.

Because strings are immutable and are ubiquitous in programming, the JVM uses a unique instance for string literals with the same character sequence in order to improve efficiency and save memory. Such an instance is called an interned string. For example, the following statements:

```
String s1 = "Welcome to Java";
String s2 = new String("Welcome to Java");
String s3 = "Welcome to Java";
System.out.println("s1 == s2 is " + (s1 == s2));
System.out.println("s1 == s3 is " + (s1 == s3));
```



display

```
s1 == s2 is false
s1 == s3 is true
```

In the preceding statements, s1 and s3 refer to the same interned string—"Welcome to Java"—so s1 == s3 is true. However, s1 == s2 is false, because s1 and s2 are two different string objects, even though they have the same contents.

Once a string is created, its contents cannot be changed. The methods `replace()`, `replaceFirst()`, and `replaceAll()` return a new string derived from the original string (without changing the original string!). Several versions of the `replace`() methods are provided to replace a character or a substring in the string with a new character or a new substring. For example,

```
"Welcome".replace('e', 'A') returns a new string, WAlcomA.
"Welcome".replaceFirst("e", "AB") returns a new string, WABlcome.
"Welcome".replace("e", "AB") returns a new string, WABlcomAB.
"Welcome".replace("el", "AB") returns a new string, WABcome.
```

The `split()` method can be used to extract tokens from a string with the specified delimiters. For example, the following code

```
//Store string tokens in array of Strings
String[] tokens = "Java#HTML#Perl".split("#");
for (int i = 0; i < tokens.length; i++)
 System.out.print(tokens[i] + " ");
```

Displays

```
Java HTML Perl
```

8.7 New String methods in JDK 11

`repeat``(int count)`
Returns a string whose value is the concatenation of this string repeated count times.

```
String str = "abc";
String repeated = str.repeat(3);
System.out.println(repeated);
```

Output:
```
abcabcabc
```

`strip``()`
Returns a string whose value is this string, with all leading and trailing white space removed.
`stripLeading``()`
Returns a string whose value is this string, with all leading white space removed.
`stripTrailing``()`

```
String s = " hello jdk 11     ";
String stripResult = s.strip();
System.out.println(stripResult);
```

Output:
```
hello jdk 11
```

`isBlank``()`
Returns true if the string is empty or contains only white space codepoints, otherwise false.

```
String s1 = "";
System.out.println(s1.isBlank());
String s2 = "  ";
System.out.println(s2.isBlank());
String s3 = "  .  ";
System.out.println(s3.isBlank());
```

Output:
```
true
true
false
```

8.8 Character Class (Self-Reading)

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. For example, '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).

Often in the program, you need to test whether a character is a number, a letter, an uppercase letter, or a lowercase letter. As shown in Appendix B, the ASCII character set, that the Unicodes for lowercase letters are consecutive integers starting from the Unicode for 'a' , then for 'b' , 'c' , . . ., and 'z' . The same is true for the uppercase letters and for numeric characters. This property can be used to write the code to test characters. For example, the following code tests whether a character ch is an uppercase letter, a lowercase letter, or a digital character.

```
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");
```

For convenience, Java provides the following methods in the Character class for testing characters as shown in the following table.

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 digit.
`isWhitespace(ch)`	Returns true if the specified character is a whitespace.
`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.

PROGRAM 8-8

```
1  public class characterDemo {
2   public static void main(String[] args) {
3     System.out.println("isDigit('a') is " + Character.isDigit('a' ));
4     System.out.println("isLetter('a') is " + Character.isLetter('a' ));
5     System.out.println("isLowerCase('a') is " +
6                        Character.isLowerCase('a' ));
7     System.out.println("isUpperCase('a') is " +
8                        Character.isUpperCase('a' ));
9     System.out.println("toLowerCase('T') is " +
10                       Character.toLowerCase('T' ));
11    System.out.println("toUpperCase('q') is " +
12                       Character.toUpperCase('q' ));
13    }
14 }
```

PROGRAM OUTPUT

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

[Reference]

[1] Introduction to Java Programming Comprehensive Version 10th Ed, Daniel Liang, 2016.
[2] Java How To Program 10th Ed, Paul Deitel, Harvey Deitel, 2016