

Object-Oriented Programming

Abstract Data Type

- An abstraction of something is a simplified view of it—the abstraction “hides” the unnecessary details and allows us to focus only on the parts of interest to us.
- For example, a chart in a newspaper might be an abstraction of how world money markets are behaving.
- Abstract data types (ADTs) are an important form of program abstraction.
- An ADT consists of some hidden or protected data and a set of methods to perform actions on the data.
- When we hide data in a class, we say that the data have been encapsulated.

Abstract Data Type

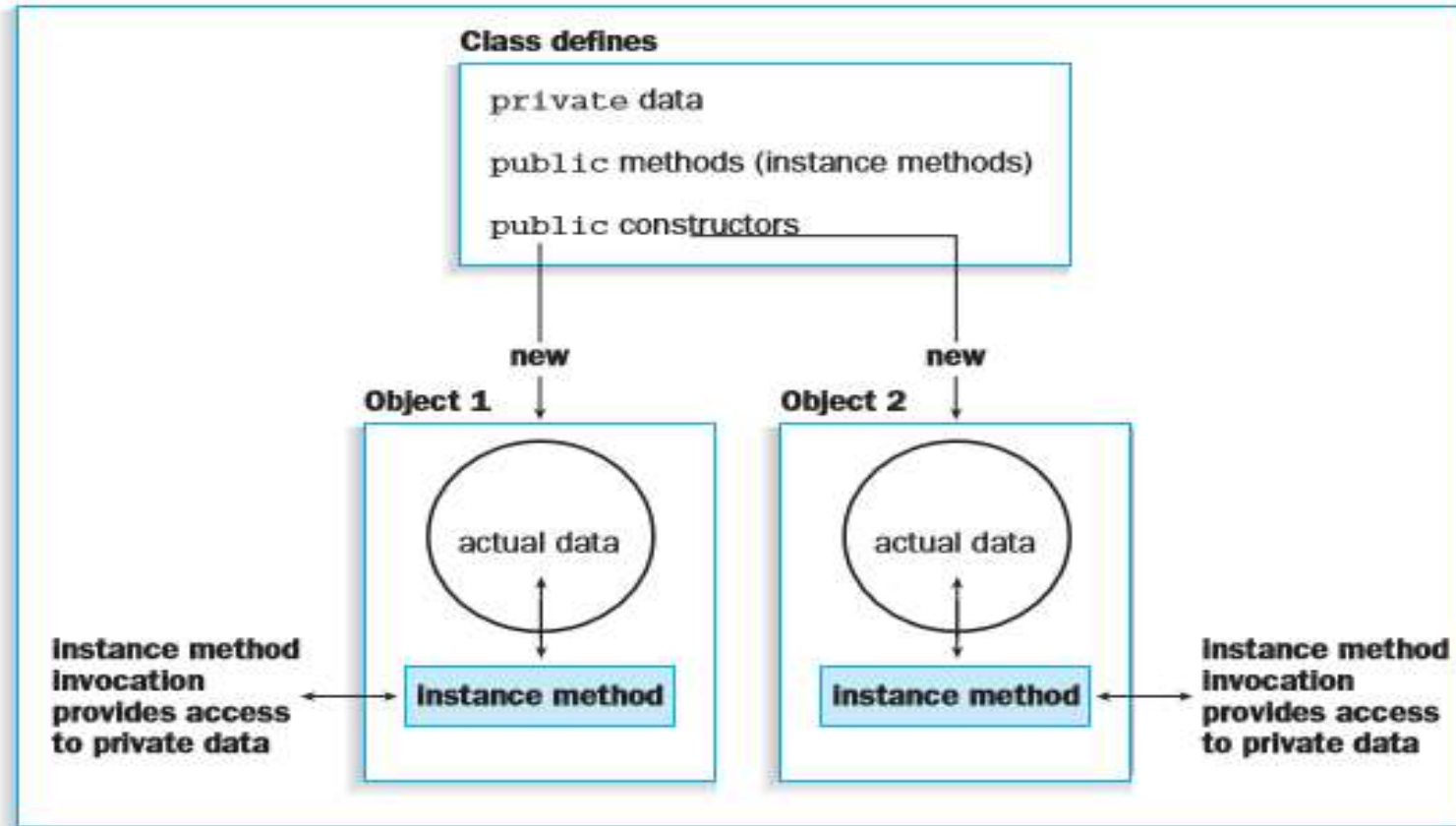


Figure 3.1 ADTs encapsulation data and methods

Abstract Data Type

- Encapsulation is illustrated in Figure 3.1.
- The figure shows a class that defines private data, public methods, and public constructors.
- It shows that objects can be instantiated from the class with the new operator.
- Finally, it shows that the private data is “hidden” inside the objects, as indicated by the heavy circle surrounding the data, and that it can be accessed only through the public instance methods.
- The implementation of the data, constructors, and methods is normally hidden from a programmer who uses the class.
- The class acts as a boundary surrounding the constructor, methods and data.

ADT

- The concept of an abstract data type is not new. Consider for a moment the class String.
- A variable of type String can be instantiated, using the String constructors, to reference a String object.
- this object can invoke many predefined instance methods such as length, concat, replace, toUpperCase, and so on.
- The method of implementing the data type String is hidden from the programmer.
- Without consulting with the author of the class String, there is no way of knowing the internal format of a String.
- We can of course guess that it might be stored as an array of characters!

ADT

- In the construction of an abstract data type, the data should be kept private to prevent access and hence changes to the values from outside of the class.
- The constructors and instance methods that are to be accessed from outside the class should be defined as public.

Constructors

- The instantiation of an object is understood to be the **allocation of memory** for storing the object's data and the **initialization** of this memory space with appropriate values.
- Instantiation is made possible by the use of a constructor, which serves several purposes.
 1. A constructor is given the **same name as the class** to allow for the data type of objects to be declared.
 2. A constructor is normally used in conjunction with the **keyword new**, which allocates memory space from the **heap**. The heap is an area of memory set aside for the dynamic allocation of computer memory to objects during run time.
 3. A constructor provides the **storage** in memory and the **initialization** of the instance variables allocated to the object.
 4. For **each separate invocation** of the constructor, a **new object** will become instantiated.

Syntax of a Constructor

The syntax of a constructor follows.

SYNTAX

Constructor:

```
public class-name ( formal-parameter-list )  
{  
    declarations  
    statements  
}
```

A constructor must be defined as being **public**, otherwise there is no means of using the constructor from outside of its class.

The name of the constructor is always the same as its class name.

Data values that specify a particular object are passed to the constructor via the formal parameter list.

It is these values that are used to initialize the instance variables of the class.

Instance Methods

SYNTAX

Method:

```
modifier(s) return-type method-name ( formal-parameter-list )  
{  
    declarations  
    statements  
}
```

Declarations refer to local constant and variable declarations for use within the method, and **statements** refer to the executable instructions within the method.

It is **possible to apply other modifiers** to a method.

Remember a method described as public means that it can be accessed from **outside** of the class.

The **return-type** identifies the type of value that the method will return. This can be a **primitive type or a class**.

If no data is returned by the method, the keyword **void** is used for the return-type.

The formal-parameter-list indicates the data types for any arguments the function expects to receive from the caller.

Each individual argument passed to the method must have its **own corresponding parameter**.

If no arguments are being passed to the method, then the **parentheses remain empty**.

Instance Methods

SYNTAX

Return Statement:

```
return expression;
```

The return-type identifies the **type of the value** that the method will return with its return statement.

where the expression may be omitted depending upon the use of the statement.

// The creation of the SwimmingPool class

```
public class SwimmingPool  
{
```

// constant private

```
final float CAPACITY_CUBIC_FOOT = 7.48f;
```

//instance variables

```
private float lengthOfPool;  
private float widthOfPool;  
private float shallowDepthOfPool;  
private float deepDepthOfPool;  
private float volume;  
private float capacity;
```

//constructor

```
public SwimmingPool(float length,float width,float shallowEndDepth,float deepEndDepth)  
{  
lengthOfPool = length;  
widthOfPool = width;  
shallowDepthOfPool = shallowEndDepth;  
deepDepthOfPool = deepEndDepth;  
}
```

// instance methods

```
public float volumeOfWater()
{
    volume = 0.5f*(lengthOfPool*widthOfPool)* (shallowDepthOfPool+deepDepthOfPool);
    return volume;
}

public float capacityOfPool()
{
    capacity = volume * CAPACITY_CUBIC_FOOT; return capacity;
}

public float timeToFillPool(float rateOfFlow)
{
    return (capacity / rateOfFlow)/60.0f;
}
}
```

- It would be illogical to attempt to execute this class on the computer.
- Do you know why? What we have achieved is to create an abstract data type of a `SwimmingPool`.
- In other words, we can create objects of this type and send various messages to these objects.
- However, we have not created a complete pro-gram. To do that we need a separate class that contains a main method.

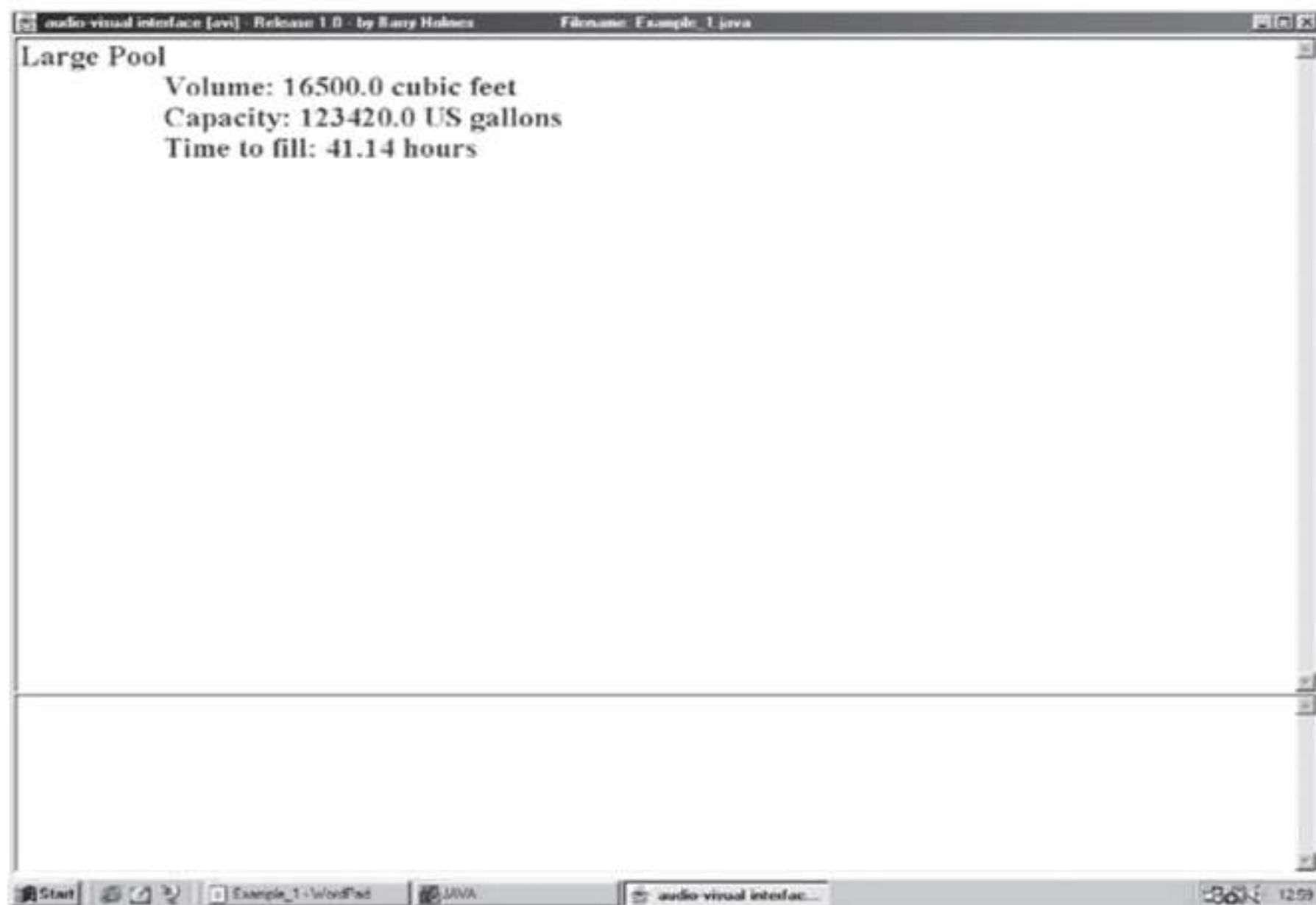
program to demonstrate using the SwimmingPool class

```
import avi.*;
class Example_1
{
public static void main(String[] args)
{
final float RATE_OF_FLOW = 50.0f;
float volume, capacity, time;

Window screen = new Window("Example_1.java","bold","blue",24);
screen.showWindow();
SwimmingPool largePool = new SwimmingPool(100.0f,30.0f,3.0f,8.0f);

volume = largePool.volumeOfWater();
capacity = largePool.capacityOfPool();
time = largePool.timeToFillPool(RATE_OF_FLOW);

screen.write("Large Pool\n\tVolume: "+volume+" cubic feet\n");
screen.write("\tCapacity: "+capacity+" US gallons\n");
screen.write("\tTime to fill: "+time+" hours\n\n");
}
}
```



Creation of Triangle Class

```
class Triangle
{
    private int l,b;
    private double res;
    public Triangle(int length,int breadth)
    {
        l=length;
        b= breadth;
    }
    public double area_triangle()
    {
        res = (0.5)*l*b;
        return res;
    }
}
```


Program to demonstrate the use of Triangle class

```
import avi.*;
class Test_Area
{
    public static void main(String args[])
    {
        double res;
        Window screen = new Window("area","bold","red",56);
        screen.showWindow();
        Triangle t=new Triangle(12,2);

        res=t.area_triangle();

        screen.write("\n\n\n area of a triangle\n\n\n");
        screen.write(res);
    }
}
```

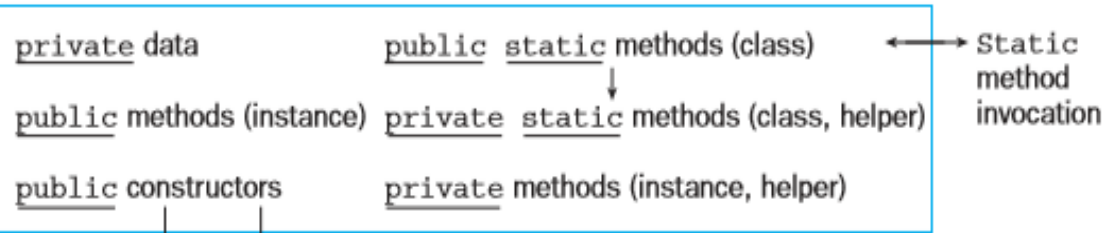
Class Methods

- A class method, also known as a static method, is one that cannot be invoked through an object.
- To differentiate an instance method from a class method, one of the modifiers used in the signature of the class method is declared as static.
- Class methods should be defined as static and public
 1. if they are to be accessed from outside of the class;
 2. otherwise, they should be labeled as static and private.

Class Methods

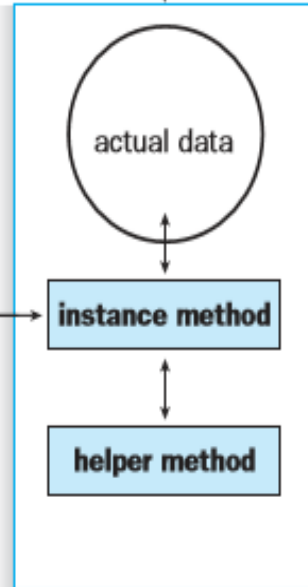
- we have identified two ways of classifying methods:
 1. A method defined as static is called a class method;
 2. a method not defined as static is called an instance method.
- A private method other than the main method is often called a helper method

Class defines



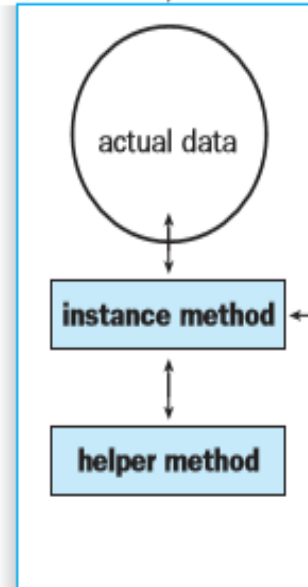
new

Object 1



new

Object 2



The figure emphasizes that **public class methods** are invoked through the **class** itself.

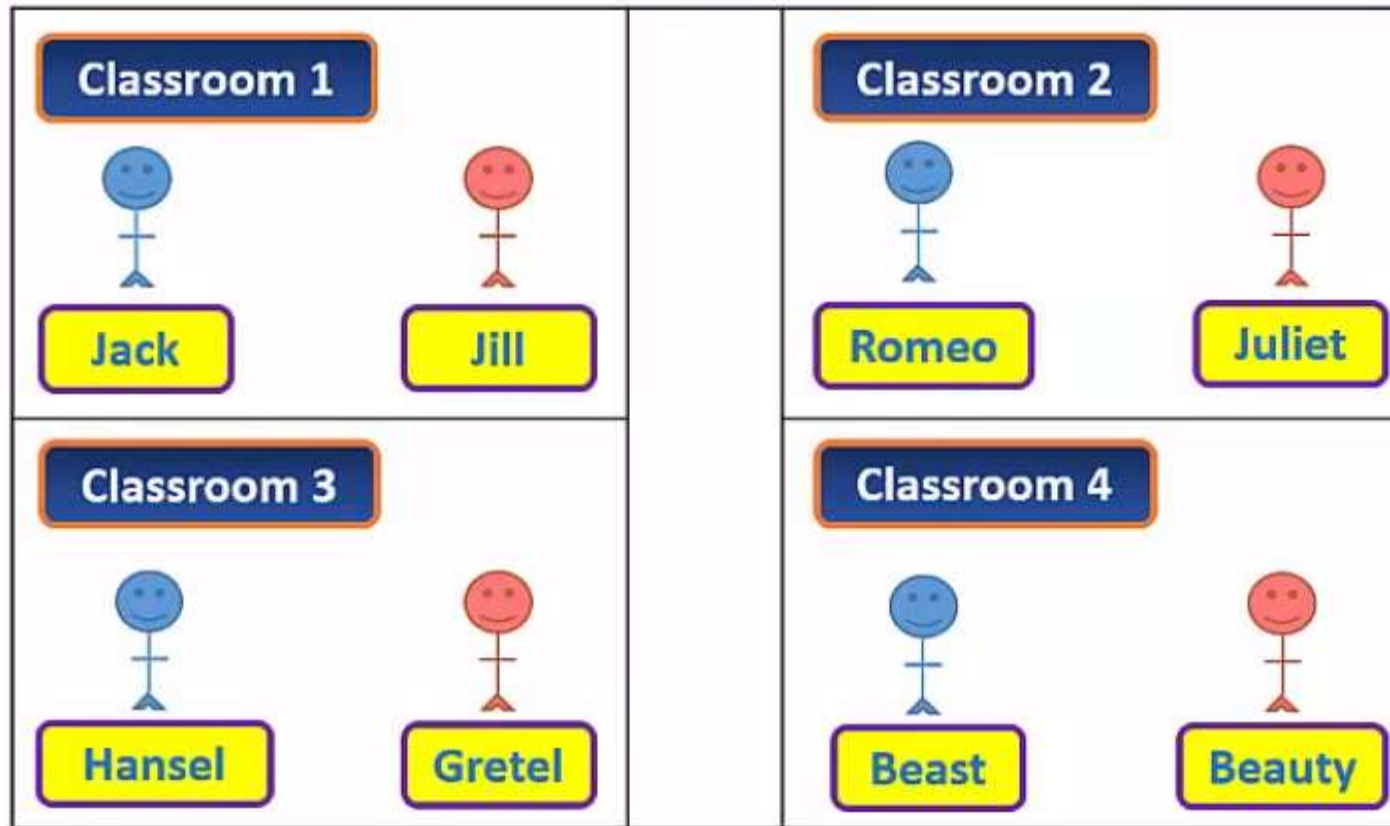
instance methods are invoked through an instance of the class, i.e., through an **object**.

helper methods can be invoked only from **other methods** defined within the class;

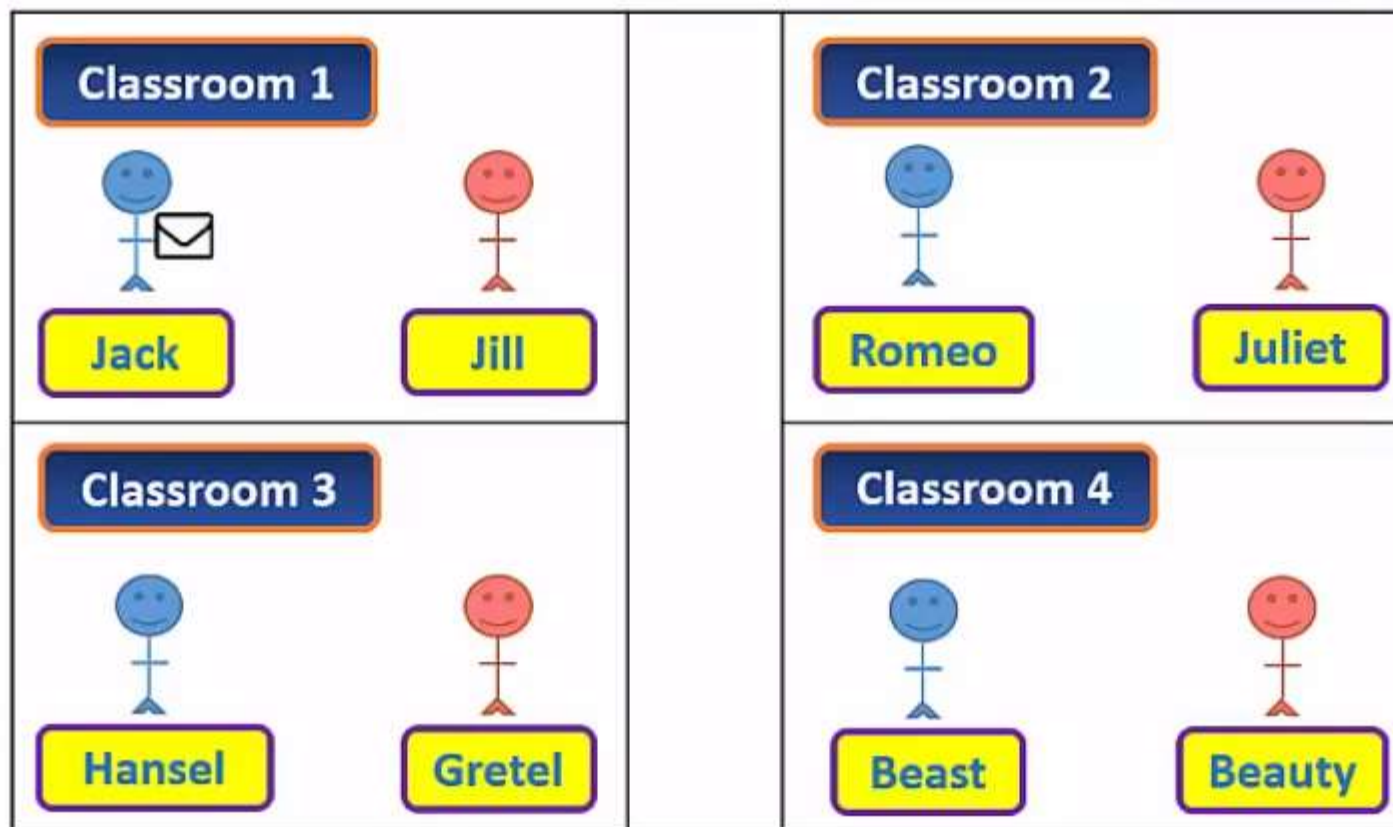
Helper methods cannot be invoked from outside the class like public methods

Figure 3.2 A class may define constructors, instance, class, and helper methods

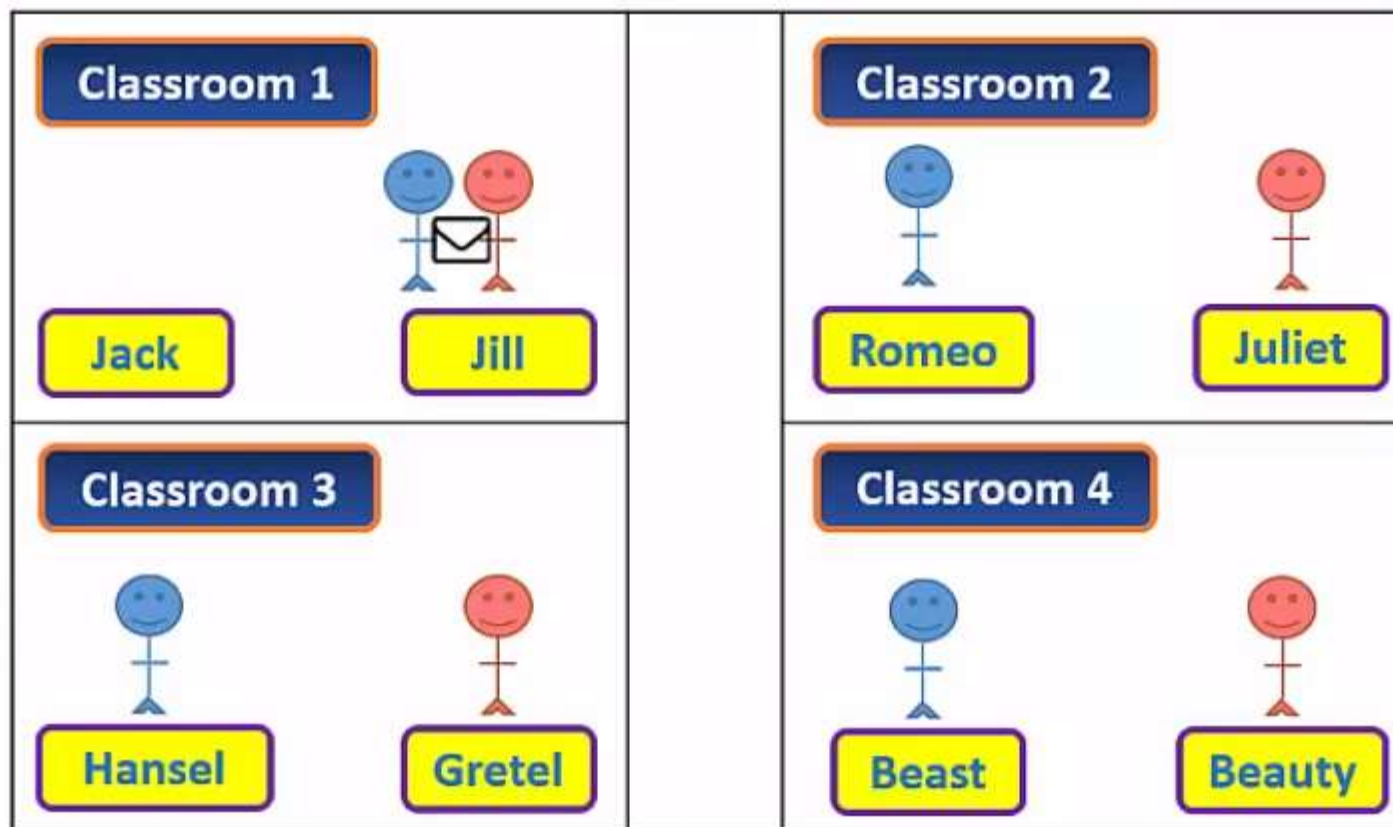
CLASSES: HELPER METHODS



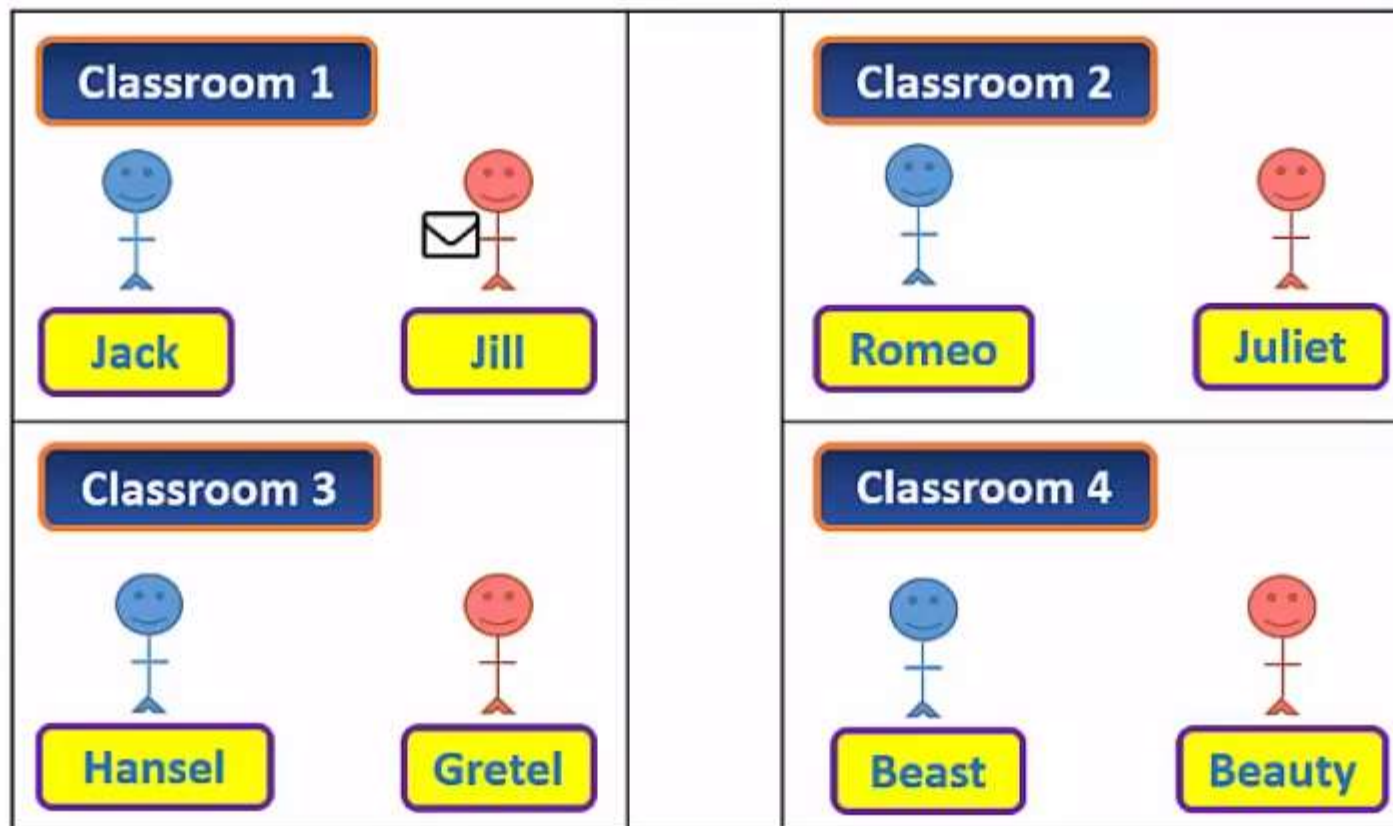
CLASSES: HELPER METHODS



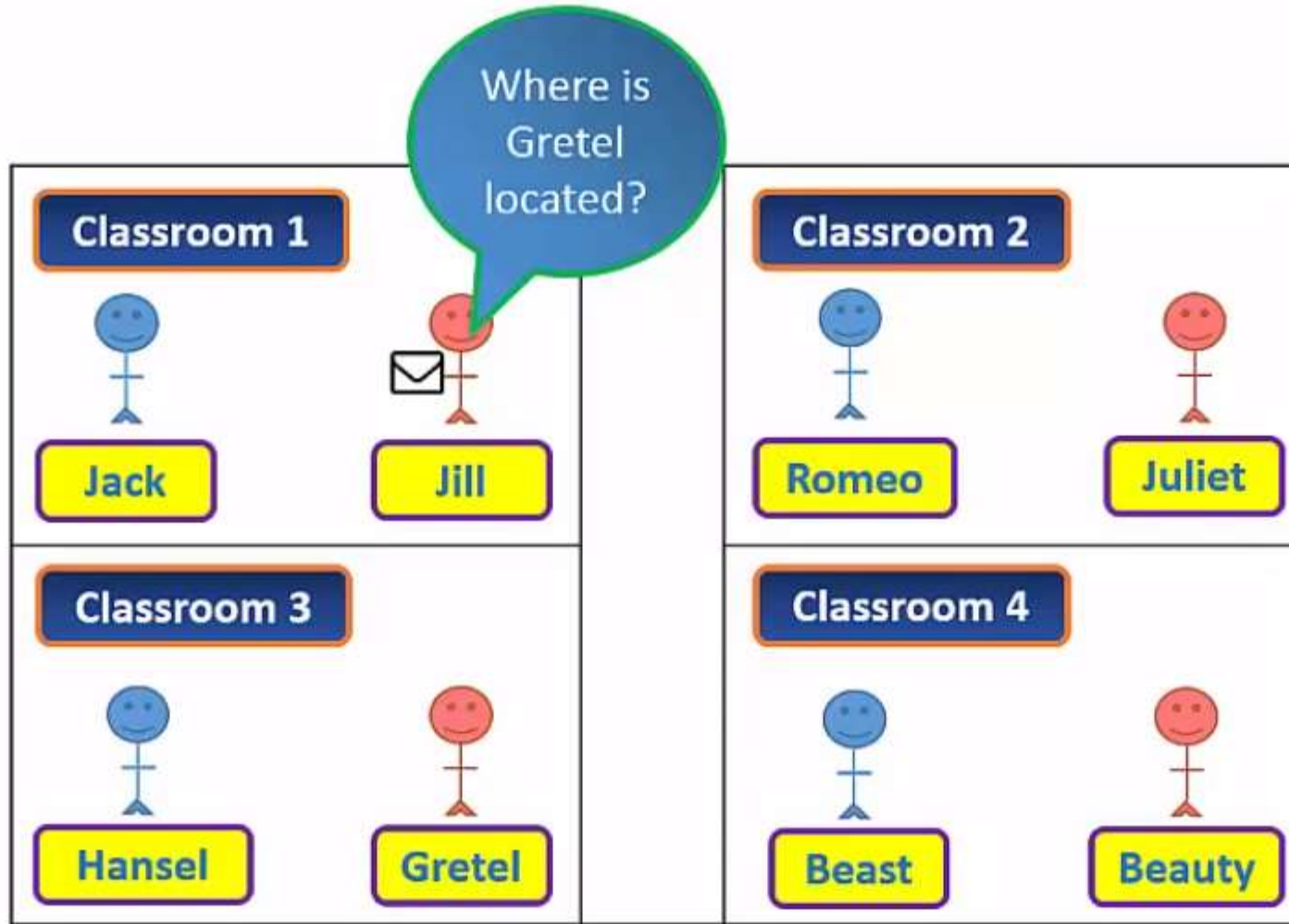
CLASSES: HELPER METHODS



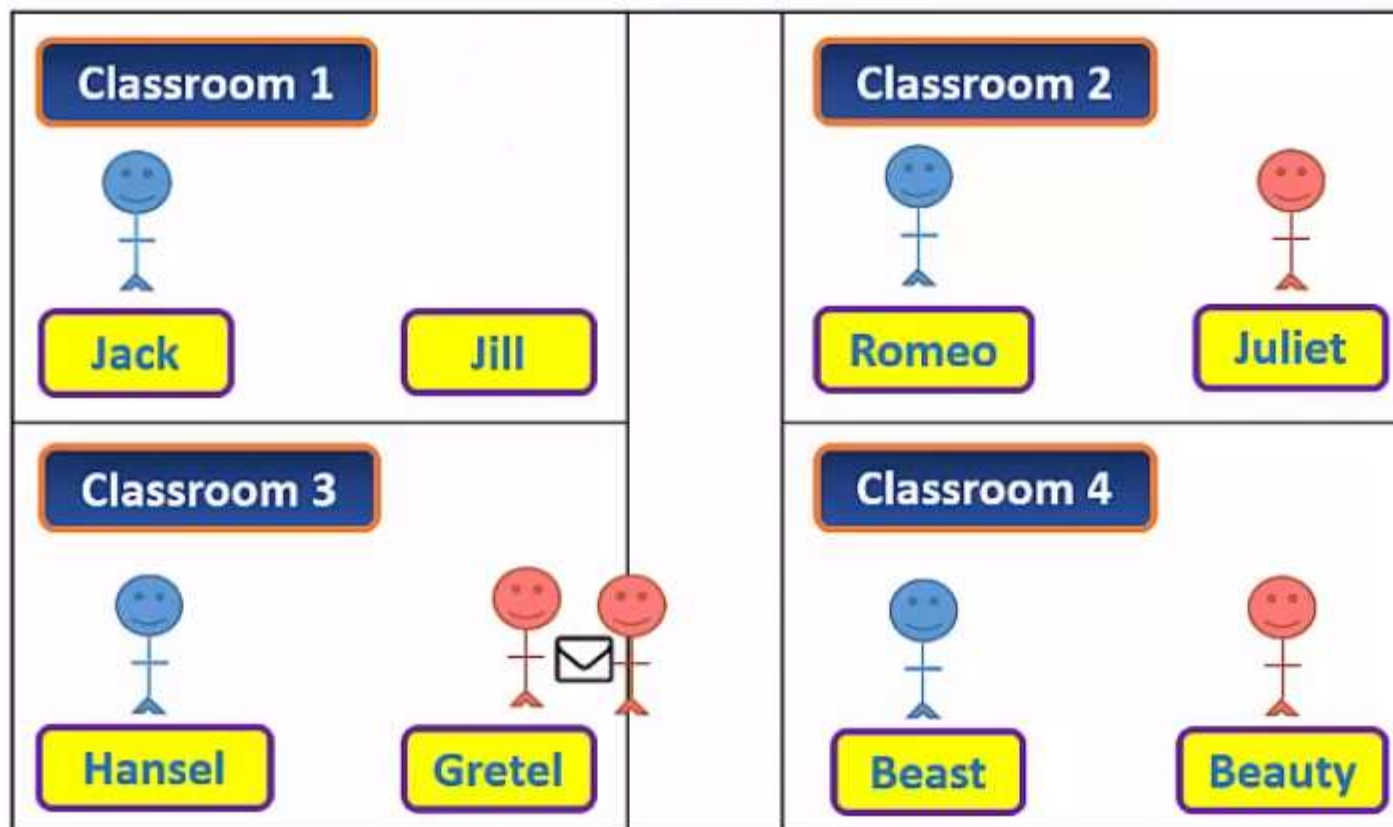
CLASSES: HELPER METHODS



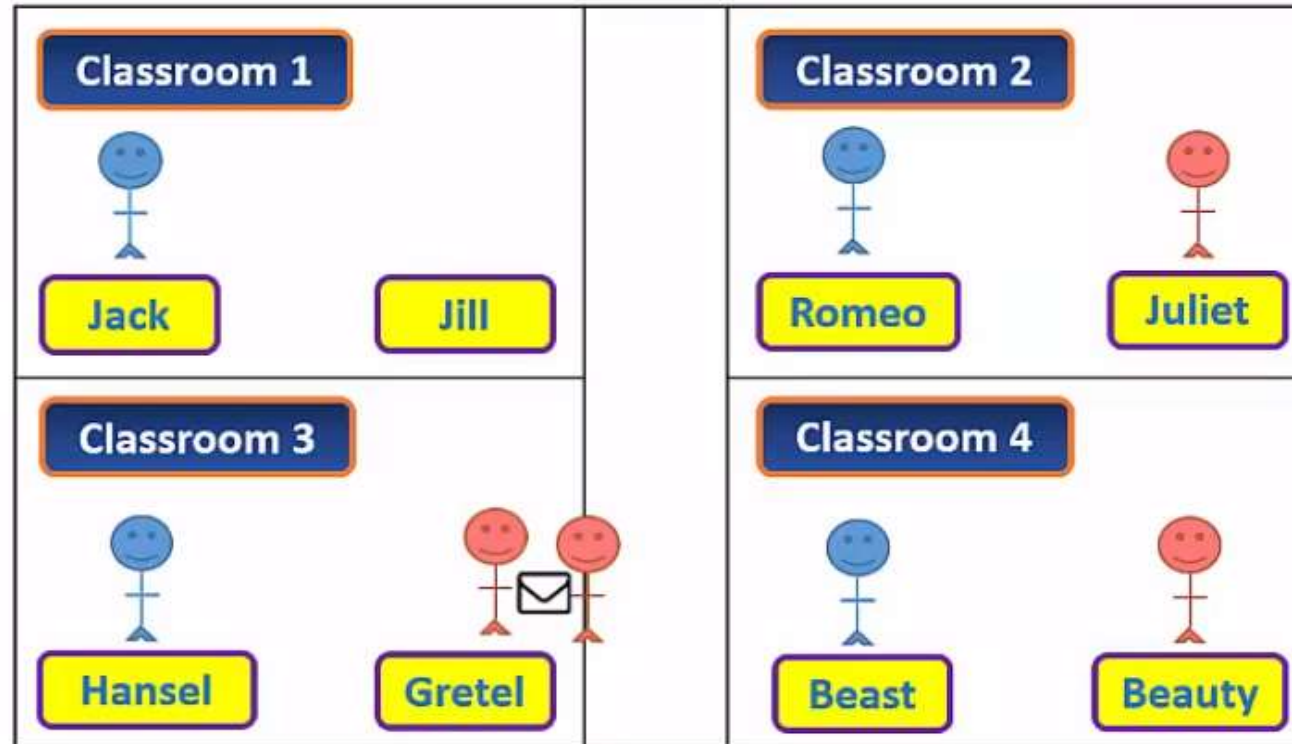
CLASSES: HELPER METHODS



CLASSES: HELPER METHODS



CLASSES: HELPER METHODS



Example 1:

If people are in the same classroom, they do not need the classroom name

Example 2:

If people are in a different classroom, they do need the classroom name

CLASSES: HELPER METHODS

```
public class Student
{
    public String getName()
    {
        return "Rand";
    }
    public int getAge()
    {
        return 20;
    }
    public double getGPA()
    {
        return 3.25;
    }
}
```

```
public class StudentRunner
{
    public static void main(String[] args)
    {
        Student stu = new Student();

        System.out.println("Name: " + stu.getName());
        System.out.println("Age : " + stu.getAge());
        System.out.println("GPA : " + stu.getGPA());
    }
}
```

Output:


```
Name: Rand
Age : 20
GPA : 3.25
```

```
}
```


CLASSES: HELPER METHODS

```
public class Student
{
    public String getName()
    {
        return "Rand";
    }
    public int getAge()
    {
        return 20;
    }
    public double getGPA()
    {
        return 3.25;
    }
    public String getInformation()
    {
        return "Name: " + getName() +
            "\nAge : " + getAge() +
            "\nGPA : " + getGPA();
    }
}

return "Name: " + this.getName() +
    "\nAge : " + this.getAge() +
    "\nGPA : " + this.getGPA();
```



```
public class StudentRunner
{
    public static void main(String[] args)
    {
        Student stu = new Student();

        System.out.println("Name: " + stu.getName());
        System.out.println("Age : " + stu.getAge());
        System.out.println("GPA : " + stu.getGPA());

        System.out.println(stu.getInformation());
    }
}
```

Output:

```
Name: Rand
Age : 20
GPA : 3.25
```

```
Name: Rand
Age : 20
GPA : 3.25
```

Example 1:
If METHODS are in
the same CLASS,
they do not need
an IDENTIFIER
name

Example 2:
If METHODS are in
a different CLASS,
they do need an
IDENTIFIER name

CLASSES: HELPER METHODS

```
public class Student
{
    public String getName()
    {
        return "Rand";
    }
    public int getAge()
    {
        return 20;
    }
    public double getGPA()
    {
        return 3.25;
    }
}
```

```
public class StudentRunner
{
    public static void main(String[] args)
    {
        Student stu = new Student();

        System.out.println("Name: " + stu.getName());
        System.out.println("Age : " + stu.getAge());
        System.out.println("GPA : " + stu.getGPA());

        System.out.println(stu.getInformation());
    }
}
```

Output:

```
Name: Rand
Age : 20
GPA : 3.25
```

```
Name: Rand
Age : 20
GPA : 3.25
```

```
        return "Name: " + getName()
            + "\nAge : " + getAge()
            + "\nGPA : " + getGPA();
    }

    return "Name: " + this.getName() +
        "\nAge : " + this.getAge() +
        "\nGPA : " + this.getGPA();
}
```

Example 1:
If METHODS are in
the same CLASS,
they do not need
an IDENTIFIER
name

Example 2:
If METHODS are in
a different CLASS,
they do need an
IDENTIFIER name

CLASSES: HELPER METHODS

```
public class Average
```

```
{  
    public double getAverage(int num1, int num2, int num3)  
    {  
        int sum = num1 + num2 + num3;  
        return (double) sum / 3;  
    }  
}
```

```
public class AverageRunner
```

```
{  
    public static void main(String[] args)  
    {  
        Average three = new Average();  
        double avg = three.getAverage(50,75,100);  
        System.out.println("The average of 50,75 and 100 is " + avg);  
    }  
}
```

Output:

```
The average of 50,75 and 100 is 75.0
```

CLASSES: HELPER METHODS

```
public class Average
{
    public double getAverage(int num1, int num2, int num3)
    {
        int sum = num1 + num2 + num3;
        return (double) sum / 3;
    }
}
```

```
public class AverageRunner
{
    public static void main(String[] args)
    {
        Average three = new Average();
        double avg = three.getAverage(50,75,100);

        System.out.println("The average of 50,75 and 100 is " + avg);
    }
}
```

Output:

The average of 50,75 and 100 is 75.0

CLASSES: HELPER METHODS

```
public class Average
{
    public double getAverage(int num1, int num2, int num3)
    {
        int sum = num1 + num2 + num3;
        return (double) sum / 3;
    }
    public int sumThree(int num1, int num2, int num3)
    {
        return n1 + n2 + n3;
    }
}
```

```
public class AverageRunner
{
    public static void main(String[] args)
    {
        Average three = new Average();
        double avg = three.getAverage(50,75,100);
        System.out.println("The average of 50,75 and 100 is ")
    }
}
```

Output:

The average of 50,75 and 100 is 75.0

CLASSES: HELPER METHODS

```
public class Average
{
    public double getAverage(int num1, int num2, int num3)
    {
        int sum = sumThree(num1, num2, num3);
        return (double) sum / 3;
    }

    public int sumThree(int num1, int num2, int num3)
    {
        return n1 + n2 + n3;
    }
}
```

```
public class AverageRunner
{
    public static void main(String[] args)
    {
        Average three = new Average();
        double avg = three.getAverage(50,75,100);

        System.out.println("The average of 50,75 and 100 is " + avg);
    }
}
```

Output:

The average of 50,75 and 100 is 75.0

CLASSES: HELPER METHODS

```
public class Average
{
    public double getAverage(int num1, int num2, int num3)
    {
        int sum = sumThree(num1, num2, num3);
        return (double) sum / 3;
    }
    private int sumThree(int num1, int num2, int num3)
    {
        return n1 + n2 + n3;
    }
}
```

```
public class AverageRunner
{
    public static void main(String[] args)
    {
        Average three = new Average();
        double avg = three.getAverage(50,75,100);
        System.out.println("The average of 50,75 and 100 is " + avg);
    }
}
```

Helper Method:
declared private and are
strictly for use inside of the
class that it is in

Output:

The average of 50,75 and 100 is 75.0

Helper Methods::

1. If METHODS are in the same CLASS, they do not need an IDENTIFIER name
2. If METHODS are in a different CLASS, they do need an IDENTIFIER name
3. Helper Method: declared private and are strictly for use inside of the class that it is in

Time to think?

- Is it possible to declare driver class in one file and implementation class in a different file?
- If yes, what are the criteria's to be considered?
- Is it abstraction?

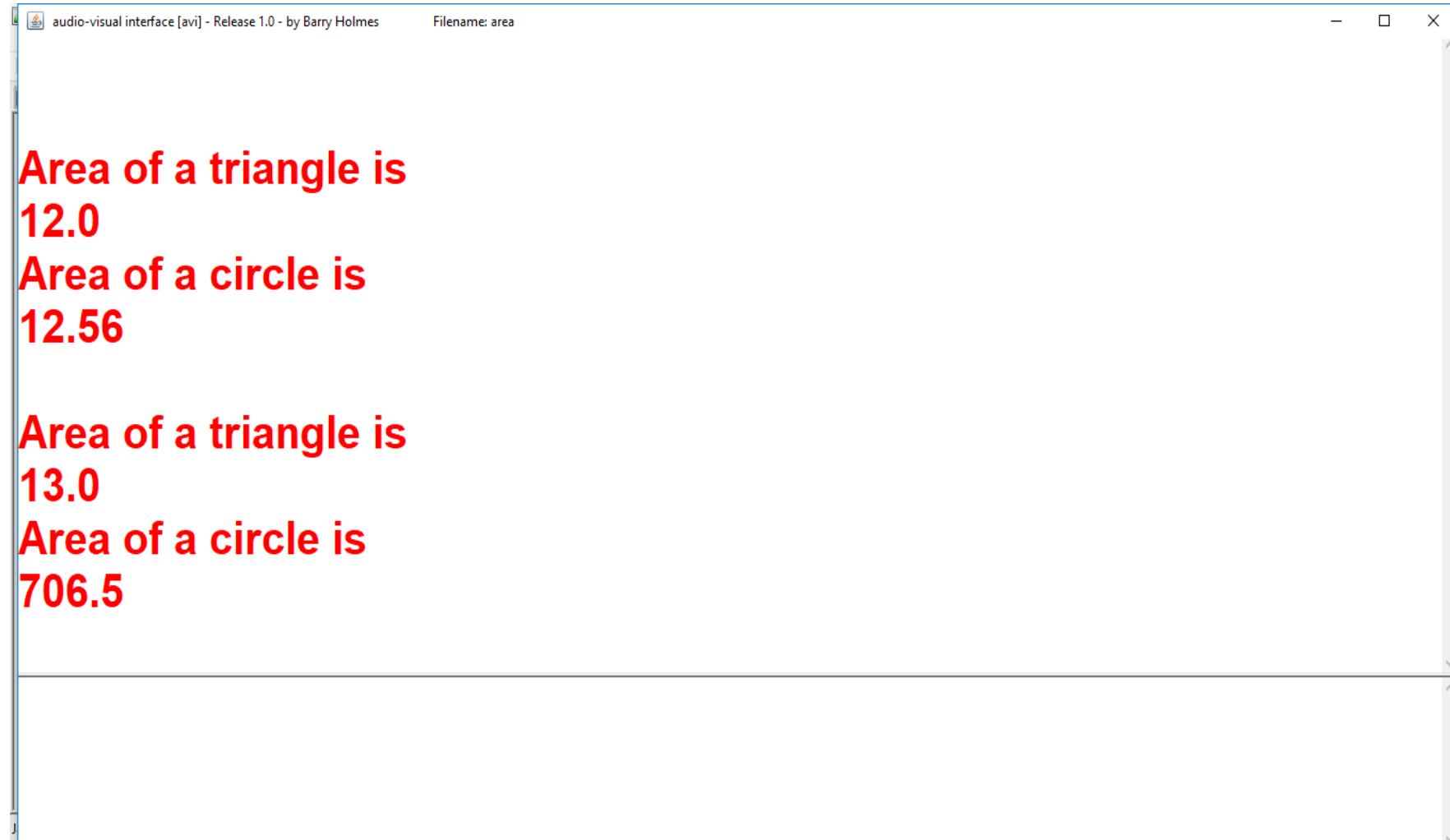
Static helper method

```
import avi.*;
class help1
{
    private static void display(Window screen,double re1,double re2)
    {
        screen.write("\n Area of a triangle is \n"+re1);
        screen.write("\n\nArea of a circle is\n"+re2);
    }

    public static void main(String args[])
    {
        double res1,res2;
        Window screen = new Window("area","bold","red",56);
        screen.showWindow();
        Shapes s1= new Shapes(12,2);
        Shapes s2 = new Shapes(2);
        Shapes s3= new Shapes(13,2);
        Shapes s4= new Shapes(15,5);
        display(screen,s1.tri(),s2.cir());
        display(screen,s3.tri(),s4.cir());
    }
}
```

```
public class Shapes
{
    int a, b;
    public Shapes(int r)
    {
        a=r;
    }
    public Shapes(int l, int p)
    {
        a=l;
        b=p;
    }
    public double tri()
    {
        return (0.5)*a*b;
    }
    public double cir()
    {
        return (3.14)*a*a;
    }
}
```

output




```

import avi.*;
class Example_4
{
    a 'helper' method to display the statistics of the pool
    private static void displayStatistics(Window screen,String nameOfPool,float volume,float capacity,float time)
    {
        screen.write(nameOfPool+"\n\tVolume: "+volume+" cubic feet\n");
        screen.write("\tCapacity: "+capacity+" US gallons\n");
        screen.write("\tTime to fill: "+time+" hours\n\n");
    }
    public static void main(String[] args)
    {
        final float RATE_OF_FLOW = 50.0f;
        Window screen = new Window("Example_4.java","bold","blue",24);
        screen.showWindow();
        create a large SwimmingPool object
        SwimmingPool largePool = new SwimmingPool(100.0f,30.0f,3.0f,8.0f);
        displayStatistics(screen,"Large Pool",largePool.volumeOfWater(),
            largePool.capacityOfPool(),largePool.timeToFillPool(RATE_OF_FLOW));

        create a small SwimmingPool object
        SwimmingPool smallPool = new SwimmingPool(50.0f,20.0f,5.0f,5.0f);
        displayStatistics(screen,"Small Pool",smallPool.volumeOfWater(),
            smallPool.capacityOfPool(), smallPool.timeToFillPool(RATE_OF_FLOW));
    }
}

```

Static methods

- Static Method in Java belongs to the class and not its instances.
- A static method can access only **static variables of class** and invoke only **static methods of the class**.
- Usually, **static methods are utility methods** that we want to expose to be used by other classes without the need of creating an instance.
- **Java Wrapper classes** and utility classes contain a lot of static methods.
- The main() method that is the entry point of a java program itself is a static method.

Static methods

- **Utility Class**, also known as Helper **class**, is a **class**, which contains just static methods, it is stateless and cannot be instantiated.
- It contains a bunch of related methods, so they can be reused across the application.

```
class helper_met // a helper class / utility class
{
    public static String preMr(String name) {
        return "Mr. " + name;
    }
    public static String preMs(String name) {
        return "Ms. " + name;
    }
    public static String preDr(String name) {
        return "Dr. " + name;
    }
}

public class helper_met1{
    public static void main(String[] args) {
        String name = "John";
        System.out.println(helper_met.preMr(name));
    }
}
```

Scope and Lifetime of Identifiers

- The scope of an identifier refers to the region of a program in which an identifier can be used.
- An identifier can have either **class scope** or **block scope**.
- An identifier with **class scope** is accessible from its point of declaration throughout the **entire class**.
- an identifier with **block scope** is accessible only from the **point of declaration to the end of the block**.
- A block begins with an open brace { and ends with a close brace } and contains declarations and executable statements.

Scope and Lifetime of Identifiers

- The life time of an identifier is the period during which the value of the identifier exists in computer memory.
- The lifetime of an identifier will vary according to the nature of the identifier.
- Identifiers declared as being **static exist for the life of the program**.
- whereas **parameters and identifiers** having block scope exist only during the **execution of the method**.
- When an **object goes out of scope**, the amount of memory allocated to storing that object is **returned back to the heap** for future use by other objects.
- The Java system automatically returns memory to the heap when it is no longer required. This process is known as **garbage collection**.

Software Development

The software development cycle consists of a set of **four phases**, each phase is not deemed to be entirely completed before moving on to the next phase.

Because the development of software evolves through the experience gained at each stage, **it is possible to go back to any of the stages and modify the solution to the problem.**

During **analysis**, the customer who commissioned the system to be built and the software developers who construct the system meet to **agree upon a description of the problem.**

In the **design** phase, **plans are generated for building the system.**

The **programming** of a software project combines **coding, testing**, and the integration of the various software components to construct the software system

The **maintenance** phase may take the form of simply **changing and retesting small amounts of code.**

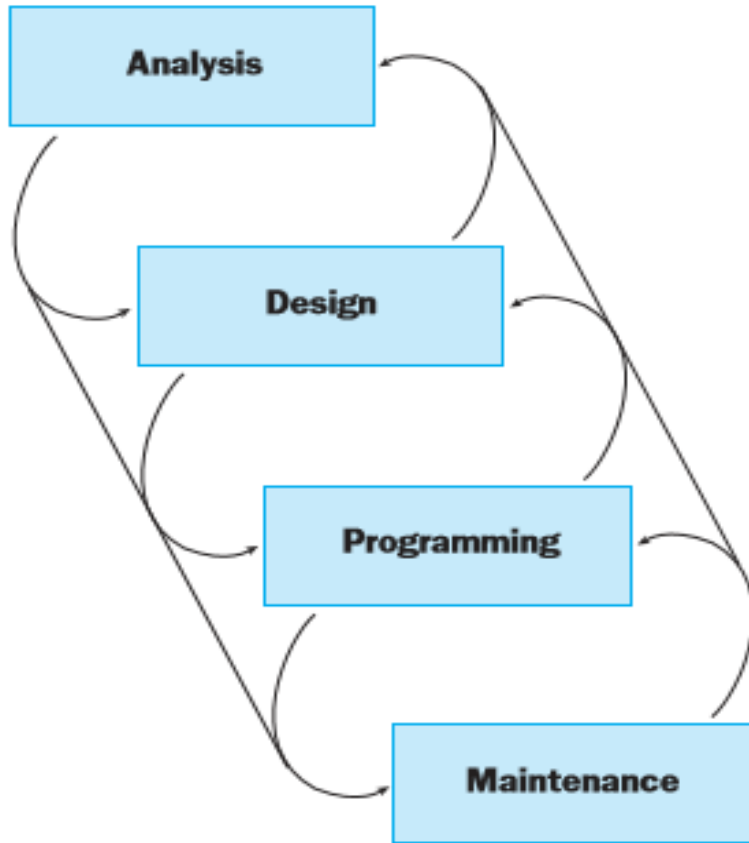


Figure 3.3 Software development life cycle

Object-Oriented Program Design

- Identify the Classes and Methods
- Algorithm Development
- Testing
- Compilation and Execution
- Documentation

Object-Oriented Program Design

- Identify the Classes and Methods

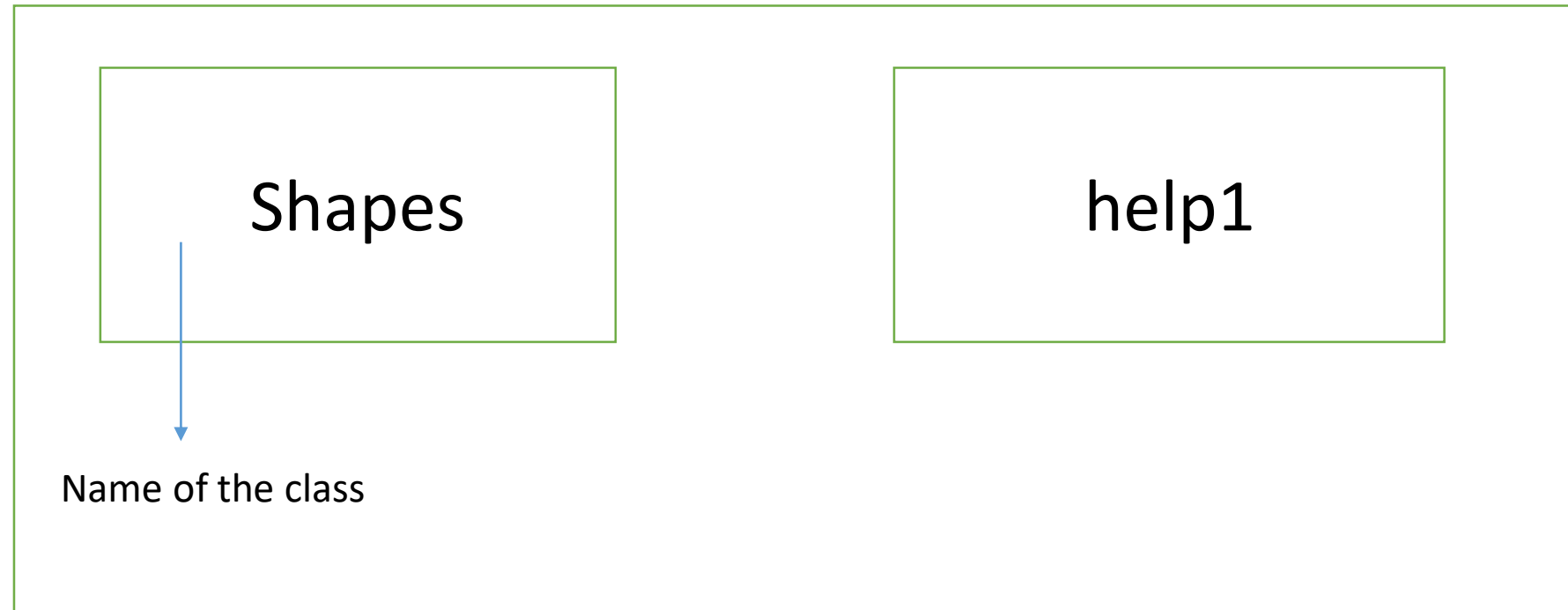


Fig: The simplest UML representation of classes

Object-Oriented Program Design

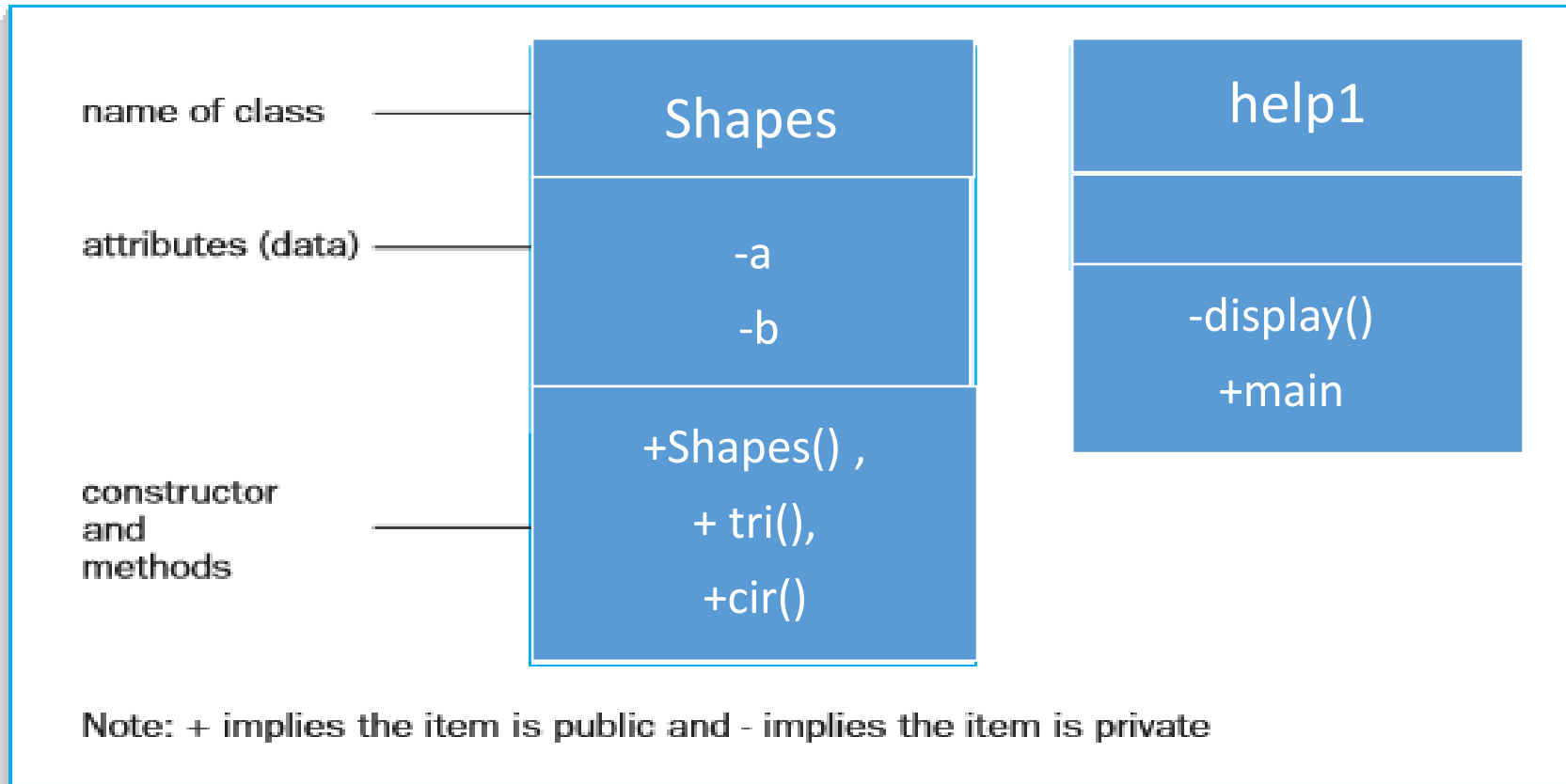
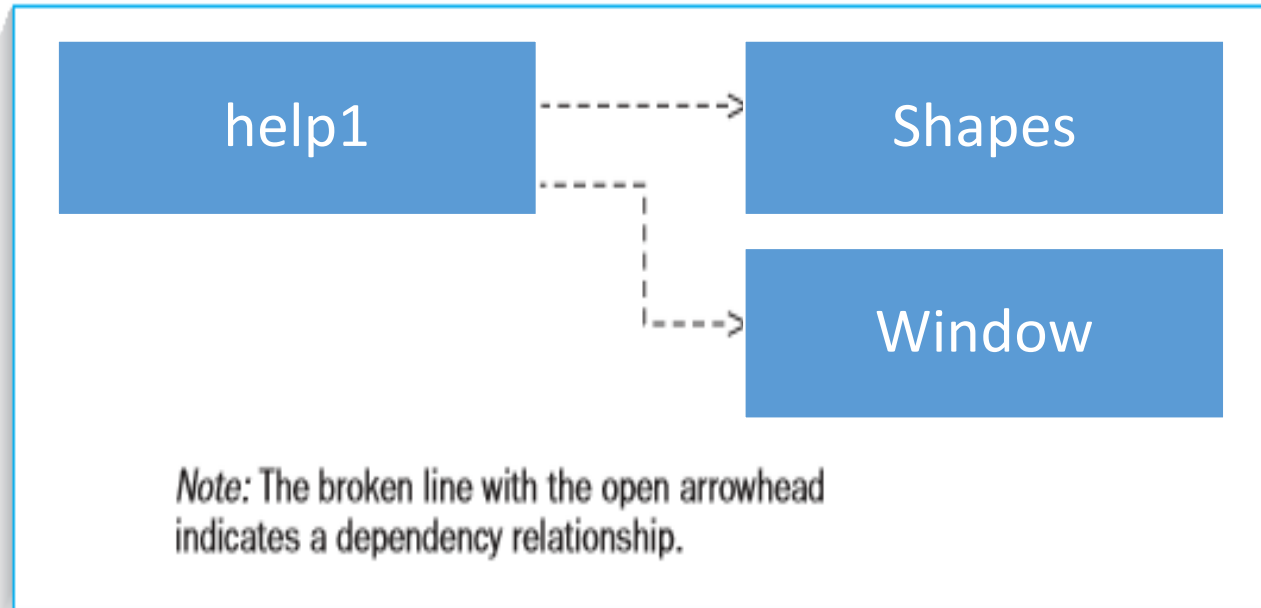


Figure 3.6 An expanded UML representation of classes, their attributes, and methods

Object-Oriented Program Design



The next stage looks for **any dependencies** that may exist between classes.

Classes can build upon and cooperate with other classes.

Often **one class depends upon another class** because it cannot be used unless the other class exists.

Figure 3.7 illustrates how the **help1** class is dependent upon the existence of a **Shapes** class and a **Window class** (found in the avi package).

Figure 3.7 The UML representation of dependencies between classes

Note the use of broken lines with **open arrow heads** to denote the dependency relationship.

Object-Oriented Program Design

- Algorithm Development

An algorithm is a solution to a problem and is expressed as a series of operations for the computer to obey. Each constructor and method will have its own algorithm

- create a window object—screen
- create a swimming pool object—large pool
- write the statistics of the large pool to the screen
- create a swimming pool object—small pool
- write the statistics of the small pool to the screen

- Testing

the next step is to trace through the algorithm with test data to verify that the solution contains no logical errors.

- Compilation and Execution

When the compilation is successful, and regardless of the testing technique adopted, further testing, often using the same test data as in the desk check, is always carried out by running the program with test input and checking if the output is correct.

Object-Oriented Program Design

- Documentation

- The standard Java tools include a documentation aid called **javadoc**, the Java API documentation generator.
- Before the coding of a method, even before the constructor, include a comment as follows.

/** Textual annotation of the purpose of the method.

@param Name of parameter followed by a description of its purpose.(This line must be repeated for each parameter the method contains.)

@return Description of the data being returned.(This line is omitted if the method returns void.)

```
// program to demonstrate the creation of class
```

```
public class Shapes
```

```
{
```

```
    int a, b; // instance variables
```

```
// constructor
```

```
/** The Shapes class enables an object that represents to calculate the area of a circle and triangle. @param r is the radius/length of the circle/triangle. @param p is the width of the triangle. */
```

```
    public Shapes(int r)
```

```
    {
```

```
        a=r;
```

```
    }
```

```
    public Shapes(int l, int p)
```

```
    {
```

```
        a=l;
```

```
        b=p;
```

```
    }
```

```
// instance methods
```

```
/** Calculates the area of a triangle. @return The area of a triangle. */
```

```
public double tri()
```

```
{
```

```
    return (0.5)*a*b;
```

```
}
```

```
/** Calculates the area of a circle. @return The area of a circle. */
```

```
public double cir()
```

```
{
```

```
    return (3.14)*a*a;
```

```
}
```

Output of Shapes class using Javadoc command

```
C:\Users\rashmi.RASHMI1\Desktop\java\java pgms>javadoc Shapes.java
Loading source file Shapes.java...
Constructing Javadoc information...
Standard Doclet version 11.0.1
Building tree for all the packages and classes...
Generating .\Shapes.html...
Generating .\package-summary.html...
Generating .\package-tree.html...
Generating .\constant-values.html...
Building index for all the packages and classes...
Generating .\overview-tree.html...
Generating .\index-all.html...
Building index for all classes...
Generating .\allclasses-index.html...
Generating .\allpackages-index.html...
Generating .\deprecated-list.html...
Building index for all classes...
Generating .\allclasses.html...
Generating .\allclasses.html...
Generating .\index.html...
Generating .\help-doc.html...

C:\Users\rashmi.RASHMI1\Desktop\java\java pgms>
```

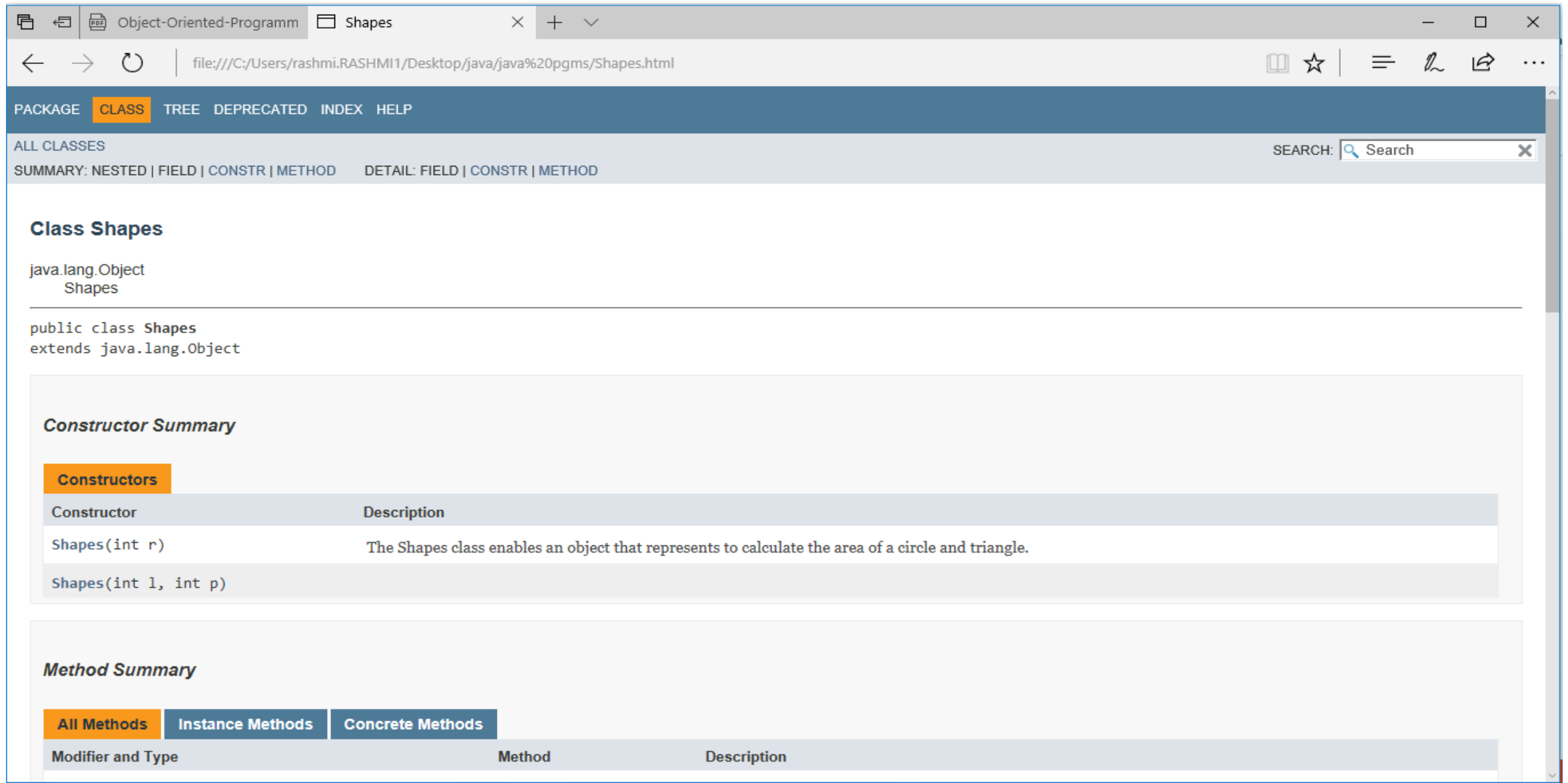



Figure 3.8 An illustration of the online documentation being viewed by a browser

Object-Oriented-Programm

Shapes

file:///C:/Users/rashmi.RASHMI1/Desktop/java/java%20pgms/Shapes.html

PACKAGECLASSTREEDEPRECATEDINDEXHELP

ALL CLASSES

SEARCH:

SUMMARY: NESTED | FIELD | CONSTR | METHODDETAIL: FIELD | CONSTR | METHOD

All Methods

Instance Methods

Concrete Methods

Modifier and Type	Method	Description
double	<code>cir()</code>	Calculates the area of a circle.
double	<code>tri()</code>	Calculates the area of a triangle.

Methods inherited from class java.lang.Object

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

Shapes

```
public Shapes(int r)
```

The Shapes class enables an object that represents to calculate the area of a circle and triangle.

Parameters:

`r` - is the radius of the circle.

Object-Oriented-Programm

Shapes

×

+

▼

←

→

↺

file:///C:/Users/rashmi.RASHMI1/Desktop/java/java%20pgms/Shapes.html

📖

☆

≡

🔍

🔗

⋮

PACKAGE **CLASS** TREE DEPRECATED INDEX HELP

ALL CLASSES

SEARCH:

×

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Shapes

```
public Shapes(int l,
              int p)
```

Parameters:

l - is the length of the triangle,

p - is the width of the triangle.

Method Detail

tri

```
public double tri()
```

Calculates the area of a triangle.

Returns:

The area of a triangle.

Object-Oriented-Programm

Shapes

×

+

▼

—

□

×

←→↺

file:///C:/Users/rashmi.RASHMI1/Desktop/java/java%20pgms/Shapes.html

📖☆|☰🔍🔗⋮

PACKAGECLASSTREEDEPRECATEDINDEXHELP

ALL CLASSES

SEARCH: 🔍 Search

SUMMARY: NESTED | FIELD | CONSTR | METHODDETAIL: FIELD | CONSTR | METHOD

tri

public double tri()

Calculates the area of a triangle.

Returns:

The area of a triangle.

cir

public double cir()

Calculates the area of a circle.

Returns:

The area of a circle.

PACKAGECLASSTREEDEPRECATEDINDEXHELP

ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHODDETAIL: FIELD | CONSTR | METHOD

Object-Oriented Program Design

- The documentation is normally generated after the compilation phase is satisfactorily completed with no errors present in the source code.
- Use the same window (MS-DOS or terminal) that you used for compilation to input the javadoc command.
- A simplified form of the syntax to produce Java documentation is:

javadoc classname.java

- Notice the amount of information that is output by this command during the automated documentation process.

The AVI Package Revisited

- time to explain about three more classes from the avi package. These classes are the **Audio, FilmStrip, and Timer classes**.
- One feature of the **Audio and FilmStrip** classes is they both **use arrays** in their constructors.
- You already know how to initialize the array **args** using command-line string data.
- however, it is also possible to initialize arrays other than args with string data at the point of declaration of the array.
- For example, if we wanted to declare an array named filename and store just the one name of a sound file in the first cell (indexed as 0), then we would code:

```
String[] filename = {"funky.wav"};
```

The AVI Package Revisited

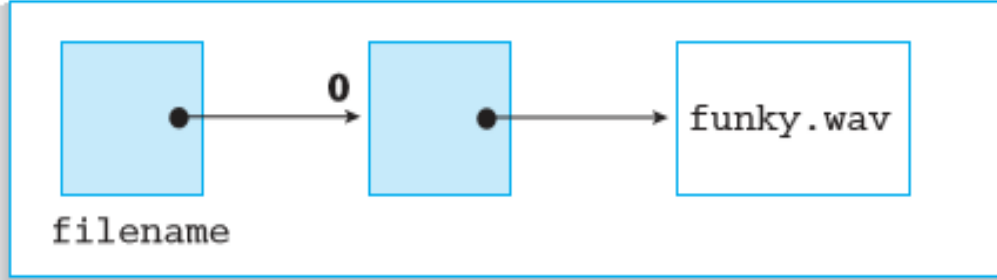


Figure 3.11 Array filename

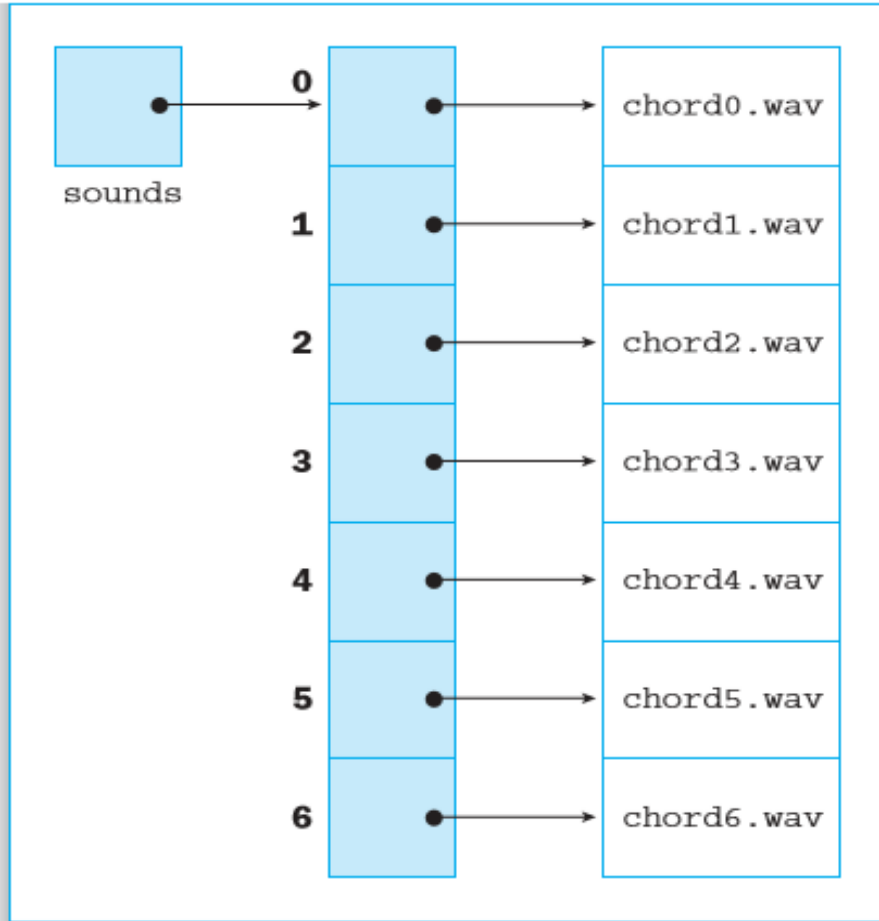
```
String[] filename = {"funky.wav"};
```

The representation of the data in memory is depicted in Figure 3.11.

Remember **an array is an object**. Therefore, it is stored by **reference and not by value**, hence the arrow from the memory location filename to the first (and only) cell of the array.

However, a **string is also an object**, and again is stored by reference, hence the arrow from the only cell (indexed 0) to the memory containing the string representation of the name of the file.

The AVI Package Revisited



If we wanted to declare another array named `sounds` and store the names of seven sound files in the cells of the array indexed from 0 through to 6, respectively, then we would code:

```
String[] sounds = {"chord0.wav","chord1.wav","chord2.wav",  
"chord3.wav", "chord4.wav","chord5.wav","chord6.wav"};
```

Once again, the array `sounds` is an object and is stored by reference—hence the arrow from the memory location `sounds` to cell 0 of the array (see Figure 3.12).

Notice that the consecutive cells, indexed 0 through 6, of the array all contain references to the string representation of the name of each sound file.

Figure 3.12 Array `sounds`

The AudioClass

The Audio class contains the following constructor and instance methods:

```
public class Audio
{
    public Audio(Window parent, String[] filenames);
    public void playSound(int index);
    public static void beep(WindowPane parent);
}
```

To create an Audio object, you must use the class constructor that requires two items of data in the formal parameter list:

parent—a Window type that specifies the container onto which to display any information or error messages about the sound files.

filenames—a string array containing the names of the sound files that are to be played.

The AudioClass

Assuming that you have already created a **screen window object** and a **string array filename**, to create an object named output of the Audio class, the Audio constructor would be coded as

```
Audio output = new Audio(screen, filename);
```

The sound files whose names are in the string array can then be played using the instance method **playSound**.

For example, if you wanted to play the sound file stored in cell 0 of the array,code

```
output.playSound(0);
```

The AudioClass

- Note: Only wav and au format audio files can be successfully played using the Audio class.
- The wav-formatted files tend to eat up your hard drive if you are a real sound bug. This format allows 16-bit stereo samples up to 44.1 KHz.

Sample Program

```
import avi.*;
class Audi
{
    static public void main(String[] args)
    {
        String[] filename = {"sound/Recording1.wav"};
        // create window
        Window screen = new Window("Audi.java", "bold", "blue", 16);
        screen.showWindow();
        // create sound object and play sound
        Audio output = new Audio(screen, filename);
        output.playSound(0);
        // display information
        screen.write("WHEN THE MUSIC STOPS CLOSE THE WINDOW.");
    }
}
```

The Timer Class

- Since the method is static, you do not use an object to invoke the method. Instead it is called directly, for example by

`Timer.delay(5)` - if you require a delay of five seconds.

- Other static methods in this class will allow you to get the `current hour`, minute, and second from the computer's clock—these are `getHour()`, `getMinute()`, and `getSecond()`, respectively.
- Should you want to get the `current time` of day, then use `getTime()`. This method returns a string giving the time in hours, minutes, and seconds.
- The final static method, `getDate()`, returns the `current date` as a string of characters.

The Timer Class

- The Timer class contains the following class or static methods:

```
public class Timer
{
    public static void delay(int seconds);
    public static int getHour();
    public static int getMinute();
    public static int getSecond();
    public static String getTime();
    public static String getDate();
}
```

This class is important to the avi package. Since it offers a **delay class method** that allows a **pause** between playing sounds or showing pictures.

```
// program to demonstrate the Timer class and playing sounds, one after another
import avi.*;
class timer1
{
    static private void music(Audio output, int index)
    {
        final int DURATION = 3;
        output.playSound(index);
        Timer.delay(DURATION);
    }
    static public void main(String[] args)
    {
        String[] filenames = {"sound/Recording1.wav", "sound/sound1.wav", "sound/sound2.wav"};
        // create window
        Window screen = new Window("timer1.java", "bold", "blue", 16);
        screen.showWindow();
        // create sound object and play sound
        Audio output = new Audio(screen, filenames);
        // play music
        music(output, 0);
        music(output, 1);
        music(output, 2);
        screen.write("WHEN THE SOUNDS STOP CLOSE THE WINDOW.");
    }
}
```

Task to do?

- Write a program to create a timer class object to retrieve current date, time, hour, minute and second. Display the output using window class object.

The FilmStrip Class

This class is used to output JPEG, GIF, or animated GIF images onto the screen.

```
public class FilmStrip
{
    public FilmStrip(Window parent,
        String[] filenames,
        int widthOfFrame,
        int heightOfFrame);
    public void showFilmStrip();
    public void hideFilmStrip();
    public void showFrame(int frame);
    public void showFrames(int[] frames);
    public void clearImages();
}
```

The arguments that you supply can be literal constants or variables.

parent—a Window type that specifies the container onto which to place the picture or pictures.

filenames—a string array containing the names of the image files that are to be output.

widthOfFrame and heightOfFrame—the width and height, respectively, of the image, in pixels.

The FilmStrip Class

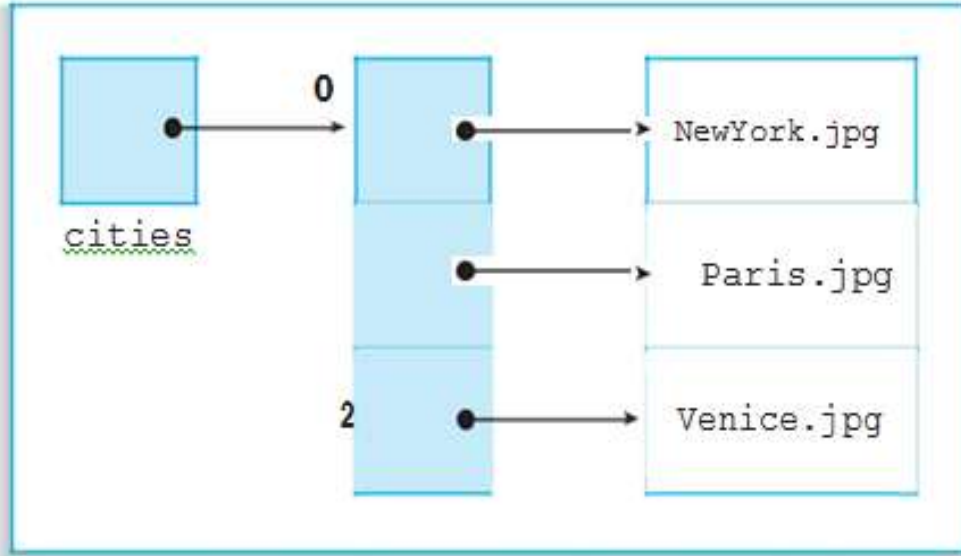


Figure 3.13 Array `cities`

Figure 3.13 illustrates how the strings defined by the following declaration are stored in the array `cities`.

```
String[] cities = {"NewYork.jpg", "Paris.jpg", "Venice.jpg"};
```

Assuming that a screen object has already been created, it is possible to create a `FilmStrip` object by coding:

```
FilmStrip capitals = new FilmStrip(screen, cities, IMAGE_WIDTH,  
IMAGE_HEIGHT);
```

All the images represented by the three filenames in the array `cities` can be displayed as a film strip by coding: `capitals.showFilmStrip()`.

The FilmStrip Class

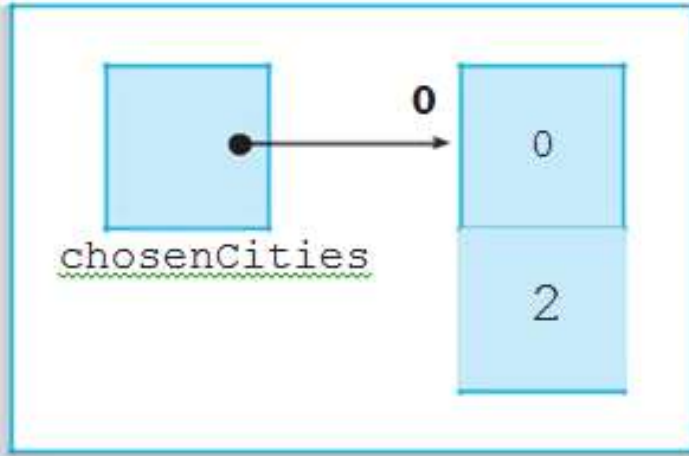


Figure 3.14 An array of integers

It is possible to declare and initialize an array of integers as follows:

```
int[] chosenCities = {0,2};
```

Figure 3.14 illustrates the creation and storage of this integer array containing the values 0 and 2 at cells 0 and 1, respectively.

To show a selection of frames from the film strip you must first initialize an integer array with the indexed positions of the selected images from the cities array.

If you wanted to show the images for New York and Venice only, then you must first create an array of indices that correspond to the positions of the filenames of these images in the cities array.

```
int[] chosenCities = {0,2};
```

You may show the images for New York and Venice by coding:

```
capitals.showFrames(chosenCities);
```

The FilmStrip Class

- All of the images may be cleared from the screen by using the instance method `clearImages()`.
- The film strip may be temporarily hidden from view by using the method `hideFilmStrip()`. The film strip may be brought back into view by the method `showFilmStrip()`.

program to display images

```
import avi.*;
class Example_9
{
    public static void main(String[] args)
    {
        // store the names of the three image files
        String[] cities = {"NewYork.jpg","Paris.jpg","Venice.jpg"};

        // create a Window object screen
        Window screen = new Window("Example_9.java");
        screen.showWindow();

        // declare size of image
        final int IMAGE_WIDTH = screen.getWidth()/5;
        final int IMAGE_HEIGHT = (int)((float)IMAGE_WIDTH * 0.667);

        create a film strip object containing the three images found in the files
        FilmStrip capitals = new FilmStrip(screen,cities,IMAGE_WIDTH,IMAGE_HEIGHT);
        screen.write("image files are\ncopyright "+"(c) 2000 Barry Holmes");
    }
}
```

show the film strip on the screen for 5 seconds

```
capitals.showFilmStrip();  
Timer.delay(5);  
capitals.clearImages();
```

show only Paris for 5 seconds from the filmstrip

```
capitals.showFrame(1);  
Timer.delay(5);  
capitals.clearImages();
```

show both NewYork and Venice from the film strip

```
int[] chosenCities = {0,2}; capitals.showFrames(chosenCities);  
Timer.delay(5);
```

destroy window and exit

```
screen.closeWindowAndExit();  
}  
}
```



SELECTION

The More AVI:

The slider Class:

```
public class Slider
{
    public Slider(Window parent,
                  String prompt,
                  int minValue,
                  int maxValue,
                  int increment);
    public void showSlider();
    public int getValue();
}
```

- Sliders are used for the **input of integer values**.
- The advantage of a slider over a dialog box is that you can guarantee that a user will **input data only within predefined limits**.
- Erroneous data cannot be input since the user is constrained to move the slider between preset limits.

The slider Class:

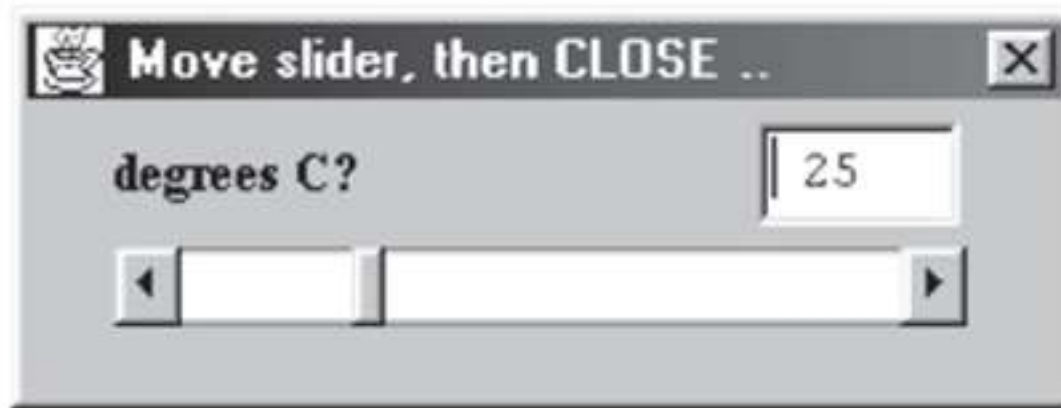


Figure 4.1 A slider object from the `avi` package

To create a Slider object you must use the class constructor that requires five items of data in the formal parameter list:

parent—a Window type that specifies the container on which to display a slider object.

prompt—a description of the quantities being represented; it will be written in the slider box as a prompt.

program to demonstrate the use of a slider to input a temperature in degrees Celsius and convert the value to degrees Fahrenheit

```
import avi.*;
```

```
public class Example_1
```

```
{
```

```
public static void main(String[] args)
```

```
{
```

```
int celsius;
```

```
// create and show window
```

```
Window screen = new Window("Example_1.java");
```

```
screen.showWindow();
```

```
//create and show slider
```

```
Slider inputTemperature = new Slider(screen,"degrees C?",0,100,1);
```

```
inputTemperature.showSlider();
```

```
//input temperature in degrees celsius
```

```
celsius = inputTemperature.getValue();
```

```
convert temperature to degrees Fahrenheit and display the value on the screen
```

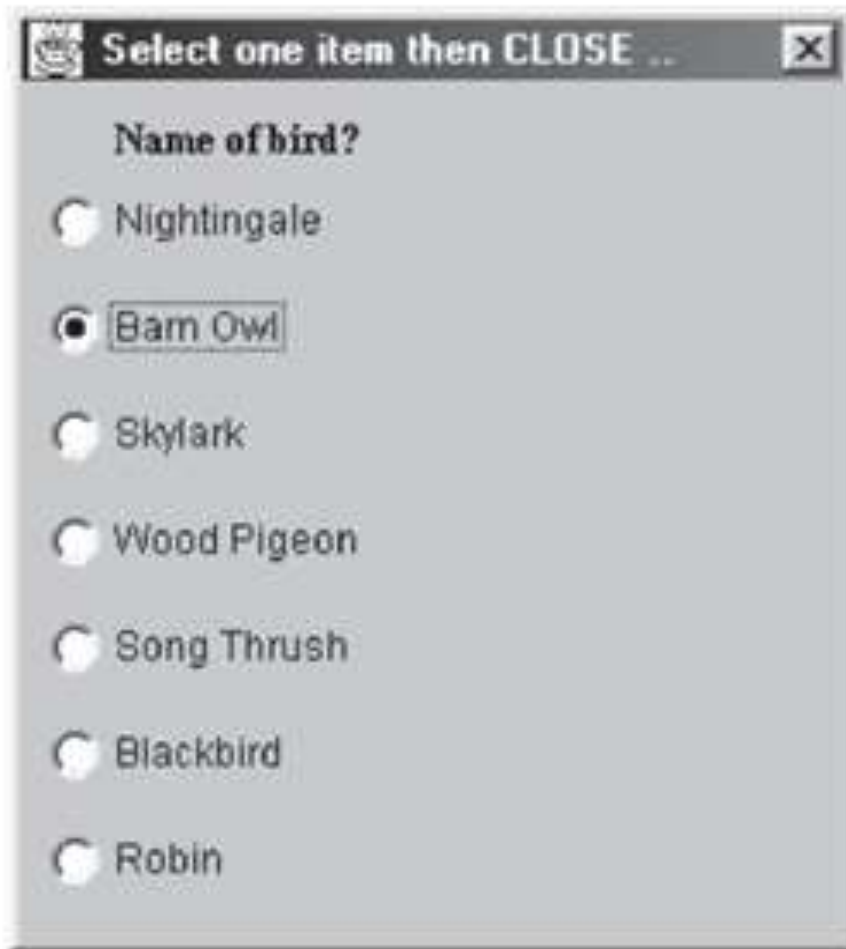
```
screen.write("Temperature input was "+celsius+" C; ");
```

```
screen.write("equivalent temperature is "+ ((celsius)*(9.0f/5.0f)+32)+" F\n");
```

```
}
```

```
}
```

The Radio Button Class:



- A group of radio buttons may be used for the **input of just one item of data** selected from many items.
- The **mouse-pointer** is used to depress a button.
- The key feature of radio buttons is that **only one button may indicate a choice**.

Figure 4.2 A `RadioButtons` object from the `avi` package

The Radio Button Class:

```
public class RadioButtons
{
    public RadioButtons(Window
        parent,
            String prompt,
            String[] itemsInList);
    public void showRadioButtons();
    public void getNameOfButton()
    public int getPositionOfButton();
}
```

To create a radio-buttons object you must use the class constructor that requires three items of data in the formal parameter list:

parent—a Window type that specifies the container on which to display a radio-buttons object.

prompt—a string that is used as a cue to inform the user of the nature of the selection. For example, in Figure 4.2 the cue is "Name of bird?".

itemsInList—a string array containing a list of all the names of the radio buttons.

program to demonstrate the use of radio buttons for the selection of just one item of text from a list of items, then use the position of the selected button to select an audio file

```
import avi.*;  
class Example_2
```

```
{  
public static void main(String[] args)  
{
```

store names of files of birdsong in an array

```
String[] songs = {"Nightingale.wav","Barn Owl.wav","Skylark.wav","Wood Pigeon.wav","Song  
Thrush.wav","Blackbird.wav","Robin.wav"};
```

// store names of birds in an array

```
String[] names = {"Nightingale","Barn Owl","Skylark", "Wood Pigeon","Song Thrush","Blackbird",  
"Robin"};
```

// create and show window pane

```
Window screen = new Window("Example_2.java"); screen.showWindow();
```

// create sound object

```
Audio birdSong = new Audio(screen, songs);
```

create and show radio buttons

```
RadioButtons input = new RadioButtons(screen, "Name of bird?", names);  
input.showRadioButtons();
```

select from radio buttons list

```
int position = input.getPositionOfButton();
```

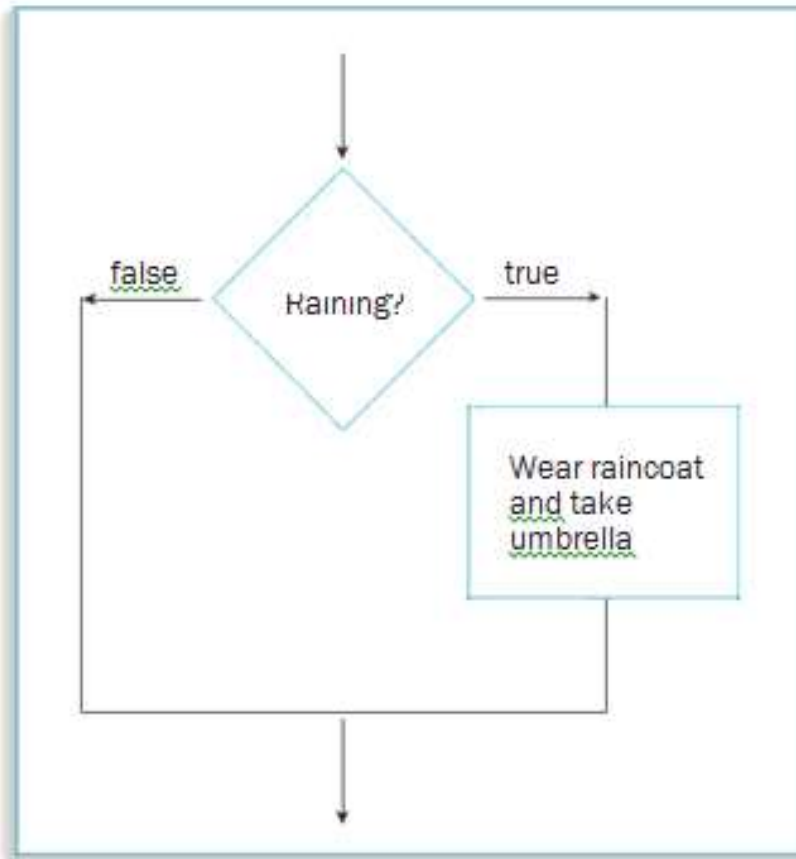
play sound of selected bird

```
birdSong.playSound(position);
```

// write information to the screen

```
screen.write("You should be listening to the song of a "+names[position]+".\n\n");  
screen.write("Sound files edited and digitized by Barry "+"Holmes.\n\n\n");  
screen.write("Close the window when the bird song finishes.");  
}  
}
```

If...Else Statement:



- If the conditional expression is **true**, then the statement(s) will be executed; if the conditional expression is **false**, then the statement(s) will not be executed.
- Afterward, the computer will continue with the execution of the next statement after statement(s).

Figure 4.3 Single-branch selection

If...Else Statement:

The syntax of the if statement follows:

SYNTAX

If statement: `if (conditional-expression)`
`statement(s);`

If...Else Statement:

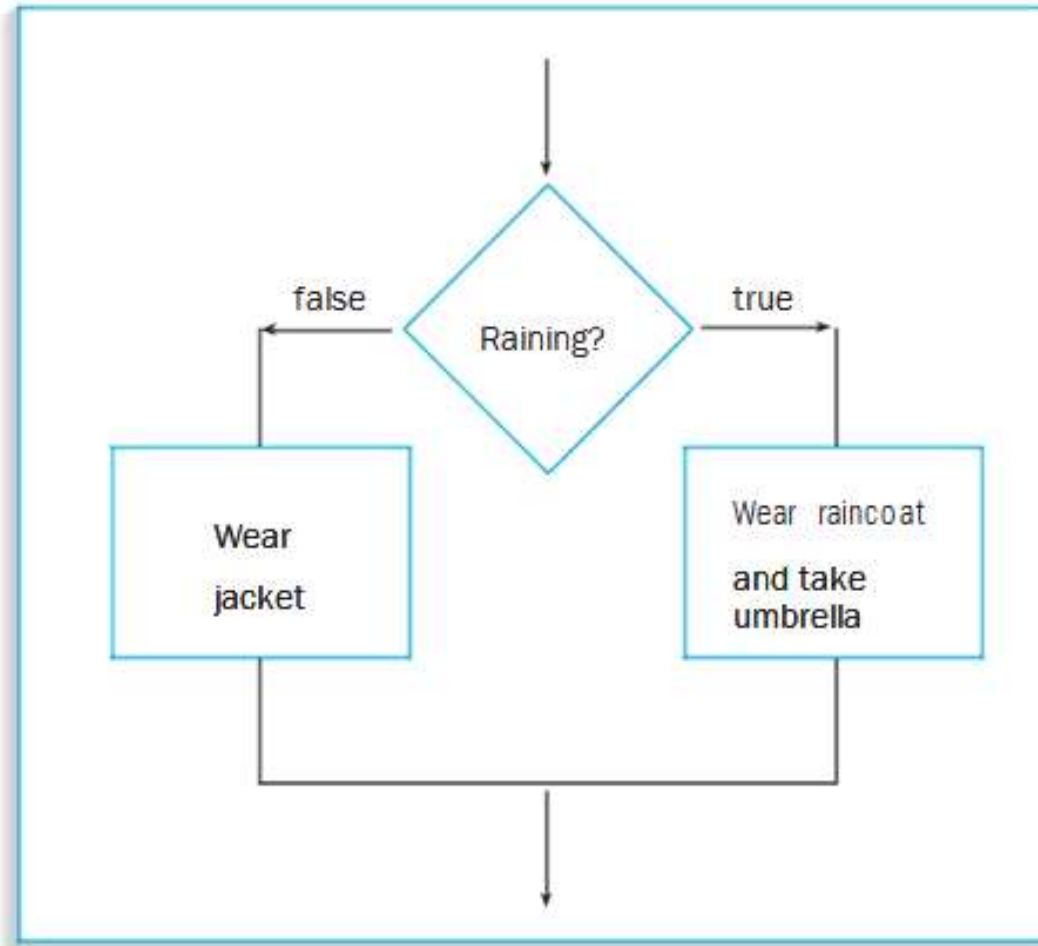


Figure 4.4 Double-branch selection

program to demonstrate the if statement

```
import avi.*;
public class Example_3
{
public static void main(String[] args)
{
String[] buttons = {"DRY","RAINING"};
String weather;
```

// create and show window

```
Window screen = new Window("Example_3.java","bold","blue",18);
screen.showWindow();
```

//create and show radio buttons

```
RadioButtons inputWeather = new RadioButtons(screen,"Weather conditions?",buttons);
inputWeather.showRadioButtons();
```

//get selection from button

```
weather = inputWeather.getNameOfButton();
```

// display what to wear

```
if (weather.equals("RAINING"))
screen.write("It's raining outside wear your raincoat "+ "and take an umbrella.");
screen.write("\n\n\nClose the window to exit.");
}
}
```

The contents of the log file after running this program follows.

```
=====
                        L O G      F I L E
  audio-visual interface [avi] - Release 1.0 - by Barry Holmes
  filename: Example_3.java   date: 3/12/2000      time: 5:35:50
=====
```

At the prompt: Weather conditions?, you selected [RAINING] from the radio buttons.

It's raining outside wear your raincoat and take an umbrella.

Close the window to exit.

program to demonstrate the if..else statement

```
import avi.*;
public class Example_4
{
    public static void main(String[] args)
    {
        String[] buttons = {"DRY","RAINING"};
        String weather;

        // create and show window pane
        Window screen = new Window("Example_4.java","bold","blue",18);
        screen.showWindow();

        //create and show radio buttons
        RadioButtons inputWeather = new RadioButtons(screen,"Weather conditions?",buttons);
        inputWeather.showRadioButtons();

        //get selection from radio button
        weather = inputWeather.getNameOfButton();

        //write what to wear
        if (weather.equals("RAINING"))
            screen.write("It's raining outside wear your raincoat and "+ "take an umbrella.");
        else
            screen.write("The weather is dry wear your jacket.");
        screen.write("\n\n\nClose the window to exit.");
    }
}
```

The contents of the log file after running this program follows.

```
=====
                        L O G      F I L E
  audio-visual interface [avi] - Release 1.0 - by Barry Holmes
  filename: Example_4.java   date: 3/12/2000      time: 5:34:15
=====
```

At the prompt: Weather conditions?, you selected [DRY] from the
radio buttons.

The weather is dry wear your jacket.

Close the window to exit.

Nested If Statement:

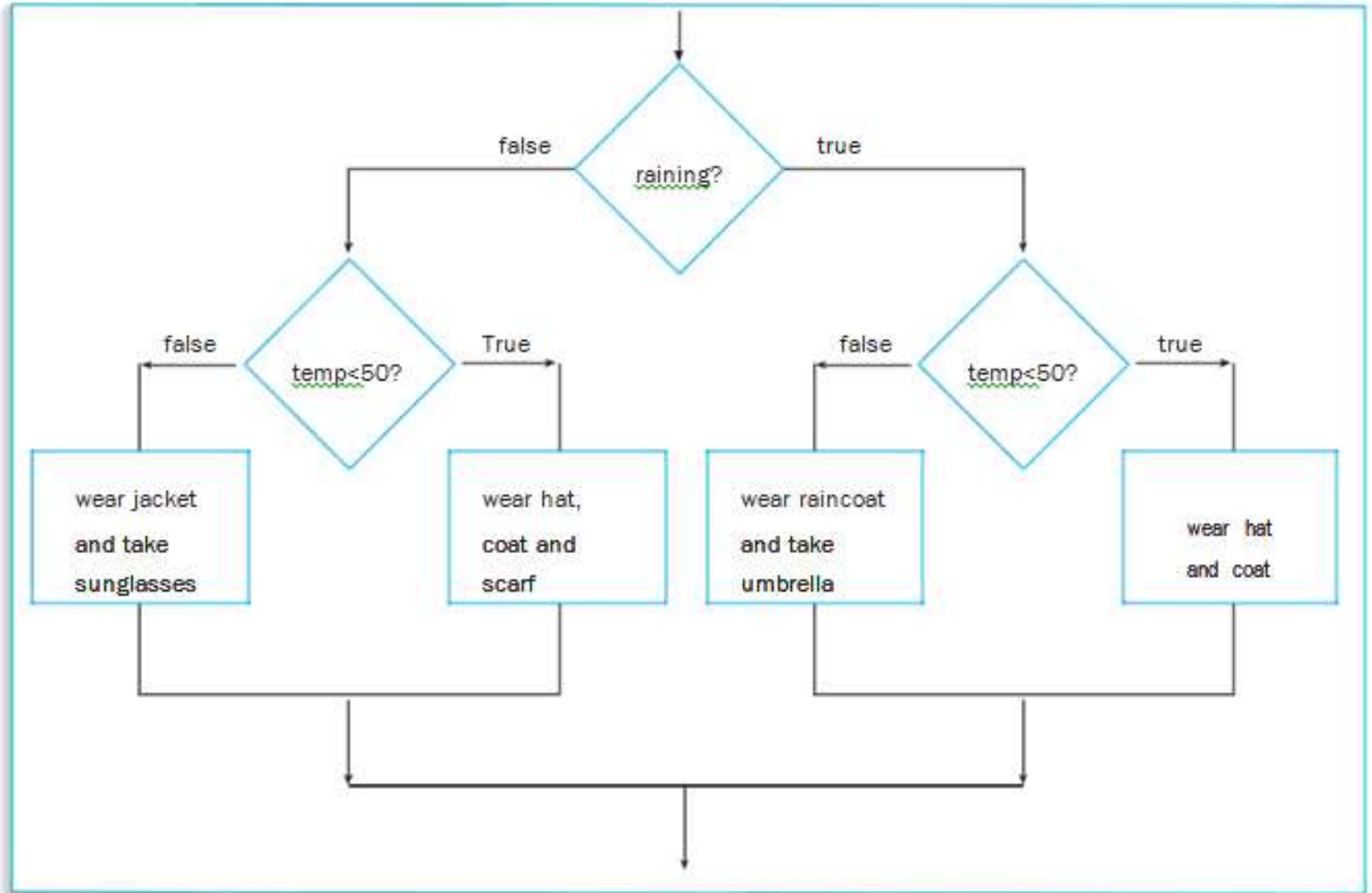


Figure 4.5 Selections within selections, known as nested selections

program to demonstrate the use of nested if .. else statements import avi.*;

```
class Example_5
```

```
{
```

```
public static void main(String[] args)
```

```
{
```

```
String[] buttons = {"DRY", "RAINING"};
```

```
String weather;
```

```
// create and show window
```

```
Window screen = new Window("Example_5.java","bold","blue",18);
```

```
screen.showWindow();
```

```
//create and show radio buttons
```

```
RadioButtons inputWeather = new RadioButtons(screen,"Weather conditions?",buttons);
```

```
inputWeather.showRadioButtons();
```

```
//get selected weather button
```

```
weather = inputWeather.getNameOfButton();
```

```
//create and show slider
```

```
Slider inputTemperature = new Slider(screen,"Temp. degrees F?",40,70,1);
```

```
inputTemperature.showSlider();
```

```
//get temperature
```

```
int temperature = inputTemperature.getValue();
```

```
// write what to wear
```

```
if (weather.equals("RAINING"))
```

```
{
```

```
if (temperature < 50)
```

```
screen.write("It's cold and wet outside, "+ "wear an overcoat and hat.");
```

```
else
```

```
screen.write("The weather is warm and wet, wear "+ "a raincoat and take an umbrella.");
```

```
}
```

```
else
```

```
{
```

```
if (temperature < 50)
```

```
screen.write("It's cold but dry, wear a hat, "+ "coat and scarf.");
```

```
else
```

```
screen.write("The weather is just great, wear a "+"jacket and take sunglasses");
```

```
}
```

```
screen.write("\n\n\nClose the window to exit.");
```

```
}
```

```
}
```

The contents of the log file after running this program follows.

```
=====
                        L O G      F I L E
  audio-visual interface [avi] - Release 1.0 - by Barry Holmes
    filename: Example_5.java   date: 3/12/2000   time: 5:40:50
=====
```

At the prompt: Weather conditions?, you selected [RAINING] from the radio buttons.

At the prompt: Temp. degrees F?, you selected [50] from the slider.

The weather is warm and wet, wear a raincoat and take an umbrella.

Close the window to exit.

Conditional Statements:

program to demonstrate the use of conditional statements

```
import avi.*;
```

```
class Example_6
```

```
{
```

```
public static void main(String[] args)
```

```
{
```

```
String[] genderButtons = {"MALE","FEMALE"};
```

```
String[] bloodButtons = {"A","B","AB","O","not listed"};
```

```
String name, gender, bloodGroup;
```

```
Int age, weight;
```

create and show window pane

```
Window screen = new Window("Example_6.java","bold","blue",18);
```

```
screen.showWindow();
```

create and show dialog box and input name of applicant

```
DialogBox inputName = new DialogBox(screen,"Name?");
```

```
inputName.showDialogBox();
```

```
name = inputName.getString();
```

//create and show radio buttons and input gender

```
RadioButtons inputGender = new RadioButtons(screen,"Gender?",genderButtons);  
inputGender.showRadioButtons();  
gender = inputGender.getNameOfButton();
```

//create and show radio buttons and input blood group

```
RadioButtons inputBloodGroup = new RadioButtons(screen,"Blood Group?",bloodButtons);  
inputBloodGroup.showRadioButtons();  
bloodGroup = inputBloodGroup.getNameOfButton();
```

// create and show slider and input age

```
Slider inputAge = new Slider(screen,"Age in years?",10,80,1);  
inputAge.showSlider();  
age = inputAge.getValue();
```

//create and show slider and input weight

```
Slider inputWeight = new Slider(screen,"Weight in pounds?",50,250,1); inputWeight.showSlider();  
weight = inputWeight.getValue();
```

//analyze applicant for suitability in trial

```
if (gender.equals("FEMALE"))  
    if (bloodGroup.equals("O"))  
        if (age >= 18 && age <= 40)  
            if (weight >= 90 && weight <= 180)  
screen.write(name+" your application "+"for the medical trial "+"was successful.");  
}  
}
```

The contents of the log file after running this program follows.

=====

L O G F I L E

audio-visual interface [avi] - Release 1.0 - by Barry Holmes

filename: Example_6.java date: 3/12/2000 time: 5:42:59

=====

At the prompt: Name? you input [Lucy Lockett] at the dialog box.

At the prompt: Gender? you selected [FEMALE] from the radio buttons.

At the prompt: Blood Group? you selected [O] from the radio buttons.

At the prompt: Age in years? you selected [32] from the slider.

At the prompt: Weight in pounds? you selected [125] from the slider.

Lucy Lockett your application for the medical trial was successful.

Else If Statements:

- The complexity of nested if statements can be reduced by combining conditions and using logical AND.

```
if (buttons[position].equals("RAINING"))  
{  
  if (temperature < 50)  
    screen.write("It's cold and wet outside, wear an overcoat "+ "and hat.");  
  else  
    screen.write("The weather is warm and wet, wear a "+ "raincoat and take an umbrella.");  
}  
else  
{  
  if (temperature < 50)  
    screen.write("It's cold but dry, wear a hat, coat and scarf.");  
  else  
    screen.write("The weather is just great, wear a jacket "+ "and take sunglasses");  
}
```

Else If Statements:

Condition X	Condition Y	X && Y
false	false	false
false	true	false
true	false	false
true	true	true

can be recoded as

Figure 4.7 Truth table for logical AND

```
if(buttons[position].equals("RAINING")) && (temperature < 50)
screen.write("It's cold and wet outside, "+ "wear an overcoat and hat.");
```

else

```
if (buttons[position].equals("RAINING")) && (temperature >= 50)
screen.write("The weather is warm and wet, "+ "wear a raincoat and take an umbrella.");
```

else

```
if (buttons[position].equals("DRY")) && (temperature < 50)
screen.write("It's cold but dry, wear a hat, coat and scarf.");
```

else

```
screen.write("The weather is just great, wear a jacket "+ "and take sunglasses");
```


Boolean Data Types:

initialize boolean flag to trap any unsuitable criteria

```
boolean reject = false;
```

analyze applicant for suitability in trial

```
if(!gender.equals("FEMALE"))
```

```
{  
  reject = true;  
}
```

```
else
```

```
{  
  if (!bloodGroup.equals("O"))  
    reject = true;
```

```
}
```

```
else
```

```
{  
  if (age < 18 || age > 40)
```

```
    reject = true;
```

```
}
```

```
else
```

```
{  
  if (weight < 90 || weight > 180)
```

```
    reject = true;
```

```
}
```

```
else
```

```
screen.write(name+" your profile for the medical trial is "+"acceptable.");
```

```
if (reject) screen.write(name+" your application for the medical trial was "+"NOT successful.");
```

- A variable of primitive type boolean is permitted to have only **one of two values, either true or false.**

Boolean Data Types:

Condition X	Condition Y	X Y
false	false	false
false	true	true
true	false	true
true	true	true

Figure 4.8 Truth table for logical OR

The conditions for gender, blood group, age, and weight can also be combined into

```
(! gender.equals("FEMALE") || ! bloodGroup.equals("O") || (age < 18 || age > 40) ||  
(weight < 90 || weight > 180))
```

Switch:

SYNTAX

```
Switch Statement:  switch (expression)
                   {
                       case c1: statement(s);
                       case c2: statement(s);
                       .
                       .
                       default: statement(s);
                   }
```

- An ordinal variable has a value that belongs to an ordered set of items.
- For example, integers are ordinal types since they belong to the set of values from 22,147,483,648 to +2,147,483,647.
- The expression must evaluate to an ordinal value.
- Each possible ordinal value is represented as a case label that indicates the statement to be executed corresponding to the value of the expression.
- Those values that are not represented by case labels will result in the statement after the optional default being executed.

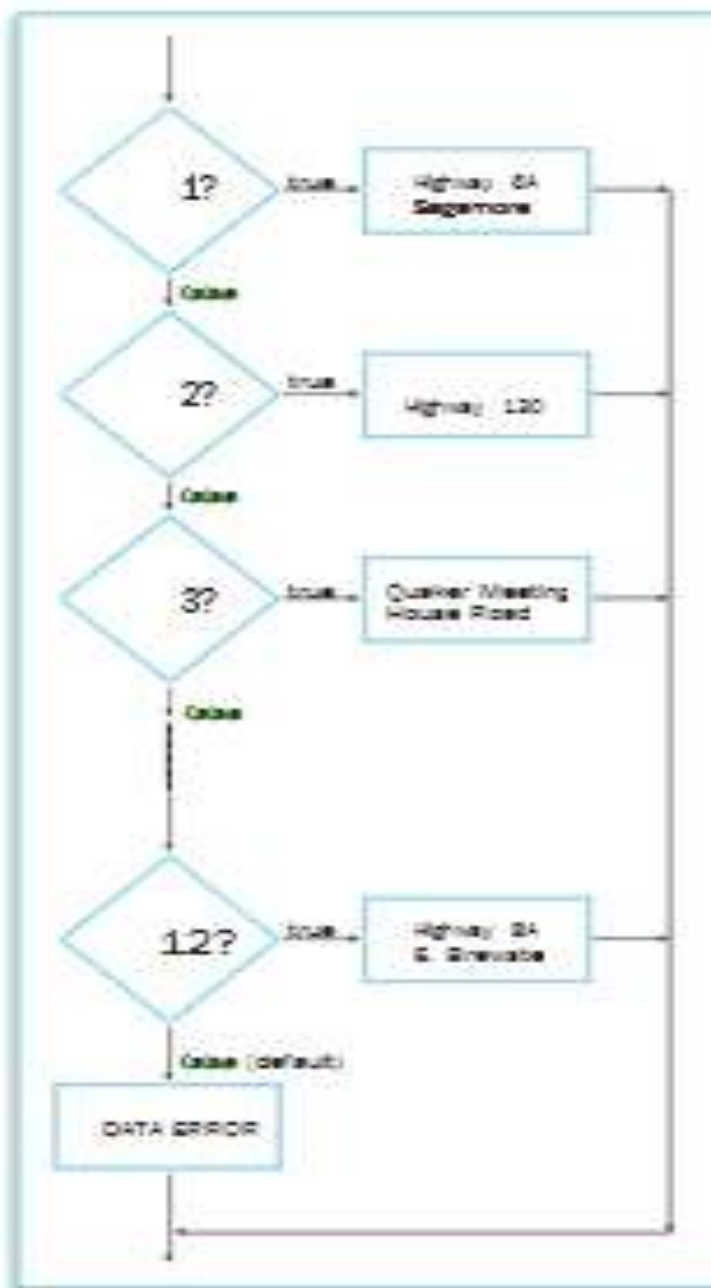
Switch:

```
int number = inputData.getInteger(); // input from a dialog box  
switch (number)
```

```
{  
case 1:   screen.write("one"); break;  
  
case 2:   screen.write("two"); break;  
  
case 3:   screen.write("three"); break;  
  
default: screen.write("number not in the range 1..3");  
}
```

- One method of exiting from a switch statement is through the use of a *break* statement at the end of every case list.
- If the optional default statement was not present and the value of number had not been in the range 1 to 3, then the computer would branch to the end of the switch statement.

Switch:



//program to demonstrate the switch statement

```
import avi.*;
```

```
class Example_7
```

```
{
```

```
public static void main(String[] args)
```

```
{
```

// create and show window

```
Window screen = new Window("Example_7.java","bold","blue",18); s
```

```
screen.showWindow();
```

create and show dialog box and input exit number

```
DialogBox inputExitNumber = new DialogBox(screen,"Exit number on Highway 6?");
```

```
inputExitNumber.showDialogBox();
```

```
int exitNumber = inputExitNumber.getInteger();
```

```
screen.write("Exit "+exitNumber+" on Highway 6 connects with ");
```

select on the value of the exit number switch (exitNumber)

```
{
```

```
case 1:  screen.write("Highway 6a/ Sagamore Bridge"); break;
case 2:  screen.write("Highway 130"); break;
case 3:  screen.write("Quaker Meeting House Road"); break;
case 4:  screen.write("Chase Road/ Scorten Road"); break;
case 5:  screen.write("Highway 149/ Martons Mills"); break;
case 6:  screen.write("Highway 132/ Hyannis"); break;
case 7:  screen.write("Willow Street/ Higgins Crowell Road"); break;
case 8:  screen.write("Union Street/ Station Avenue"); break;
case 9:  screen.write("Highway 134/ S.Dennis"); break;
case 10: screen.write("Highway 124/ Harwich Port"); break;
case 11: screen.write("Highway 137/ S.Chatham"); break;
case 12: screen.write("Highway 6A/ E.Brewster"); break;
default: screen.write("DATA ERROR - incorrect exit number"); Audio.beep(screen);
}

}

}
```

```
=====
                L O G      F I L E
audio-visual interface [avi] - Release 1.0 - by Barry Holmes filename:
      Example_7.java    date: 3/12/2000   time: 5:48:19
=====
```

===

At the prompt: Exit number on Highway 62, you input [7] at the dialog box.

Exit 7 on Highway 6 connects with Willow Street/ Higgins Crowell Road

```
=====
                L O G      F I L E
audio-visual interface [avi] - Release 1.0 - by Barry Holmes filename:
      Example_7.java    date: 3/12/2000   time: 5:49:45
=====
```

===

At the prompt: Exit number on Highway 62, you input [13] at the dialog box.

Exit 13 on Highway 6 connects with DATA ERROR - incorrect exit number

Switch:

```
switch (month)
{
//test for months Jan, Mar, May, Jul, Aug, Oct, Dec
case 1:
case 3:
case 5:
case 7:
case 8:
case 10:
case 12: daysInMonth = 31; break;

//test for months Apr, Jun, Sep, Nov
case 4:
case 6:
case 9:
case 11: daysInMonth = 30; break;

//test for Feb (or use default)
case 2: daysInMonth = 28;
}
```

The Wrapper Class:

- All the **primitive types** have corresponding classes that provide **some general methods** that are useful when dealing with data of the specified type.
- The classes are known as *wrapper* classes since they **literally wrap the primitive data type** in a class.

//A partial listing of the Integer class follows.

```
public final class Integer
{
    public static final int MIN_VALUE = 0x80000000;
    public static final int MAX_VALUE = 0x7fffffff; ..
```

// constructors

```
public Integer(int value);
public Integer(String s);
instance method public int intValue();
.
.
}
```

The Wrapper Class:

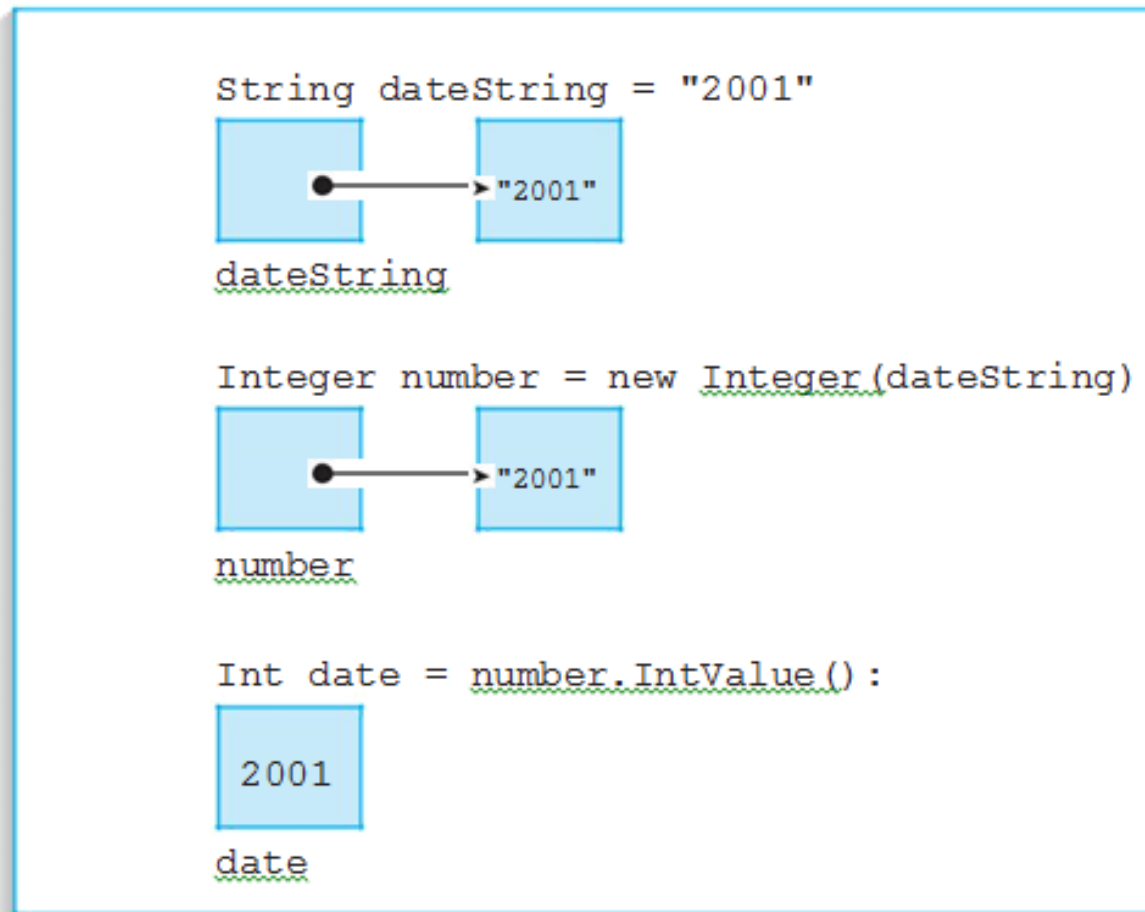


Figure 4.10 Use of the Integer wrapper class

The Memo Class:

The Memo class contains the following constructor and public instance methods:

```
public class Memo
{
    public Memo(Window parent,
                String firstLine,
                String secondLine,
                String thirdLine,
                boolean mode);

    public void showMemo();
    public void hideMemo();
}
```

- To create a Memo object, you must use the class constructor that requires five items of data in the formal parameter list:
- **parent**—a Window type that specifies the **container** on which to display a Memo object.
- **firstLine, secondLine, and thirdLine**—string types that allow for **textual information to be displayed** by the Memo object. You do not have to fill every line; it is permissible to use an empty string ("") where required.
- **mode**—a boolean type that specifies whether the **memo is to remain** on the screen until the memo's window is closed. If the mode is set to true, the only way to advance to the next executable statement in the program is by closing the memo's window. If the mode is set to false, then the computer will advance to the next executable program instruction without waiting for you to even read what is contained in the memo object.

The Memo Class:

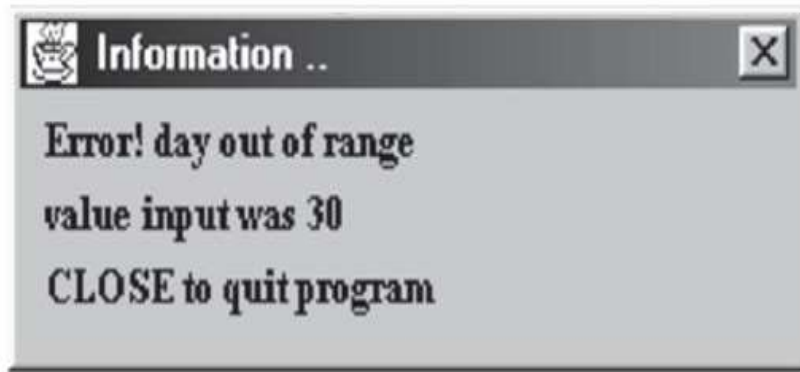


Figure 4.14 An example of a Memo object

the Memo constructor may be coded as:

```
Memo message = new Memo(screen,  
    "Error! day out of range",  
    "value input was "+day,  
    "CLOSE to quit program",  
    true);
```

The memo is displayed on the screen using the instance method `showMemo()`, for example:

```
message.showMemo();
```

The memo may be hidden from view using the instance method `hideMemo()`.

The This Object:

- The `this` keyword may be used within a class to refer to any object of that class type.
- Can use within a class, when you want to invoke an instance method of that class when the object is declared outside of the class.