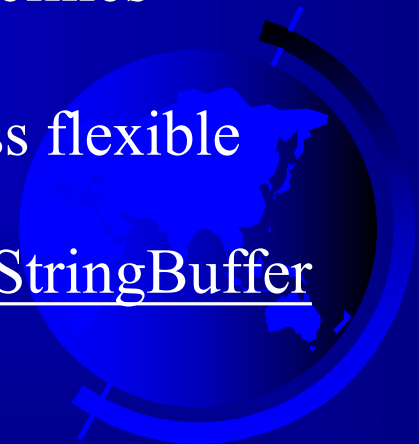


Chapter 10 Object-Oriented Thinking



Objectives

- ☞ To create immutable objects from immutable classes to protect the contents of objects
- ☞ To determine the scope of variables in the context of a class
- ☞ To use the keyword this to refer to the calling object itself
- ☞ To apply class abstraction to develop software
- ☞ To explore the differences between the procedural paradigm and object-oriented paradigm
- ☞ To develop classes for modeling composition relationships
- ☞ To design programs using the object-oriented paradigm
- ☞ To design classes that follow the class-design guidelines
- ☞ To use the String class to process fixed strings
- ☞ To use the StringBuilder/StringBuffer class to process flexible strings
- ☞ To distinguish among the String, StringBuilder, and StringBuffer classes



Object-Oriented Thinking

☞ **Procedural programming**

- data and operations on the data are separate

☞ **OO programming**

- places data and operations on the data in an object.
- organizes programs in a way that mirrors the real world, in which all objects are associated with both attributes and activities.
- Classes provide more flexibility and modularity for building reusable software. It makes programs easier to develop and easier to maintain.

☞ A Java program can be viewed as a collection of cooperating objects.



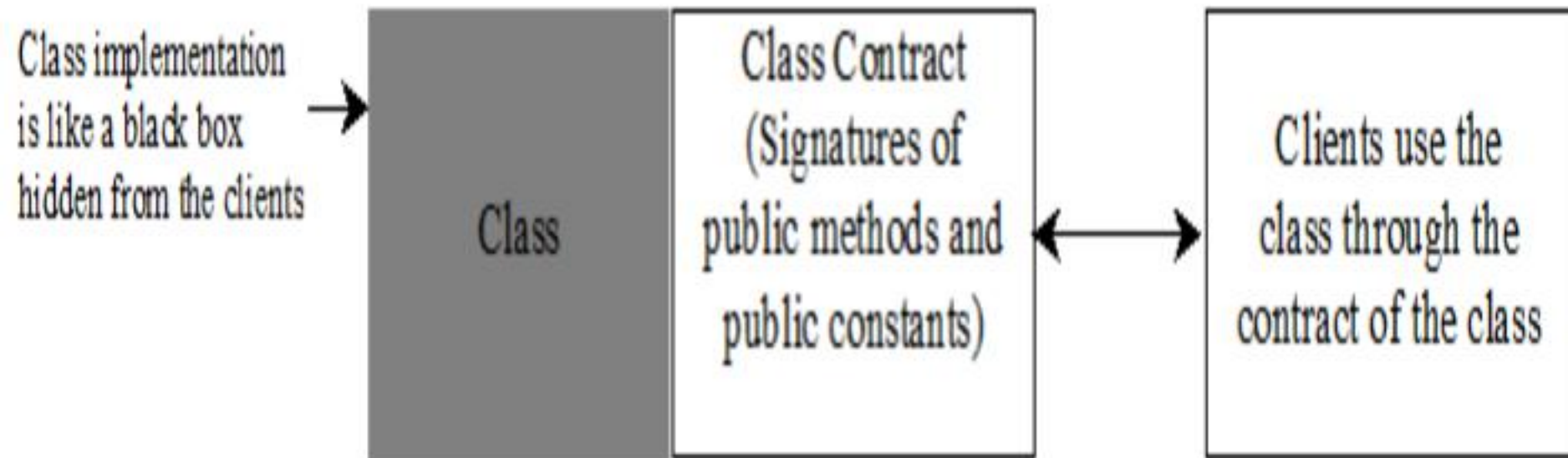
Class Abstraction and Encapsulation

Class abstraction :

separate class implementation from the use of the class.

Class Encapsulation :

- the user know how to use the class.
- but does not need to know how the class is implemented.



Example: The Course Class

Suppose you need to process course information. Each course has a name and has students enrolled. You should be able to add/drop a student to/from the course. You can use a class to model the courses, as shown in Figure 10.9.

Course

-courseName: String

-students: String[]

-numberOfStudents: int

+Course(courseName: String)

+getCourseName(): String

+addStudent(student: String): void

+dropStudent(student: String): void

+getStudents(): String[]

+getNumberOfStudents(): int

The name of the course.

An array to store the students for the course.

The number of students (default: 0).

Creates a course with the specified name.

Returns the course name.

Adds a new student to the course.

Drops a student from the course.

Returns the students for the course.

Returns the number of students for the course.

Write a test program to create two courses and adds students to them

TestCourse.java

```
1 public class TestCourse {
2     public static void main(String[] args) {
3         Course course1 = new Course("Data Structures");
4         Course course2 = new Course("Database Systems");
5
6         course1.addStudent("Peter Jones");
7         course1.addStudent("Brian Smith");
8         course1.addStudent("Anne Kennedy");
9
10        course2.addStudent("Peter Jones");
11        course2.addStudent("Steve Smith");
12
13        System.out.println("Number of students in course1: "
14            + course1.getNumberOfStudents());
15        String[] students = course1.getStudents();
16        for (int i = 0; i < course1.getNumberOfStudents(); i++)
17            System.out.print(students[i] + ", ");
18
19        System.out.println();
20        System.out.print("Number of students in course2: "
21            + course2.getNumberOfStudents());
22    }
23 }
```



```
Number of students in course1: 3
Peter Jones, Brian Smith, Anne Kennedy,
Number of students in course2: 2
```


Course class is implemented

Course.java

create students

add a course

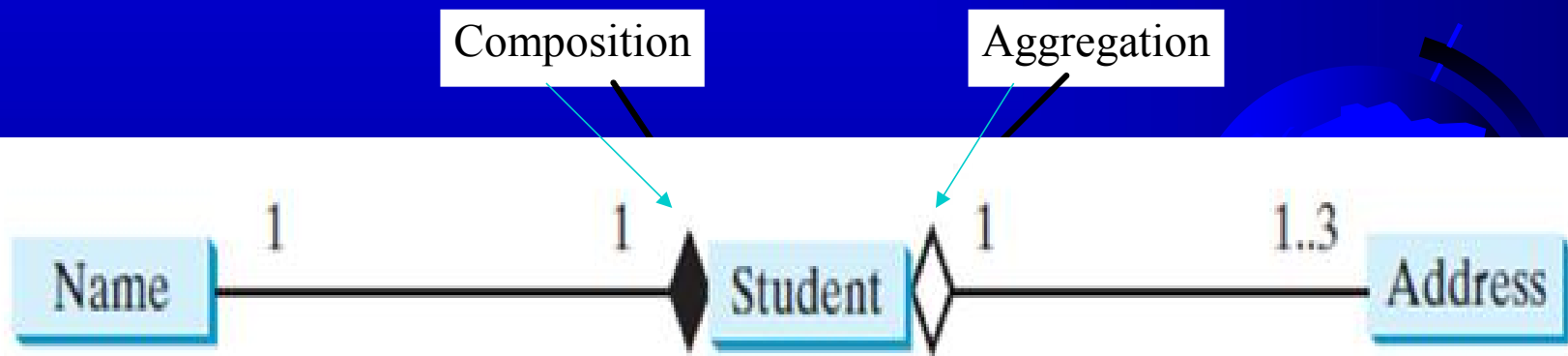
return students

```
1 public class Course {
2     private String courseName;
3     private String[] students = new String[100];
4     private int numberOfStudents;
5
6     public Course(String courseName) {
7         this.courseName = courseName;
8     }
9
10    public void addStudent(String student) {
11        students[numberOfStudents] = student;
12        numberOfStudents++;
13    }
14
15    public String[] getStudents() {
16        return students;
17    }
18
19    public int getNumberOfStudents() {
20        return numberOfStudents;
21    }
22
23    public String getCourseName() {
24        return courseName;
25    }
26
27    public void dropStudent(String student) {
28        // Left as an exercise
29    }
30 }
```

Object Composition

An object can **contain** another object

– models has-a relationships



A student has a name and an address.

An aggregation relationship is usually represented as a data field in the aggregating class.

```
public class Name {  
    ...  
}
```

Aggregated class

```
public class Student {  
    private Name name;  
    private Address address;  
  
    ...  
}
```

Aggregating class

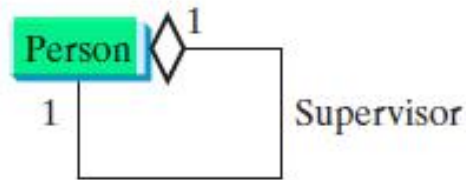
```
public class Address {  
    ...  
}
```

Aggregated class



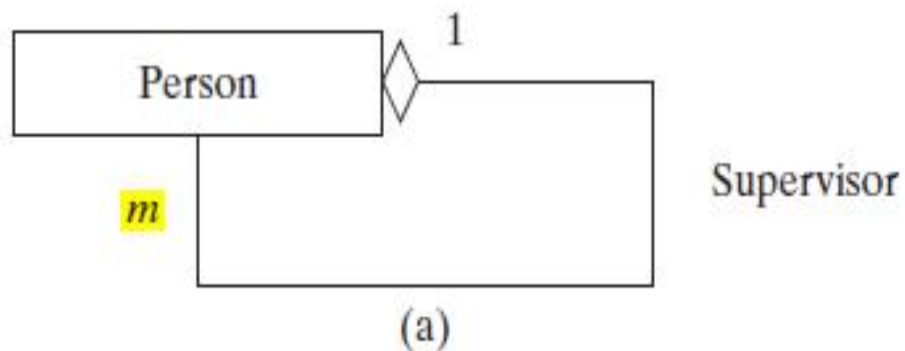
A student has a name and an address.

Aggregation may exist between objects of the same class. For example, a person may have a supervisor.



```
public class Person {  
    // The type for the data is the class itself  
    private Person supervisor;  
  
    ...  
}
```

If a person may have several supervisors

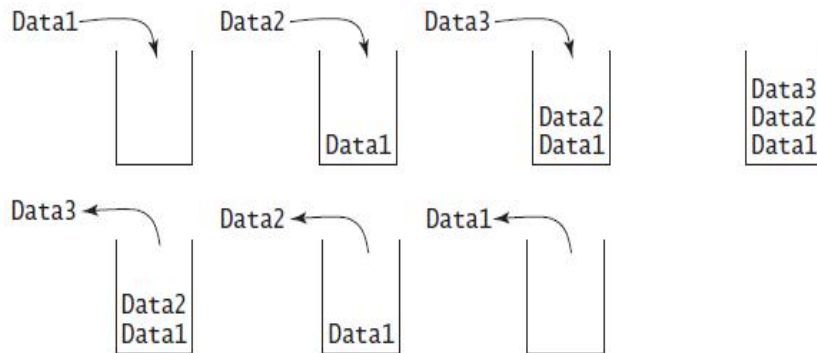


```
public class Person {  
    ...  
    private Person[] supervisors;  
}
```

(b)

Designing a Class for Stacks

Recall that a **stack** is a data structure that holds data in a **last-in, first-out** fashion, as shown in



Next, we define a class to model stacks that hold **int** values.

StackOfIntegers

-elements: int[]

-size: int

+StackOfIntegers()

+StackOfIntegers(capacity: int)

+empty(): boolean

+**peek()**: int

+push(value: int): void

+pop(): int

+**getSize()**: int

An array to store integers in the stack.

The number of integers in the stack.

Constructs an empty stack with a default capacity of 16.

Constructs an empty stack with a specified capacity.

Returns true if the stack is empty.

Returns the integer at the **top** of the stack **without removing** it from the stack.

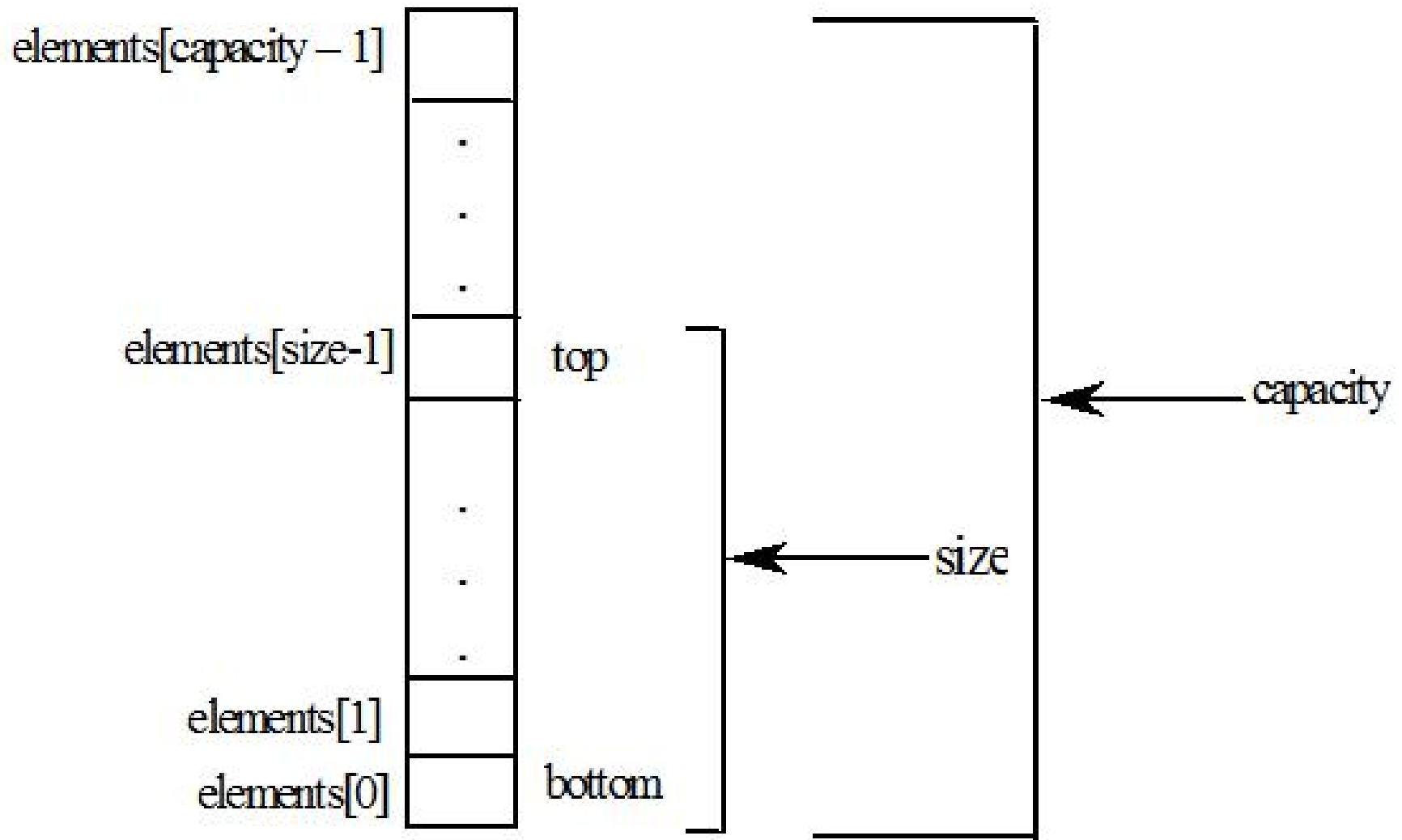
Stores an integer into the top of the stack.

Removes the integer at the top of the stack and returns it.

Returns the **number of elements in the stack**.

StackOfIntegers class encapsulates the stack storage and provides the operations for manipulating the stack.

Implementing StackOfIntegers Class



```
1 public class StackOfIntegers {
2     private int[] elements;
3     private int size;
4     public static final int DEFAULT_CAPACITY = 16;
5
6     /** Construct a stack with the default capacity 16 */
7     public StackOfIntegers() {
8         this(DEFAULT_CAPACITY);
9     }
10
11    /** Construct a stack with the specified maximum capacity */
12    public StackOfIntegers(int capacity) {
13        elements = new int[capacity];
14    }
```



```
16  /** Push a new integer into the top of the stack */
17  public void push(int value) {
18      if (size >= elements.length) {
19          int[] temp = new int[elements.length * 2];
20          System.arraycopy(elements, 0, temp, 0, elements.length);
21          elements = temp;
22      }
23
24      elements[size++] = value;
25  }
26
27  /** Return and remove the top element from the stack */
28  public int pop() {
29      return elements[--size];
30  }
31
32  /** Return the top element from the stack */
33  public int peek() {
34      return elements[size - 1];
35  }
36
37  /** Test whether the stack is empty */
38  public boolean empty() {
39      return size == 0;
40  }
41
42  /** Return the number of elements in the stack */
43  public int getSize() {
44      return size;
45  }
```

double the capacity

add to stack

Designing a Class

Coherence

- ➡ A class should describe a single entity, and all the class operations should support a coherent purpose.
- ➡ You should not combine students and staff in the same class, because students and staff have different entities.



Separating responsibilities

- ➡ A single entity with too many responsibilities can be broken into several classes to separate responsibilities.
 - E.g., the classes **String** and **StringBuilder** deal with strings, but have different responsibilities.
 - ◆ the String class deals with immutable strings
 - ◆ the StringBuilder class is for creating mutable strings
 - ◆ (last sections of this Chapter)



Consistency

Popular **styles**:

- ➡ **Place** the data declaration **before** the constructor, and place constructors before methods.
- ➡ Always provide a constructor to initialize variables
- ➡ Always provide a public no-arg constructor for constructing a default instance.



Encapsulation

- ➡ A class should use the **private** modifier to hide its data from direct access by clients.
 - Provide a **get method** only if you want the field to be readable
 - provide a **set method** only if you want the field to be updateable.
- ➡ For example, the Course class
 - provides a **get method** for **courseName**
 - but no set method, because the user is not allowed to change the course name, once it is created.



Clarity

- ☞ You should not declare a data field that can be derived from other data fields.
- ☞ For example, the following **Person** class
 - age should not be declared as a data field.

```
public class Person {  
    private java.util.Date birthDate;  
    private int age;  
  
    ...  
}
```

Using the static Modifier

- **Instance** variable/method: dependent on a specific instance
- **Static** variable/method: shared by all the instances of a class
- For example, in the class **Circle**
 - the static variable **numberOfObjects**
 - the static method **getNumberOfObjects**
 - * Shared by all the objects of the **SimpleCircle1** class , not tied to any specific instance



Which one is better?

```
public class Something {  
    private int t1;  
    private static int t2;  
  
    public Something(int t1, int t2) {  
        ...  
    }  
}
```

(a)

```
public class Something {  
    private int t1;  
    private static int t2;  
  
    public Something(int t1) {  
        ...  
    }  
  
    public static void setT2(int t2) {  
        Something.t2 = t2;  
    }  
}
```

(b)

- ☞ To **change the static data field**
 - Use a set method
 - **Do not** pass a parameter from a constructor to initialize a static data field.

Wrapper Classes

- ❑ Boolean
- ❑ Character
- ❑ Short
- ❑ Byte
- ❑ Integer
- ❑ Long
- ❑ Float
- ❑ Double

NOTE:

- (1) **Not** have no-arg constructors.
- (2) **Immutable** : object values cannot be changed once being created.



The Integer and Double Classes

java.lang.Integer

```
-value: int
+MAX_VALUE: int
+MIN_VALUE: int

+Integer(value: int)
+Integer(s: String)
+byteValue(): byte
+shortValue(): short
+intValue(): int
+longVlaue(): long
+floatValue(): float
+doubleValue():double
+compareTo(o: Integer): int
+toString(): String
+valueOf(s: String): Integer
+valueOf(s: String, radix: int): Integer
+parseInt(s: String): int
+parseInt(s: String, radix: int): int
```

java.lang.Double

```
-value: double
+MAX_VALUE: double
+MIN_VALUE: double

+Double(value: double)
+Double(s: String)
+byteValue(): byte
+shortValue(): short
+intValue(): int
+longVlaue(): long
+floatValue(): float
+doubleValue():double
+compareTo(o: Double): int
+toString(): String
+valueOf(s: String): Double
+valueOf(s: String, radix: int): Double
+parseDouble(s: String): double
+parseDouble(s: String, radix: int): double
```

The Integer Class and the Double Class

- ❑ Constructors
- ❑ Class Constants `MAX_VALUE`, `MIN_VALUE`
- ❑ Conversion Methods



Numeric Wrapper Class Constructors

To construct a wrapper object from a primitive data type value or from a string.

```
public Integer(int value)
```

```
public Integer(String s)
```

```
public Double(double value)
```

```
public Double(String s)
```



Numeric Wrapper Class Constants

MAX_VALUE

MIN_VALUE

- ☞ the max/min primitive data type
 - byte, short, int, long
 - *positive* float , double



Conversion Methods

Methods: **convert** numeric objects
into **primitive type** values

doubleValue(), floatValue(),
intValue(), longValue(), shortValue()



The Static valueOf Methods

valueOf(String s) : creates a new object initialized with the string value

```
Double doubleObject = Double.valueOf("12.4");
```

```
Integer integerObject = Integer.valueOf("12");
```



Automatic Conversion

Primitive Types \Leftrightarrow Wrapper Class Types

```
Integer[] intArray = {new Integer(2),  
                      new Integer(4),  
                      new Integer(3)};
```

can be simplified as:

```
Integer[] intArray = {2,4,3};
```

```
System.out.println(intArray[0] + intArray[1] + intArray[2]);
```



BigInteger and BigDecimal

- ➡ **BigInteger** class
 - ➡ for very large integers
- ➡ **BigDecimal** class
 - ➡ for high precision floating-point values
- ➡ Both classes:
 - in the java.math package.
 - *immutable*.
 - extend the Number class



BigInteger and BigDecimal

```
BigInteger a = new BigInteger("9223372036854775807");  
BigInteger b = new BigInteger("2");  
BigInteger c = a.multiply(b); // 9223372036854775807 * 2  
System.out.println(c);
```

LargeFactorial

```
BigDecimal a = new BigDecimal(1.0);  
BigDecimal b = new BigDecimal(3);  
BigDecimal c = a.divide(b, 20, BigDecimal.ROUND_UP);  
System.out.println(c);
```



The String Class : Construct Strings

☞ String newString = new String(stringLiteral);

- **String s = new String();**
- **String message = new String("Welcome to Java");**

☞ Java treats a string literal as a String object. So, the following statement is valid:

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

☞ create a string from an array of characters. “Good Day”:

- **char[] charArray = {'G', 'o', 'o', 'd', ' ', 'D', 'a', 'y'};**
- **String message = new String(charArray);**

☞ A String variable holds a reference to a String object that stores a string value



Strings Are Immutable

A String object is immutable; its contents cannot be changed.

☞ Does the following code change the contents of the string?

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

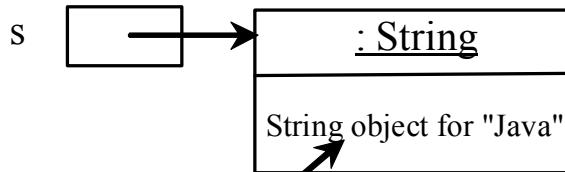


Trace Code

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

```
s = "HTML";
```

After executing `String s = "Java";`



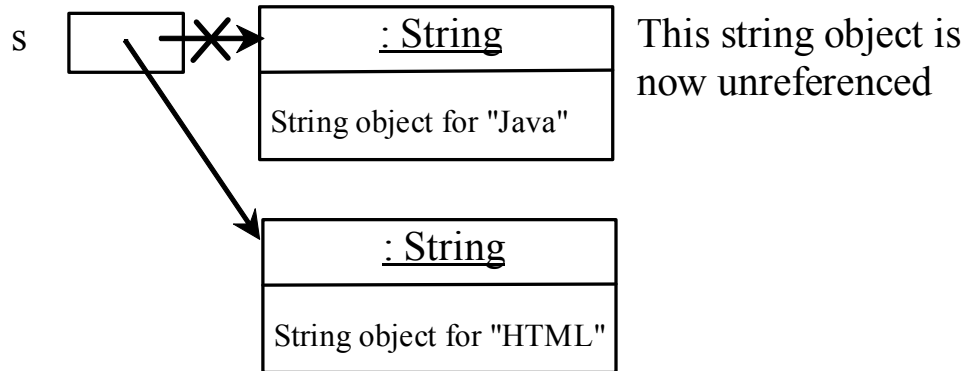
Contents cannot be changed

Trace Code

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

```
s = "HTML";
```

After executing `s = "HTML";`



Interned Strings

To improve efficiency and save memory, JVM uses a unique instance for **string literals**.

Such an instance is called *interned*.

For example, the following statements:



Examples

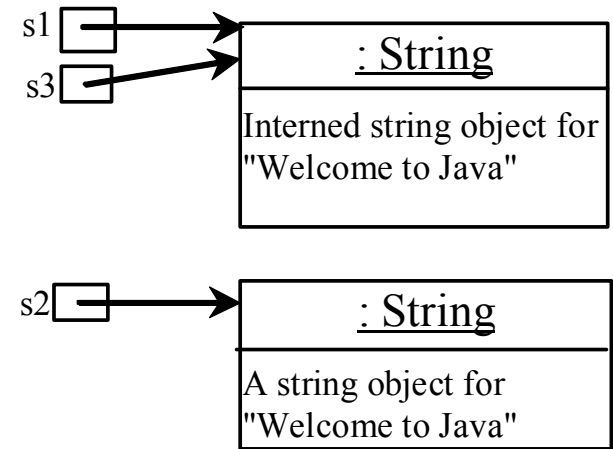
```
String s1 = "Welcome to Java";
```

```
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

If you **use the new operator**, a new object is created

If you **use the string initializer**, no new object is created if the interned object is already created.



Converting, Replacing, and Splitting Strings

java.lang.String

```
+toLowerCase(): String  
+toUpperCase(): String  
+trim(): String  
+replace(oldChar: char,  
  newChar: char): String  
+replaceFirst(oldString: String,  
  newString: String): String  
+replaceAll(oldString: String,  
  newString: String): String  
+split(delimiter: String):  
  String[]
```

Returns a new string with all characters converted to lowercase.

Returns a new string with all characters converted to uppercase.

Returns a new string with blank characters trimmed on both sides.

Returns a new string that replaces all matching characters in this string with the new character.

Returns a new string that replaces the first matching substring in this string with the new substring.

Returns a new string that replaces all matching substrings in this string with the new substring.

Returns an array of strings consisting of the substrings split by the delimiter.

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

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

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

`"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`.

Splitting a String

```
String[] tokens = "Java#HTML#Perl".split("#",  
0);  
for (int i = 0; i < tokens.length; i++)  
    System.out.print(tokens[i] + " ");  
Java HTML Perl
```

Have the same effects:

```
"Java#HTML#Perl".split("#", 0);  
"Java#HTML#Perl".split("#");
```





Java™ 2 Platform

Standard Ed. 5.0

All Classes

Packages

[java.applet](#)[java.awt](#)[java.awt.color](#)[java.awt.datatransfer](#)[java.awt.dnd](#)[StreamTokenizer](#)[StrictMath](#)[String](#)[StringBuffer](#)

sequence.

[String](#) [replaceAll](#)([String](#) regex, [String](#) replacement)Replaces each substring of this string that matches the given [regular expression](#) with the[String](#) [replaceFirst](#)([String](#) regex, [String](#) replacement)Replaces the first substring of this string that matches the given [regular expression](#) with t[String](#)[] [split](#)([String](#) regex)

Splits this string around matches of the given regular expression.

[String](#)[] [split](#)([String](#) regex, int limit)Splits this string around matches of the given [regular expression](#).boolean [startsWith](#)([String](#) prefix)

Tests if this string starts with the specified prefix.

Matching, Replacing and Splitting by Patterns

Matches method:

- At first glance, similar to the **equals** method. E.g., both evaluate to true.

```
"Java".matches("Java");
```

```
"Java".equals("Java");
```

- However, more powerful. It can match a set of strings that follow a pattern.

```
"Java is fun".matches("Java.*");
```

// the regular expression **".*"** matches **any zero or more characters**.



Matching, Replacing and Splitting by Patterns

Match/replace/split a string by specifying a **pattern**,
i.e., *regular expression*.

☞ Supplement III.F, “Regular Expressions,” for further studies.



Matching, Replacing and Splitting by Patterns

The replaceAll, replaceFirst, and split methods can be used with a regular expression.

```
String s = "a+b$#c".replaceAll("[$+#]", "NNN");  
System.out.println(s);
```

☞ \$, +, **or** #.

☞ So, the output is aNNNbNNNNNNNc.



Matching, Replacing and Splitting by Patterns

The following statement splits the string into an array of strings delimited by some punctuation marks.

```
String[] tokens = "Java,C?C#,C++".split("[.,:;?]");
```

```
for (int i = 0; i < tokens.length; i++)  
    System.out.println(tokens[i]);
```

☞ ., , , , or ?

☞ Java, C, C#, C++ : are stored into array tokens.



Finding a Character or a Substring in a String

The String class provides several overloaded methods:

java.lang.String

+indexOf(ch: char): int

Returns the index of the first occurrence of ch in the string.

Returns -1 if not matched.

+indexOf(ch: char, fromIndex: int): int

Returns the index of the first occurrence of ch after fromIndex in the string. Returns -1 if not matched.

+indexOf(s: String): int

Returns the index of the first occurrence of string s in this string.

Returns -1 if not matched.

+indexOf(s: String, fromIndex: int): int

Returns the index of the first occurrence of string s in this string after fromIndex. Returns -1 if not matched.

+lastIndexOf(ch: int): int

Returns the index of the last occurrence of ch in the string.

Returns -1 if not matched.

+lastIndexOf(ch: int, fromIndex: int): int

Returns the index of the last occurrence of ch before fromIndex in this string. Returns -1 if not matched.

+lastIndexOf(s: String): int

Returns the index of the last occurrence of string s. Returns -1 if not matched.

+lastIndexOf(s: String, fromIndex: int): int

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

indexOf

"Welcome to Java".indexOf('W') returns 0.

"Welcome to Java".indexOf('o') returns 4.

"Welcome to Java".indexOf('o', 5) returns 9.

"Welcome to Java".indexOf("come") returns 3.

"Welcome to Java".indexOf("Java", 5) returns 11.

"Welcome to Java".indexOf("java", 5) returns -1.

lastIndexOf

"Welcome to Java".lastIndexOf('W') returns 0.

"Welcome to Java".lastIndexOf('o') returns 9.

"Welcome to Java".lastIndexOf('o', 5) returns 4.

"Welcome to Java".lastIndexOf("come") returns 3.

"Welcome to Java".lastIndexOf("Java", 5) returns -1.

"Welcome to Java".lastIndexOf("Java") returns 11.

Conversion between Strings and Arrays

- ☞ convert a string to an array of characters

```
char[] chars = "Java".toCharArray();
```

- ☞ convert an array of characters to a string

- using the **String** constructor.

```
String str = new String (new char[]{'J', 'a', 'v', 'a'});
```

- using the **valueOf** method.

```
String str = String.valueOf (new char[]{'J', 'a', 'v', 'a'});
```

- ☞ copy a substring (from index **srcBegin** to index **srcEnd-1**) into a character array (**dst** starting from index **dstBegin**)

- getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)

```
char[] dst = {'J', 'A', 'V', 'A', '1', '3', '0', '1'};
```

```
"CS3720".getChars( 2, 6, dst, 4);
```

- ◆ Thus **dst** becomes {'J', 'A', 'V', 'A', '3', '7', '2', '0'}



Convert Character and Numbers to Strings

☞ String class provides overloaded static *valueOf* methods for converting a character, an array of characters, and numeric values to strings.

java.lang.String

```
+valueOf(c: char): String  
+valueOf(data: char[]): String  
+valueOf(d: double): String  
+valueOf(f: float): String  
+valueOf(i: int): String  
+valueOf(l: long): String  
+valueOf(b: boolean): String
```

Returns a string consisting of the character C.

Returns a string consisting of the characters in the array.

Returns a string representing the double value.

Returns a string representing the float value.

Returns a string representing the int value.

Returns a string representing the long value.

Returns a string representing the boolean value.

For example, String.valueOf(5.44), the return value is string “5.44”

Convert Strings to Numbers

☞ convert a string to a **double** value

Double.parseDouble(str)

☞ convert a string to an **int** value.

Integer.parseInt(str)



Formatting Strings

☞ *static method* in the **String** class to create a formatted string.

String.format(format, item1, item2, ..., itemk)

☞ For example,

String s = *String.format*("%5.2f", 45.556);

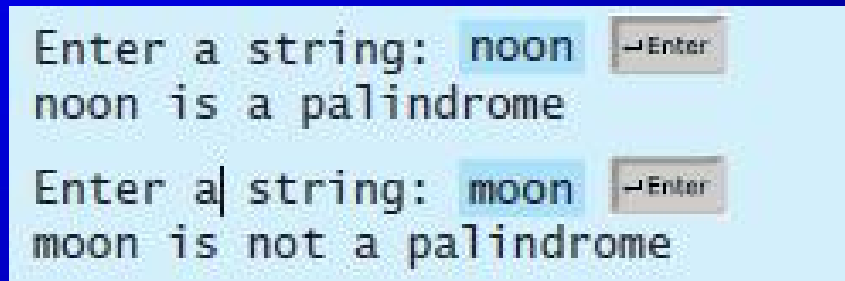
– creates a formatted string "45.56".



Problem: Finding Palindromes

☞ Checking whether a string is a **palindrome**

- a string that reads the same forward and backward. (e.g. “mom,” “dad,” and “noon,”)



```
Enter a string: noon →Enter
noon is a palindrome

Enter a string: moon →Enter
moon is not a palindrome
```

☞ Solution :

- check whether the first character in the string is the same as the last character. If so, check whether the second character is the same as the second-to-last character. This process continues until
 - ◆ a mismatch is found
 - ◆ or all characters are checked, except for the middle character if the string has an odd number of characters.

CheckPalindrome.java

```
1 import java.util.Scanner;
2
3 public class CheckPalindrome {
4     /** Main method */
5     public static void main(String[] args) {
6         // Create a Scanner
7         Scanner input = new Scanner(System.in);
8
9         // Prompt the user to enter a string
10        System.out.print("Enter a string: ");
11        String s = input.nextLine();
12
13        if (isPalindrome(s))
14            System.out.println(s + " is a palindrome");
15        else
16            System.out.println(s + " is not a palindrome");
17    }
18
19    /** Check if a string is a palindrome */
20    public static boolean isPalindrome(String s) {
21        // The index of the first character in the string
22        int low = 0;
23
24        // The index of the last character in the string
25        int high = s.length() - 1;
26
27        while (low < high) {
28            if (s.charAt(low) != s.charAt(high))
29                return false; // Not a palindrome
30
31            low++;
32            high--;
33        }
34
35        return true; // The string is a palindrome
36    }
37 }
```

input string

low index

high index

update indices



StringBuilder and StringBuffer

☞ The StringBuilder/StringBuffer class

- an alternative to the String class.
 - ◆ can be used wherever a string is used.
- more flexible than String.
 - ◆ You can add, insert, or append new contents into a string buffer
 - ◆ However, the value of a String object is fixed once the string is created.

☞ If a string does not require any change, use String rather than StringBuilder.



Three StringBuilder Constructors

java.lang.StringBuilder

```
+StringBuilder()  
+StringBuilder(capacity: int)  
+StringBuilder(s: String)
```

Constructs an empty string builder with capacity 16.
Constructs a string builder with the specified capacity.
Constructs a string builder with the specified string.



Modifying Strings in the Builder

java.lang.StringBuilder

```
+append(data: char[]): StringBuilder
+append(data: char[], offset: int, len: int):
  StringBuilder
+append(v: aPrimitiveType): StringBuilder

+append(s: String): StringBuilder
+delete(startIndex: int, endIndex: int):
  StringBuilder
+deleteCharAt(index: int): StringBuilder
+insert(index: int, data: char[], offset: int,
  len: int): StringBuilder
+insert(offset: int, data: char[]):
  StringBuilder
+insert(offset: int, b: aPrimitiveType):
  StringBuilder
+insert(offset: int, s: String): StringBuilder
+replace(startIndex: int, endIndex: int, s:
  String): StringBuilder
+reverse(): StringBuilder
+setCharAt(index: int, ch: char): void
```

Appends a char array into this string builder.

Appends a subarray in data into this string builder.

Appends a primitive type value as a string to this builder.

Appends a string to this string builder.

Deletes characters from startIndex to endIndex-1.

Deletes a character at the specified index.

Inserts a subarray of the data in the array to the builder at the specified index.

Inserts data into this builder at the position offset.

Inserts a value converted to a string into this builder.

Inserts a string into this builder at the position offset.

Replaces the characters in this builder from startIndex to endIndex-1 with the specified string.

Reverses the characters in the builder.

Sets a new character at the specified index in this builder.

Examples

```
StringBuilder stringBuilder = new StringBuilder();  
stringBuilder.append("Welcome");  
stringBuilder.append(' ');  
stringBuilder.append("to");  
stringBuilder.append(' ');  
stringBuilder.append("Java");  
– form a new string "Welcome to Java"
```

<code>stringBuilder.delete(8, 11)</code> changes the builder to <code>Welcome Java</code> .	<code>delete</code>
<code>stringBuilder.deleteCharAt(8)</code> changes the builder to <code>Welcome o Java</code> .	<code>deleteCharAt</code>
<code>stringBuilder.reverse()</code> changes the builder to <code>avaJ ot emocleW</code> .	<code>reverse</code>
<code>stringBuilder.replace(11, 15, "HTML")</code> changes the builder to <code>Welcome to HTML</code> .	<code>replace</code>
<code>stringBuilder.setCharAt(0, 'w')</code> sets the builder to <code>welcome to Java</code> .	<code>setCharAt</code>

capacity, length methods

java.lang.StringBuilder

```
+toString(): String  
+capacity(): int  
+charAt(index: int): char  
+length(): int  
+setLength(newLength: int): void  
+substring(startIndex: int): String  
+substring(startIndex: int, endIndex: int):  
  String  
+trimToSize(): void
```

Returns a string object from the string builder.

Returns the capacity of this string builder.

Returns the character at the specified index.

Returns the number of characters in this builder.

Sets a new length in this builder.

Returns a substring starting at `startIndex`.

Returns a substring from `startIndex` to `endIndex-1`.

Reduces the storage size used for the string builder.

The **StringBuilder** class contains the methods for modifying string builders.

- ➡ **Capacity:** current size of the **builder**
 - without having to increase its size.
- ➡ **Length:** actual size of the **string** stored in the builder
 - The length of the string is always less than or equal to the capacity of the builder.
 - The builder's capacity is automatically increased if more characters are added to exceed its capacity.
 - If the capacity is too large, you will waste memory space. You can use the trimToSize() method to reduce the capacity to the actual size.



Problem: Checking Palindromes

Ignoring Non-alphanumeric Characters

Counts the number of occurrence of each letter in a string.

- Assume the letters are not case-sensitive.
- Alphanumeric: letters+numbers

☞ Solution:

- 1. **Filter the string by removing the nonalphanumeric characters.** This can be done by creating an empty string builder, adding each alphanumeric character in the string to a string builder, and returning the string from the string builder.
- 2. **isLetterOrDigit(ch)** method in the **Character** class to check whether character **ch** is a letter or a digit.
- 3. Obtain a new string that is the **reversal of the filtered string.** Compare the reversed string with the filtered string using the **equals** method.

```
Enter a string: ab<c>cb?a   
Ignoring nonalphanumeric characters,  
is ab<c>cb?a a palindrome? true
```

```
Enter a string: abcc><?cab   
Ignoring nonalphanumeric characters,  
is abcc><?cab a palindrome? false
```



```

1 import java.util.Scanner;
2
3 public class PalindromeIgnoreNonAlphanumeric {
4     /** Main method */
5     public static void main(String[] args) {
6         // Create a Scanner
7         Scanner input = new Scanner(System.in);
8
9         // Prompt the user to enter a string
10        System.out.print("Enter a string: ");
11        String s = input.nextLine();
12
13        // Display result
14        System.out.println("Ignoring nonalphanumeric characters, \nis "
15            + s + " a palindrome? " + isPalindrome(s));
16    }
17
18    /** Return true if a string is a palindrome */
19    public static boolean isPalindrome(String s) {
20        // Create a new string by eliminating nonalphanumeric chars
21        String s1 = filter(s);
22
23        // Create a new string that is the reversal of s1
24        String s2 = reverse(s1);
25
26        // Compare if the reversal is the same as the original string
27        return s2.equals(s1);
28    }
29
30    /** Create a new string by eliminating nonalphanumeric chars */
31    public static String filter(String s) {
32        // Create a string builder
33        StringBuilder stringBuilder = new StringBuilder();
34
35        // Examine each char in the string to skip alphanumeric char
36        for (int i = 0; i < s.length(); i++) {
37            if (Character.isLetterOrDigit(s.charAt(i))) {
38                stringBuilder.append(s.charAt(i));
39            }
40        }
41
42        // Return a new filtered string
43        return stringBuilder.toString();
44    }
45
46    /** Create a new string by reversing a specified string */
47    public static String reverse(String s) {
48        StringBuilder stringBuilder = new StringBuilder(s);
49        stringBuilder.reverse(); // Invoke reverse in StringBuilder
50        return stringBuilder.toString();
51    }
52 }

```

check palindrome

add letter or digit