CORE JAVA

INTRODUCTION TO OOPS

- Object oriented programming combines data and instruction into an object which can be further used to assemble larger programs.
- Objects are modelled on real world entities.

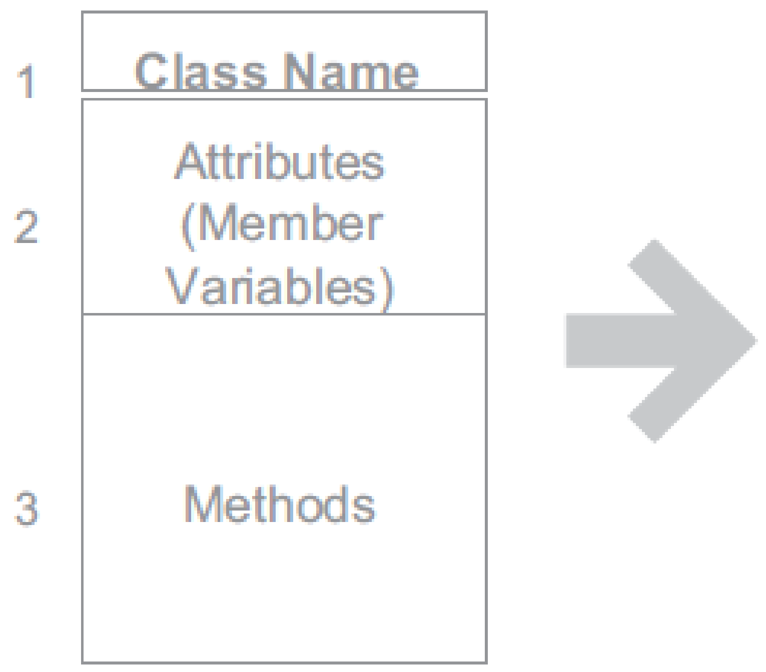
PROGRAMMING TECHNIQUES

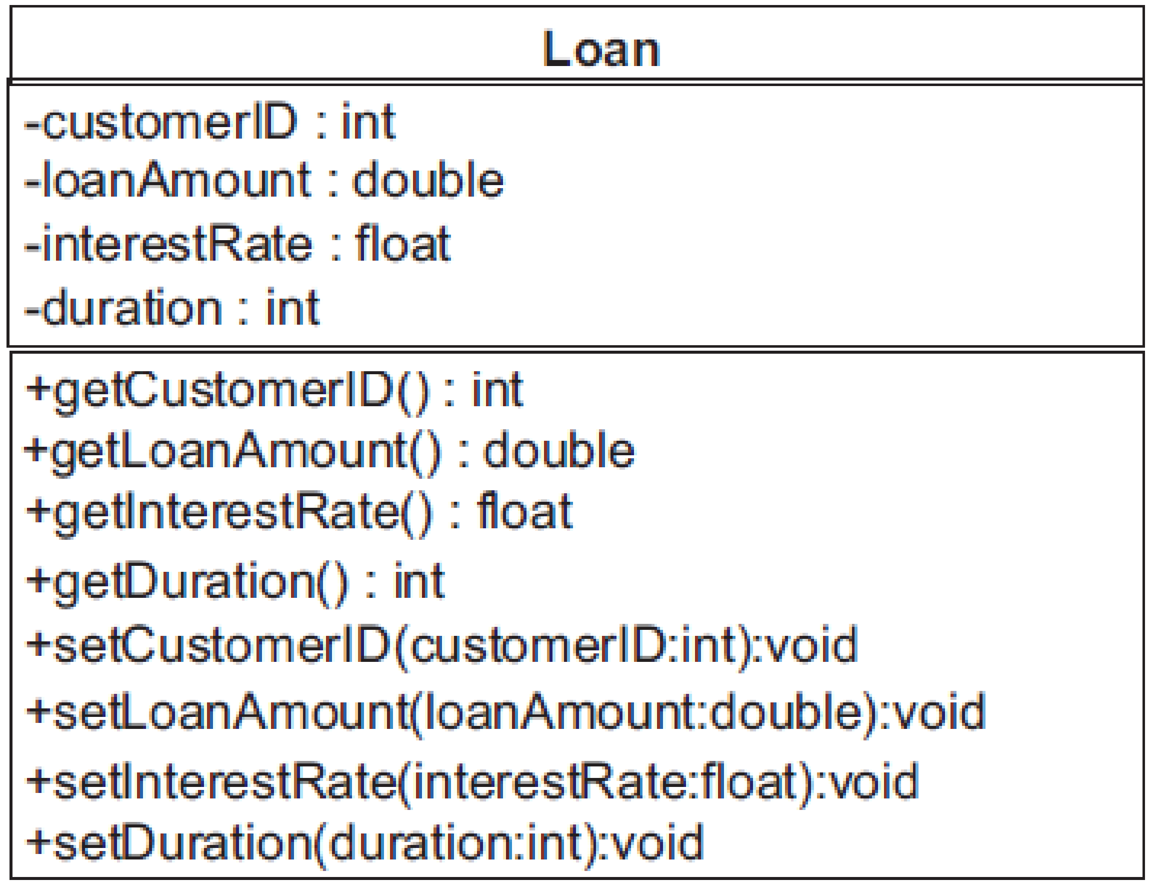
- UNSTRUCTURED PROGRAMMING
- PROCEDURAL PROGRAMMING
- OBJECT ORIENTED PROGRAMMING

OBJECT ORIENTED PROCESS

- **OBJECT ORIENTED ANALYSIS-** requirements are mapped to the perspective of classes and objects suiting the domain of requirements
- **OBJECT ORIENTED DESIGN-** representation of the process to be developed and providing the notations for the system under design. Creating diagrams like usecase diagram, class diagram, activity diagram, sequence diagram,etc
- **OBJECT ORIENTED PROGRAMMING-** implementation of object oriented concepts.

Reprasentation of a class




Loan
-customerID : int
-loanAmount : double
-interestRate : float
-duration : int
+getCustomerID() : int
+getLoanAmount() : double
+getInterestRate() : float
+getDuration() : int
+setCustomerID(customerID:int):void
+setLoanAmount(loanAmount:double):void
+setInterestRate(interestRate:float):void
+setDuration(duration:int):void

OBJECT

- A bundle of related variables and functions.
- Has a state and behavior

Characteristics of object

- ABSTRACTION- hiding of irrelevant information
- ENCAPSULATION- object encapsulates data and instructions. It exposes its behavior from the methods that it implements.
- MESSAGE PASSING- objects interact with each other with the help of methods.

Class

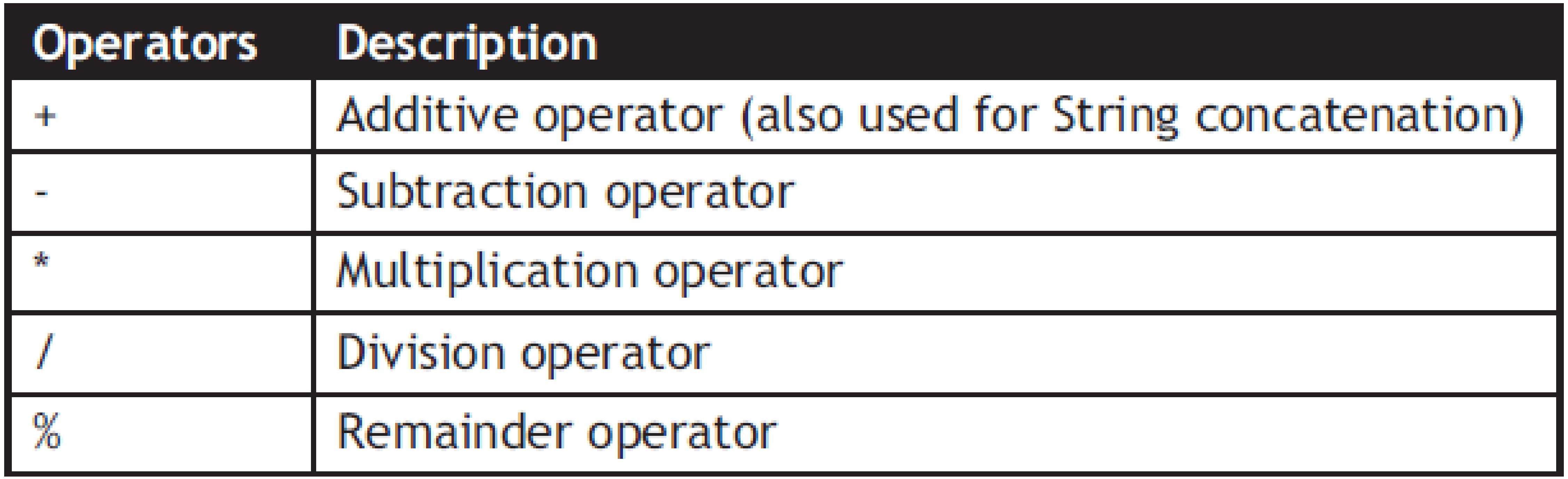
- An abstract data type.
- A blue print of an object
- Constitutes of member variables and member functions

Access Specifiers

- PUBLIC- accessible to all
- PRIVATE- accessible to only members of the same class.
- PROTECTED- accessible to the class and the ones inheriting the class

Operators

- Arithmetic Operators



Operators	Description
+	Additive operator (also used for String concatenation)
-	Subtraction operator
*	Multiplication operator
/	Division operator
%	Remainder operator

Relational Operators

Operators	Description
==	Equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
!=	Not equal to

Logical Operators

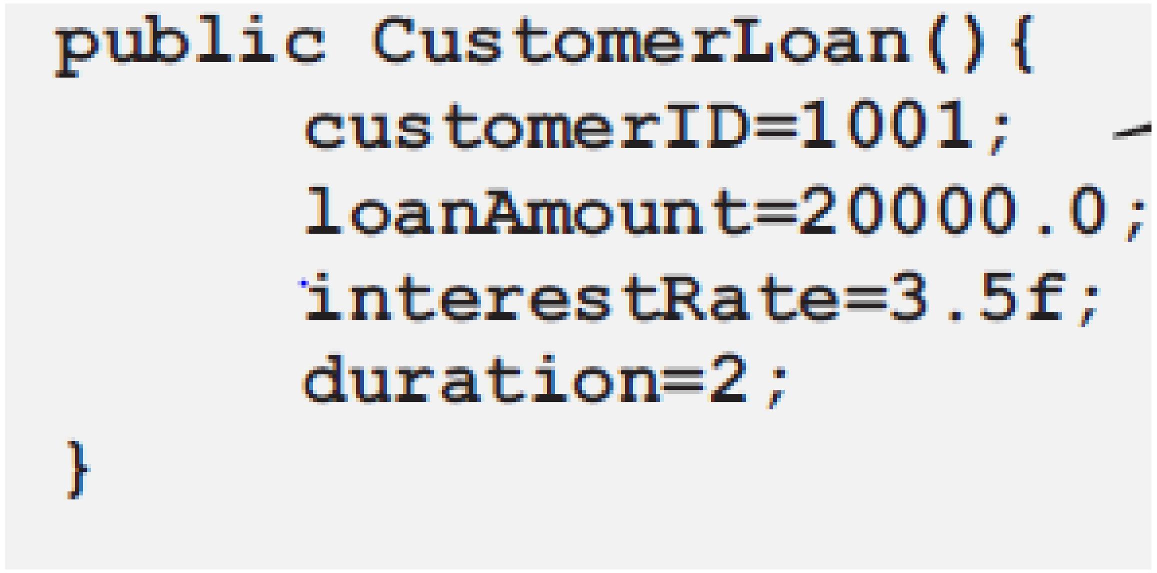
Operator	Meaning
&&	Short Circuit-AND
||	Short Circuit-OR
&	Logical AND
|	Logical OR
!	Logical compliment operator

Constructor

- Constructor is a specialized member of a class.
- It has same name as class
- It is used to initialize the member variables along with creation of an object.

Default Constructor

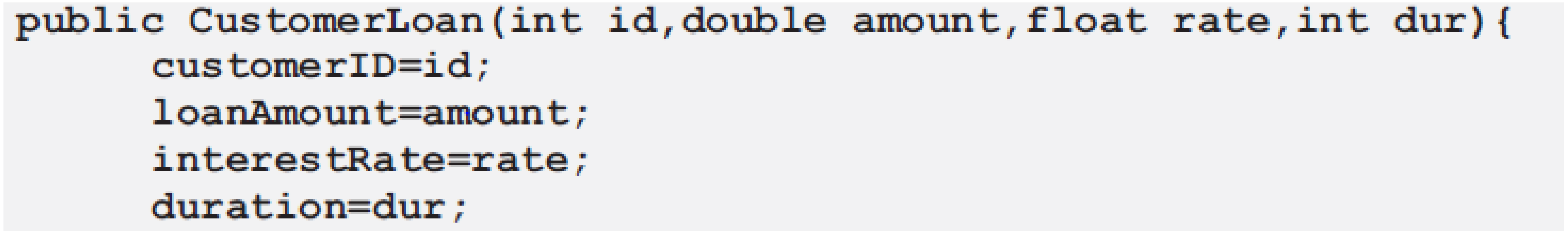
- A default constructor has no parameters.
- The instance variables are assigned with default values.
- If the default constructor is redefined,instance variables can be initialized explicitly inside the constructor.



```
public CustomerLoan(){
        customerID=1001;
        loanAmount=20000.0;
        interestRate=3.5f;
        duration=2;
}
```

Parameterized constructors

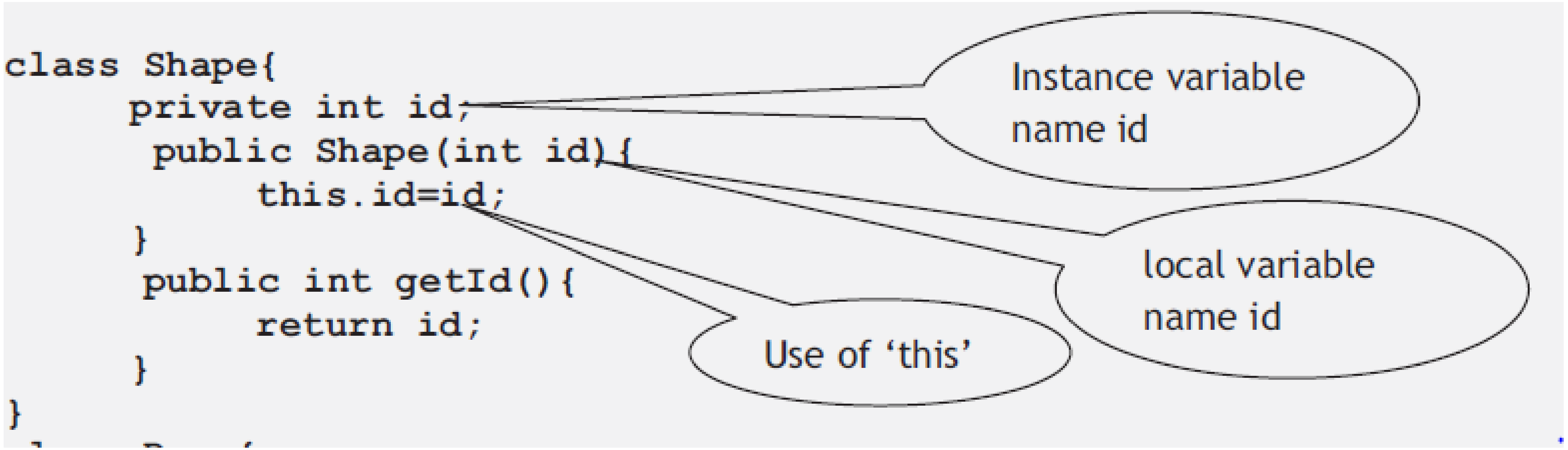
- Constructors that take arguments.
- Using parameterized constructors we can initialize instance variables during object creation



```
public CustomerLoan(int id,double amount,float rate,int dur){
      customerID=id;
      loanAmount=amount;
      interestRate=rate;
      duration=dur;
```

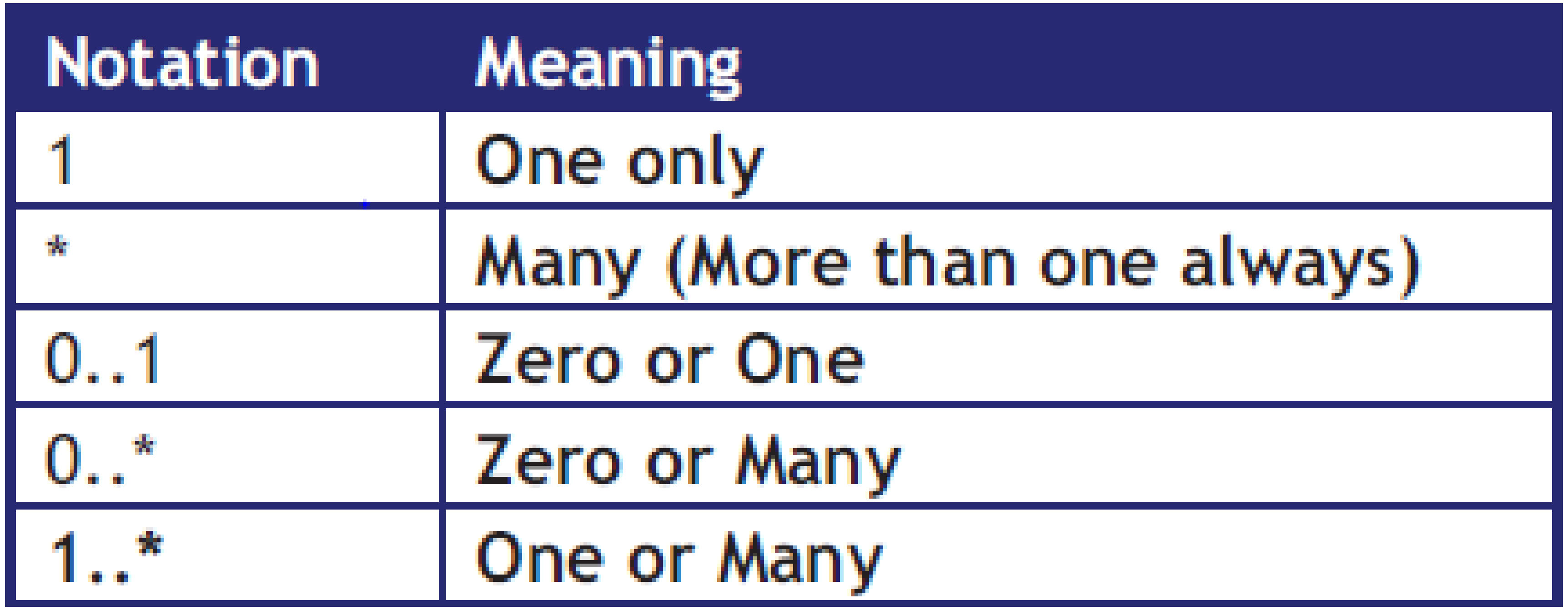
‘THIS’ Keyword

- Prevents instance variable hiding.
- Used where instance variable and local variables have the same names.

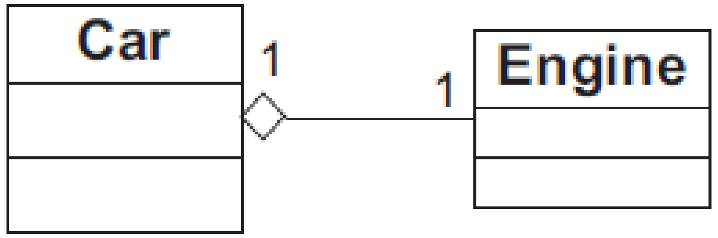
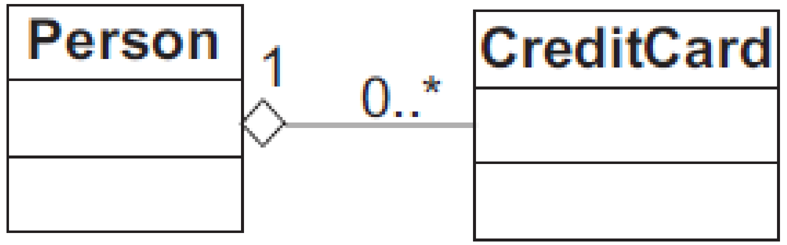
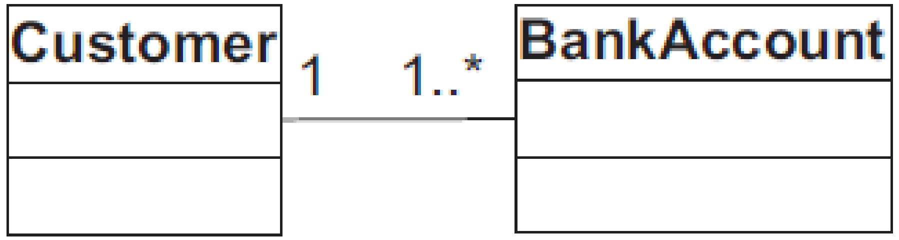


```
class Shape{
     private int id;
      public Shape(int id){
          this.id=id;
     }
     public int getId(){
          return id;
     }
}
```

Multiplicity of relationships



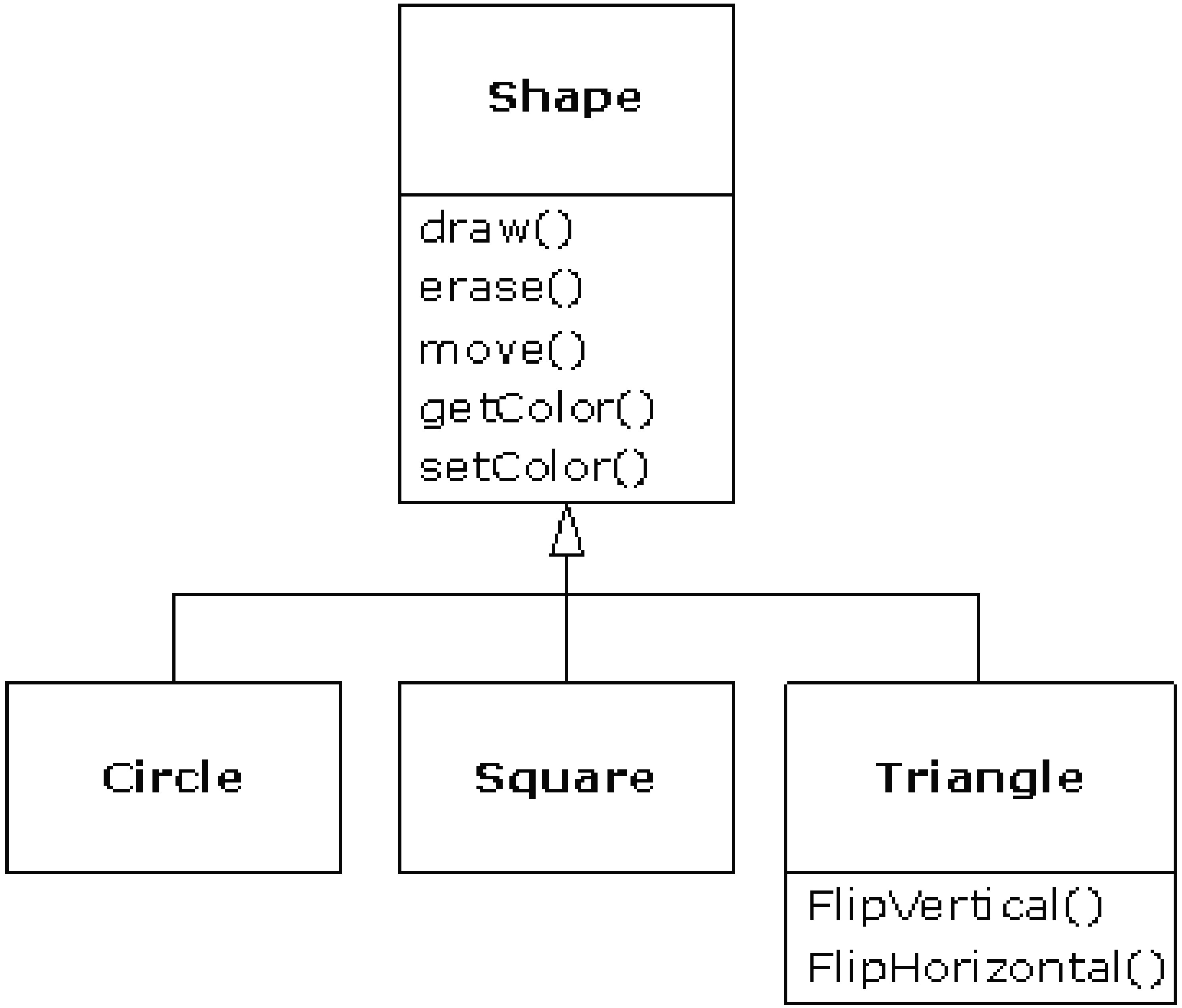
Notation	Meaning
1	One only
*	Many (More than one always)
0..1	Zero or One
0..*	Zero or Many
1..*	One or Many

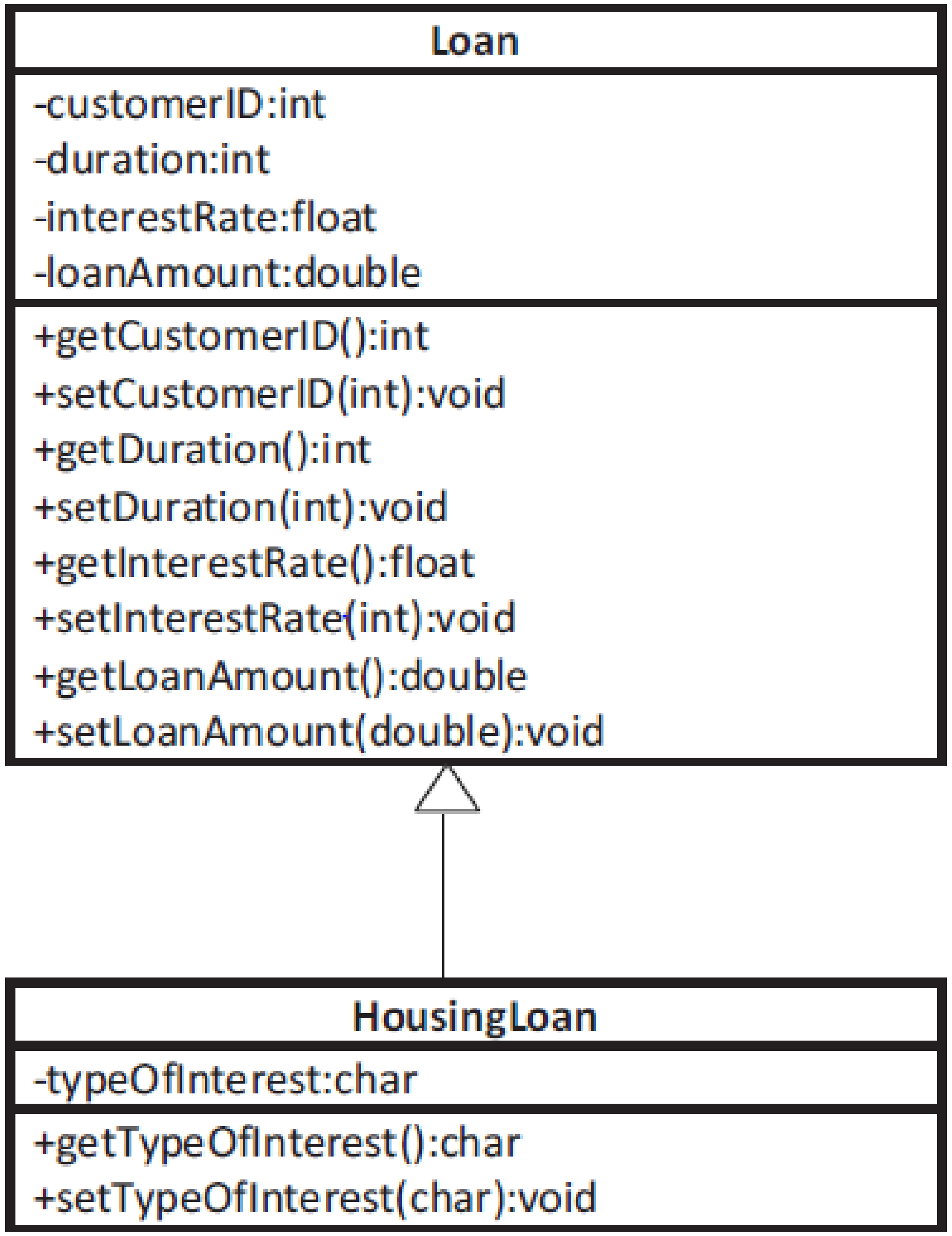
Multiplicity	Representation
One to One Aggregation *A car can have only one engine*	
One to Many Aggregation (Many = zero or more) *A person can have zero or more credit cards.*	
One to Many Association (Many = one or more) *In a bank, a customer can use one or more accounts.*	

INHERITANCE

What is inheritance in Object Oriented Technology? Inheritance is a mechanism of defining a new class based on an existing class. Inheritance enables reuse of code. Inheritance also provides scope for refinement of the existing class. Given a class, using inheritance one can define a new class which refines it.

The existing (or original) class is called the **base** class or **super** class or **parent** class.


Shape
draw()
erase()
move()
getColor()
setColor()
Circle
Square
Triangle
FlipVertical()
FlipHorizontal()


Loan
-customerID:int
-duration:int
-interestRate:float
-loanAmount:double
+getCustomerID():int
+setCustomerID(int):void
+getDuration():int
+setDuration(int):void
+getInterestRate():float
+setInterestRate(int):void
+getLoanAmount():double
+setLoanAmount(double):void
HousingLoan
-typeOfInterest:char
+getTypeOfInterest():char
+setTypeOfInterest(char):void

```
import java.io.*;
class Customer{
  private int id;
  private String name;
  public Customer(int cid,String cname){
        id=cid;
        name=cname; }
  public int getid(){
        return id;   }
  public String getname(){
        return name;
  }
}
```

```
class RegularCustomer extends Customer{
  private int discount;
  public RegularCustomer(int id, String name, int discount){
        super(id,name);
        this.discount=discount;
  }
  public int getdiscount(){
        return discount;
  }
}
```

```
class Inheritance{
  public static void main(String args[]){
        Customer cus1=new Customer(100,"John");
        RegularCustomer cus2= new RegularCustomer(200,"Tim",10);
        System.out.println("Customer Id is"+cus1.getid());
        System.out.println("Customer Name is"+cus1.getname());
        System.out.println("Customer2 Id is"+cus2.getid());
        System.out.println("Customer2 Name is"+cus2.getname());
        System.out.println("Customer2 Discount is"+cus2.getdiscount());
  }
}
```

INTERFACE

An interface is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface.

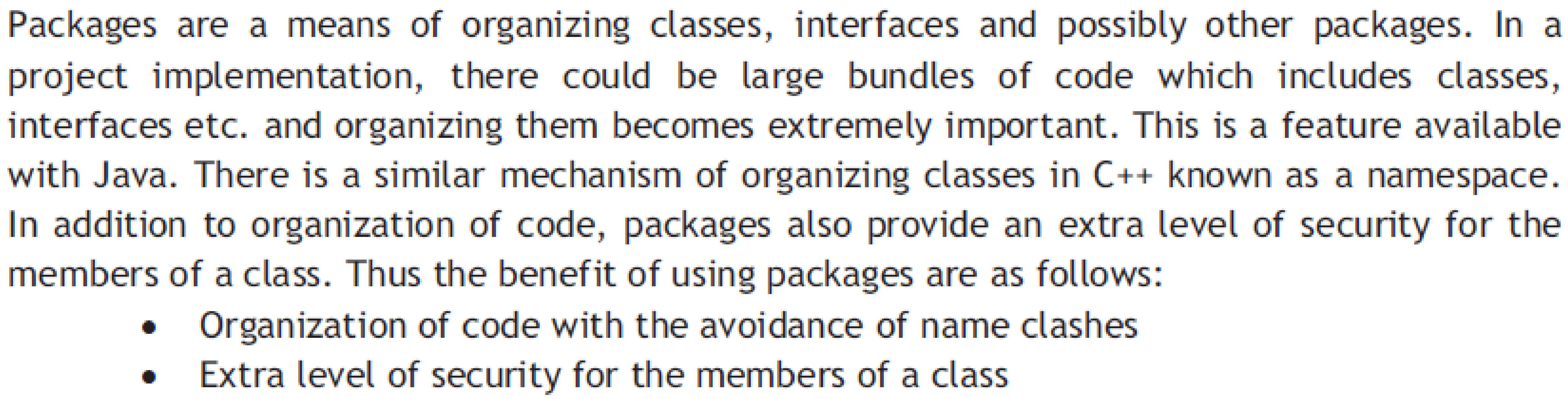
An interface is not a class. Writing an interface is similar to writing a class, but they are two different concepts.

A class describes the attributes and behaviors of an object. An interface contains behaviors that a class implements.

An interface is different from a class in several ways, including:

- You cannot instantiate an interface.
- An interface does not contain any constructors.
- All of the methods in an interface are abstract.
- An interface is not extended by a class; it is implemented by a class.

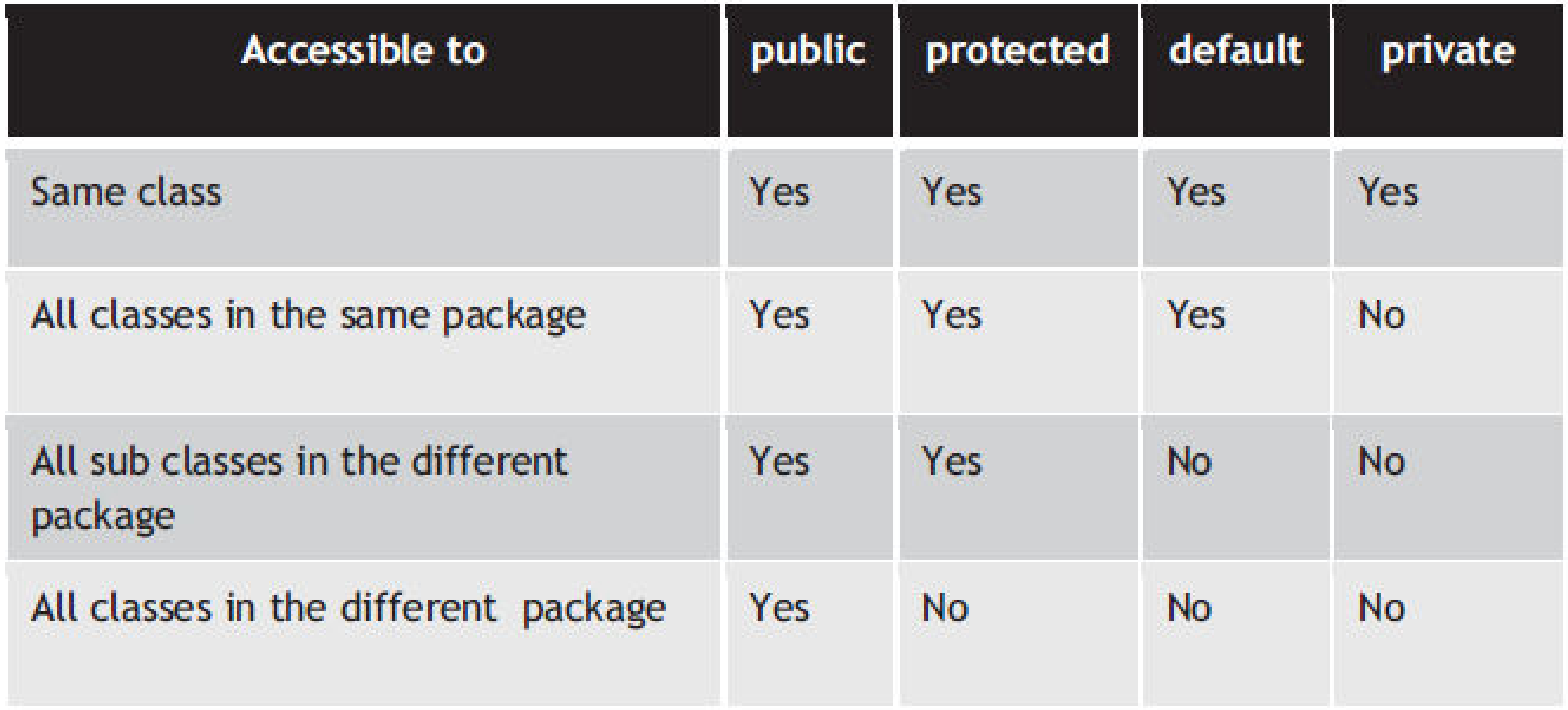
PACKAGE



Packages are a means of organizing classes, interfaces and possibly other packages. In a project implementation, there could be large bundles of code which includes classes, interfaces etc. and organizing them becomes extremely important. This is a feature available with Java. There is a similar mechanism of organizing classes in C++ known as a namespace. In addition to organization of code, packages also provide an extra level of security for the members of a class. Thus the benefit of using packages are as follows:

- Organization of code with the avoidance of name clashes
- Extra level of security for the members of a class

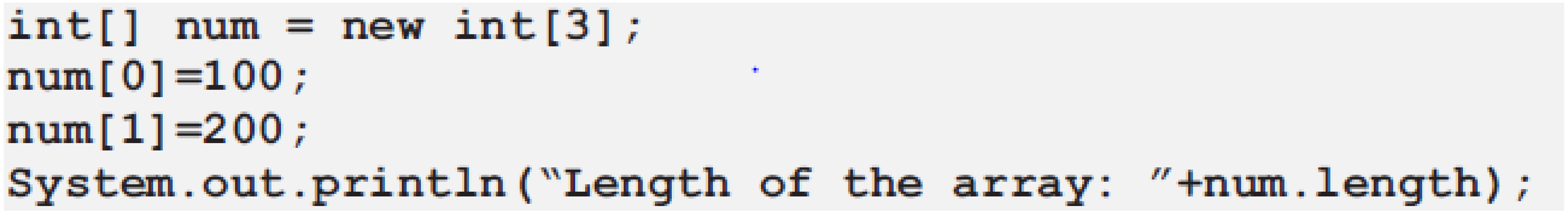
ACCESS SPECIFIERS



Accessible to	public	protected	default	private
Same class	Yes	Yes	Yes	Yes
All classes in the same package	Yes	Yes	Yes	No
All sub classes in the different package	Yes	Yes	No	No
All classes in the different package	Yes	No	No	No

Array

- A derived datatype which is used to store values of primitive datatypes.
- Is created using new operator



```
int[] num = new int[3];
num[0]=100;
num[1]=200;
System.out.println(“Length of the array: ”+num.length);
```

```
import java.io.*;


class Array{
public static void main(String args[]){
int[] A={1, 2, 3, 3};
      // Print all the array elements
      for (int i = 0; i < A.length; i++) {
         System.out.println(A[i] + " ");
      }
  }
}
```

Exception Handling

- An exception (or exceptional event) is a problem that arises during the **execution** of a program.
- When an Exception occurs the normal flow of the program is disrupted and the program/Application terminates abnormally, which is not recommended, therefore these exceptions are to be handled.
- Exceptions are caused by user error, others by programmer error, and others by physical resources that have failed in some manner.

Checked exceptions

- A checked exception is an exception that occurs at the compile time, these are also called as compile time exceptions.

- These exceptions cannot simply be ignored at the time of compilation, the Programmer should take care of (handle) these exceptions.

- Example: *FileNotFoundException*

Unchecked exceptions

- An Unchecked exception is an exception that occurs at the time of execution.

- These are also called as Runtime Exceptions.

These include programming bugs, such as logic errors.

Runtime exceptions are ignored at the time of compilation.

- For example, if you have declared an array of size 5 in your program, and trying to call the 6th element of the array then an *ArrayIndexOutOfBoundsExceptionexception* occurs.

```
public class Unchecked_Demo {
        public static void main(String args[]){
                int num[]={1,2,3,4};
                System.out.println(num[5]);
        }
}
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException:
 5 at Exceptions.Unchecked_Demo.main(Unchecked_Demo.java:8)
```

Catching Exceptions

- A method catches an exception using a combination of the **try** and **catch** keywords.
- A try/catch block is placed around the code that might generate an exception.
- Code within a try/catch block is referred to as protected code.

```
try {
     //Protected code
 }
catch(ExceptionName e1){
     //Catch block
 }
```

The code which is prone to exceptions is placed in the try block, when an exception occurs, that exception occurred is handled by catch block associated with it.

```
import java.io.*;
public class ExcepTest{
        public static void main(String args[]){
                try{
                   int a[] = new int[2];
                   System.out.println("Access element three :" + a[3]);
                 }
                 catch(ArrayIndexOutOfBoundsException e){
                         System.out.println("Exception thrown :" + e);
                   }
                   System.out.println("Out of the block");
}
}
```

```
Exception thrown :java.lang.ArrayIndexOutOfBoundsException: 3 Out of the block
```

Thread

- A thread, in the context of Java, is the path followed when executing a program.

- All Java programs have at least one thread, known as the main thread, which is created at the start of a program, when the main method is invoked.

- Every thread in Java is controlled by the java.lang.Thread class.

- When a thread is created, it is assigned a priority.

- The thread with higher priority is executed first.
- Thread is an independent path of execution through program code.
- A thread itself is not a program, but runs within a program.
- Example, a multithreaded Web Browser allows one to scroll a page while some image is being downloaded or an animation is being played.
- A thread carves out some of its own resources within a running program. It must have its own execution stack and program counter.

Multithread programming

- Java is a multi threaded programming language that allows multiple thread execution at any particular time.

- In a single threaded application, only one thread is executed at a time.

- A multithreaded program contains two or parts that can run concurrently and each part can handle different task at the same time, making optimal use of the available resources specially when your computer has multiple CPUs.

Example: a single-threaded application may allow for the typing of words. However, the single thread requires an additional single thread allowing for recording of keystrokes in order to type the words.

Thus, a single-threaded application records the keystrokes, allowing the next single-threaded application(typing of words) to follow.

However a multi-threaded application allows for the handling of both tasks(recording and typing of keystrokes) within one application.

JAVA VIRTUAL MACHINE

- It is a specification that provides runtime environment in which java bytecode can be executed..

- It is platform dependent and is available for many hardware and software platforms.

JVM OPERATIONS

- Loads code
- Verifies Code
- Executes code
- Provides a runtime environment

I/O

- Java.io package contains all the classes that are required to perform input and output in java.
- Java uses the concept of stream to make I/O operation fast.

Stream

- A stream is a sequence of data.
- In java, stream is composed of bytes.
- Types of streams: System.out(standard output stream)
  System.in (standard input stream)
  System.err(standard error stream)

I/O STREAM

- The Java development environment includes a package, java.io, that contains a set of input and output streams that your programs can use to read and write data.

- The InputStream and OutputStream classes in java.io are the abstract super classes that define the behavior for sequential input and output streams in Java.

- Also included in java.io are several InputStream and OutputStream subclasses that implement specific types of input and output streams.

Java - Applet

- An applet is a Java program that runs in a Web browser.
- An applet is a Java class that extends the java.applet.Applet class.
- A main() method is not invoked on an applet, and an applet class will not define main().
- Applets are designed to be embedded within an HTML page.

- When a user views an HTML page that contains an applet, the code for the applet is downloaded to the user's machine.

- A JVM is required to view an applet. The JVM can be either a plug-in of the Web browser or a separate runtime environment.

- The JVM on the user's machine creates an instance of the applet class and invokes various methods during the applet's lifetime.

- Applets have strict security rules that are enforced by the Web browser. The security of an applet is often referred to as sandbox security, comparing the applet to a child playing in a sandbox with various rules that must be followed.

Life Cycle of an Applet

- **init:** This method is intended for whatever initialization is needed for your applet. It is called after the param tags inside the applet tag have been processed.

- **start:** This method is automatically called after the browser calls the init method. It is also called whenever the user returns to the page containing the applet after having gone off to other pages.

- **stop:** This method is automatically called when the user moves off the page on which the applet sits. It can, therefore, be called repeatedly in the same applet.

- **destroy:** This method is only called when the browser shuts down normally. Because applets are meant to live on an HTML page, you should not normally leave resources behind after a user leaves the page that contains the applet.

- **paint:** Invoked immediately after the start() method, and also any time the applet needs to repaint itself in the browser. The paint() method is actually inherited from the java.awt.

APPLET

```
import java.applet.*;
import java.awt.*;
 public class HelloWorldApplet extends Applet {
 public void paint (Graphics g) {
g.drawString ("Hello World", 25, 50);
}
 }
```

The Applet CLASS

Every applet is an extension of the *java.applet.Applet class*. The base Applet class provides methods that a derived Applet class may call to obtain information and services from the browser context.

These include methods that do the following:

- Get applet parameters
- Get the network location of the HTML file that contains the applet
- Get the network location of the applet class directory
- Print a status message in the browser
- Fetch an image
- Fetch an audio clip
- Play an audio clip
- Resize the applet

Additionally, the Applet class provides an interface by which the viewer or browser obtains information about the applet and controls the applet's execution. The viewer may:

- request information about the author, version and copyright of the applet
- request a description of the parameters the applet recognizes
- initialize the applet
- destroy the applet
- start the applet's execution
- stop the applet's execution

The Applet class provides default implementations of each of these methods. Those implementations may be overridden as necessary.

Invoking an Applet

- An applet may be invoked by embedding directives in an HTML file and viewing the file through an applet viewer or Java-enabled browser.

- The <applet> tag is the basis for embedding an applet in an HTML file.

Below is an example that invokes the "Hello, World" applet:

```
<html>
 <title>The Hello, World Applet</title>
 <hr>
<applet code="HelloWorldApplet.class" width="320"
  height="120"> If your browser was Java-enabled, a "Hello,
  World" message would appear here. </applet>
 <hr>
</html>
```

String Handling

String class is encapsulated under java.lang package. In java, every string that you create is actually an object of type **String**.

Creating a String object

String can be created in number of ways, here are a few ways of creating string object.

1) Using a String literal

String literal is a simple string enclosed in double quotes " ". A string literal is treated as a String object.

```
String str1 = "Hello";
```

2) Using another String object

String str2 = new String(str1);

3) Using new Keyword

String str3 = new String("Java");

4) Using + operator (Concatenation)

String str4 = str1 + str2; or, String str5 = "hello"+"Java";

Concatenating String

There are 2 methods to concatenate two or more string.

- Using **concat()** method
- Using + operator

1) Using concat() method

string s = "Hello"; string str = "Java"; string str2 = s.concat(str); String str1 = "Hello".concat("Java"); //works with string literals too.

2) Using + operator

string str = "Rahul"; string str1 = "Dravid"; string str2 = str + str1; string st = "Rahul"+"Dravid";

Event Handling

- Any program that uses GUI (graphical user interface) such as Java application written for windows, is event driven.

- Event describes the change of state of any object.

- **Example :** Pressing a button, Entering a character in Textbox.

Components of Event Handling

Event handling has three main components,

- **Events :** An event is a change of state of an object.
- **Events Source :** Event source is an object that generates an event.
- **Listeners :** A listener is an object that listens to the event. A listener gets notified when an event occurs.

How Events are handled ?

- A source generates an Event and send it to one or more listeners registered with the source.

- Once event is received by the listener, they processe the event and then return.

- Events are supported by a number of Java packages, like **java.util**, **java.awt** and **java.awt.event**.

Event Classe	Description	Listener Interface
ActionEvent	generated when button is pressed, menu-item is selected, list-item is double clicked	ActionListener
MouseEvent	generated when mouse is dragged, moved,clicked,pressed or released also when the enters or exit a component	MouseListener
KeyEvent	generated when input is received from keyboard	KeyListener
ItemEvent	generated when check-box or list item is clicked	ItemListener
TextEvent	generated when value of textarea or textfield is changed	TextListener

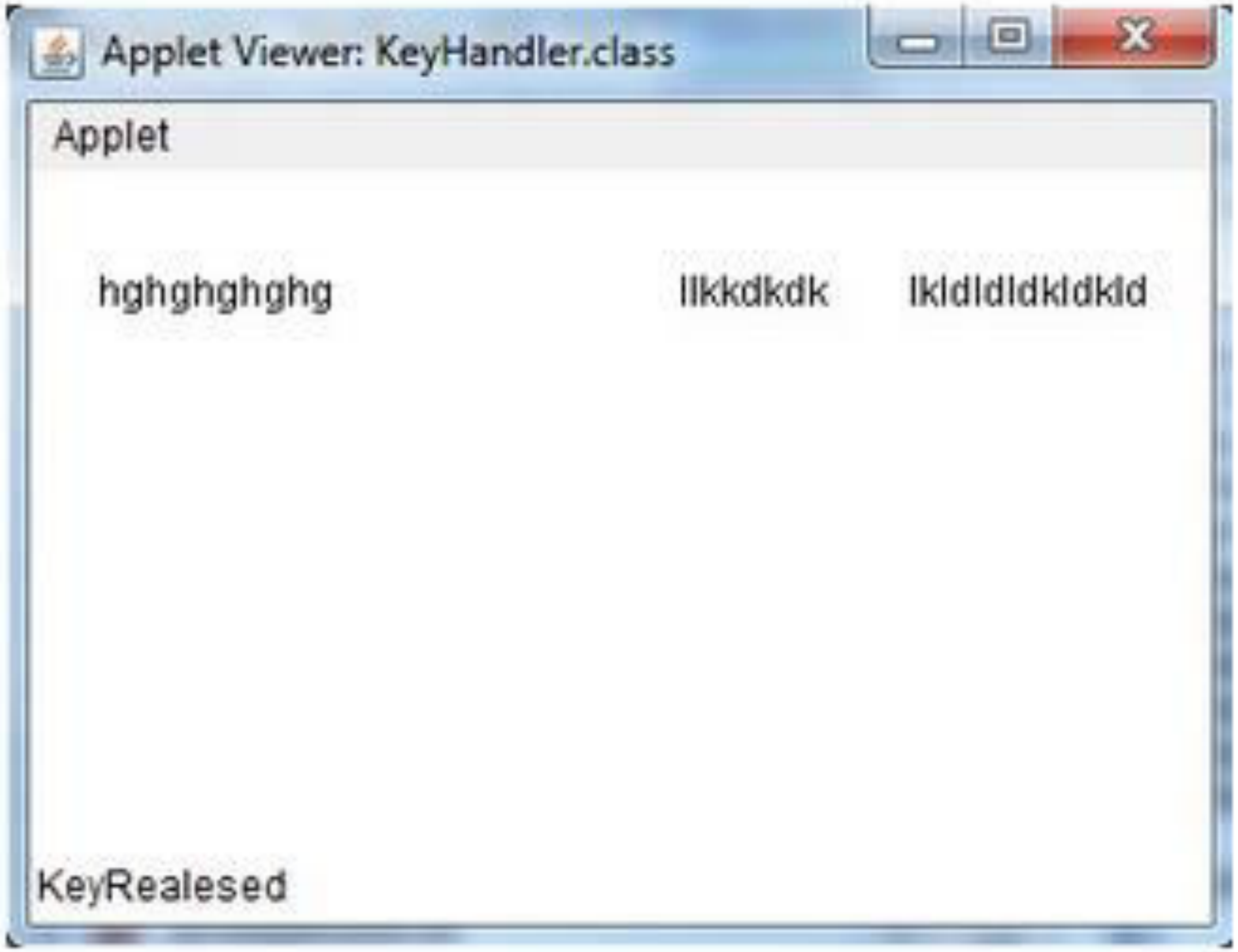
MouseWheelEvent	generated when mouse wheel is moved	MouseWheelListener
WindowEvent	generated when window is activated, deactivated, deiconified, iconified, opened or closed	WindowListener
ComponentEvent	generated when component is hidden, moved, resized or set visible	ComponentEventListener
ContainerEvent	generated when component is added or removed from container	ContainerListener
AdjustmentEvent	generated when scroll bar is manipulated	AdjustmentListener
FocusEvent	generated when component gains or loses keyboard focus	FocusListener

- import java.awt.*;
- import java.awt.event.*;
- import java.applet.*;
- import java.applet.*;
- import java.awt.event.*;
- import java.awt.*;
- public class Test extends Applet implements KeyListener {
- String msg="";

```
 public void init() {
 addKeyListener(this);
 }
public void keyPressed(KeyEvent k) {
showStatus("KeyPressed");
 }
 public void keyReleased(KeyEvent k) {
showStatus("KeyRealesed");
 }
```

```
public void keyTyped(KeyEvent k) {
 msg = msg+k.getKeyChar();
 repaint();
 }
 public void paint(Graphics g) {
 g.drawString(msg, 20, 40);
}
}
```

HTML code : < applet code="Test" width=300, height=100 >



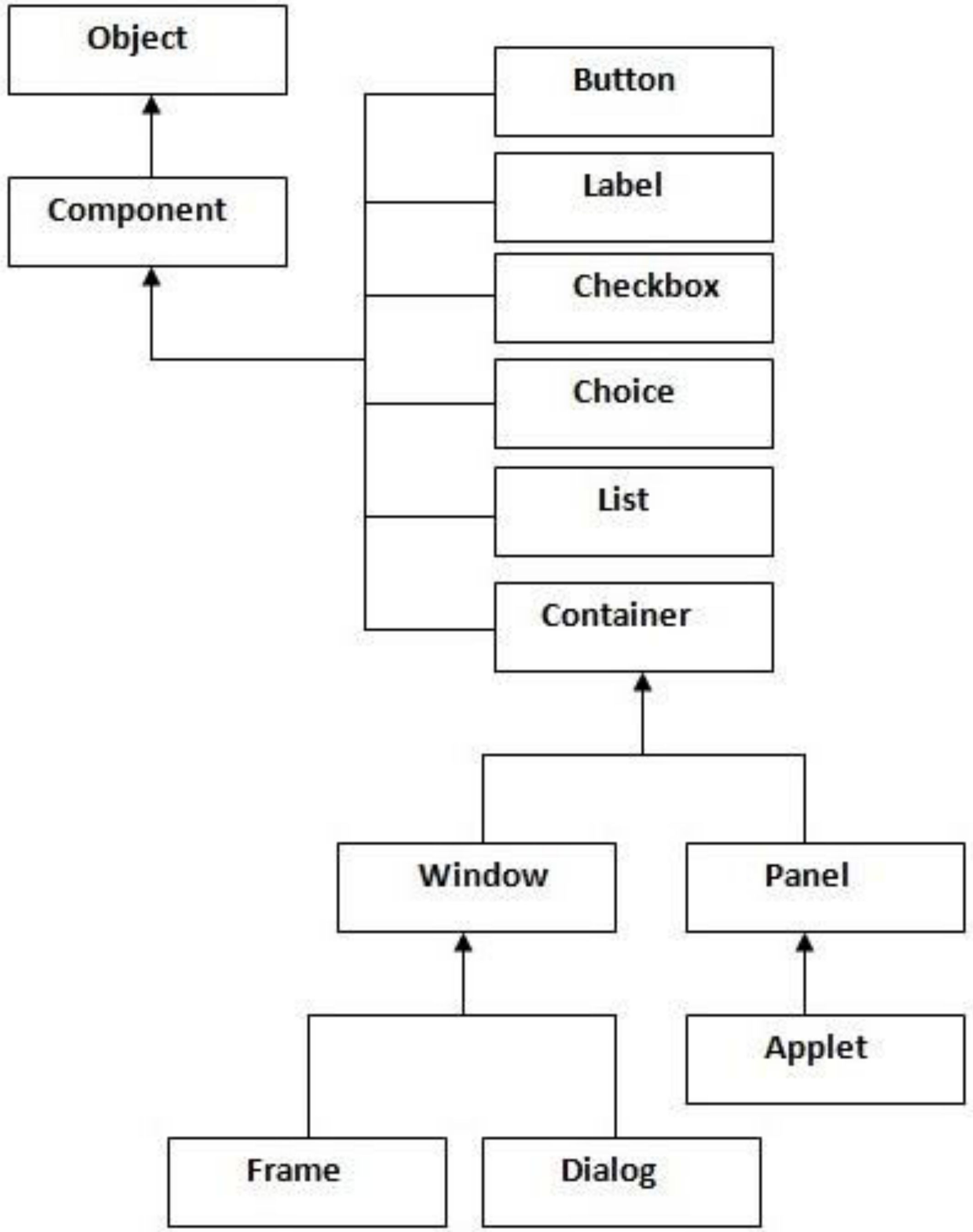
```
hghghghghg                ilkkdkdk   lkldldldkldkld

KeyRealesed
```

Java AWT

- **Package java.awt c**ontains all of the classes for creating user interfaces and for painting graphics and images.

- **Java AWT** (Abstract Windowing Toolkit) is *an API to develop GUI or window-based application in java*.
- Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components uses the resources of system.
- The java.awt package provides classes for AWT api such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

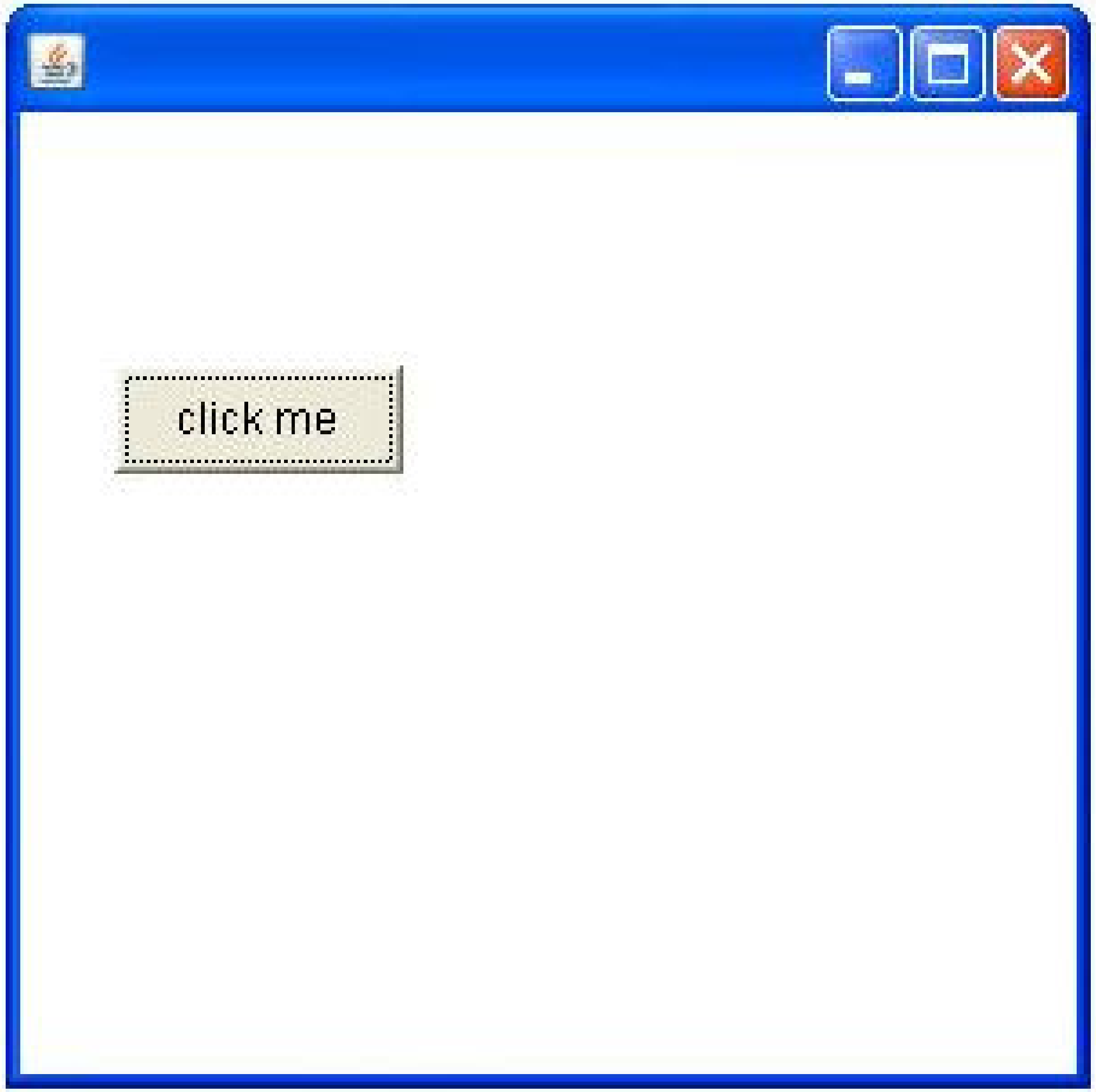
Java AWT Hierarchy



- **Container**
- The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. The classes that extends Container class are known as container such as Frame, Dialog and Panel.
- **Window**
- The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window.
- **Panel**
- The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.
- **Frame**
- The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.

- The Java programming language class library provides a user interface toolkit called the Abstract Windowing Toolkit, or the AWT. The AWT is both powerful and flexible. Newcomers, however, often find that its power is veiled. The class and method descriptions found in the distributed documentation provide little guidance for the new programmer.

- import java.awt.*;
- class First extends Frame{
- First(){
- Button b=new Button("click me");
- b.setBounds(30,100,80,30);// setting button position
-
- add(b);//adding button into frame
- setSize(300,300);//frame size 300 width and 300 height
- setLayout(null);//no layout manager
- setVisible(true);//now frame will be visible, by default not visible
- }
- public static void main(String args[]){
- First f=new First();
- }}


click me

AWT CONTROLS

- Controls Are components that allow a user to interact with your application in various ways.
- LayoutManager automatically positions within a container.
- So we can say that the appearance of a window is a combination of such controls and Layoutmanagers.

- Control Fundamentals
- AWT supports the following controls:
- Labels
- Push Buttons
- Check Boxes
- Choice Lists
- Lists
- Scroll Bars
- Text Editing

- How to add or remove controls?
- Create an instance of desired control and then add it to the window by calling add() method.

  Syntax:-
  Component add(Component obj)
- We can also remove our control from window by calling remove() method.

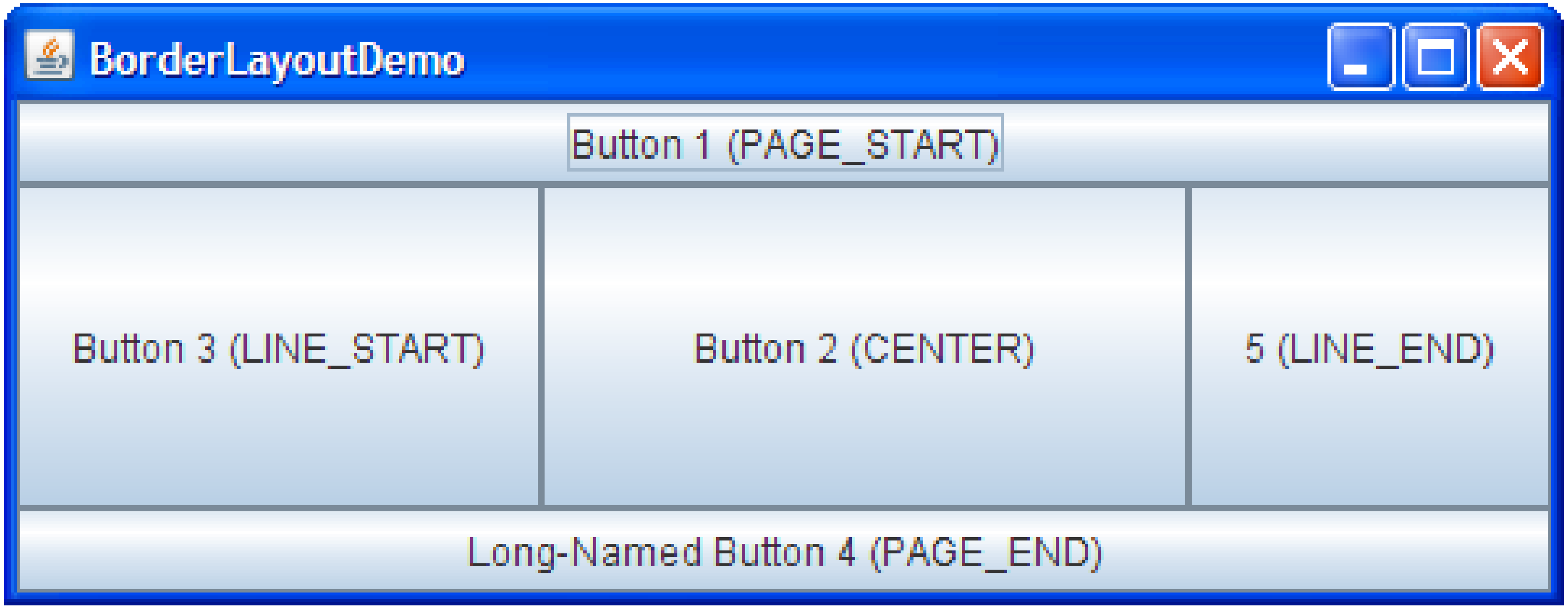
  Syntax:-
  Void remove(Component obj)
  And also controls can be removed by calling removeAll()

- Label
- A Label object is a component for placing text in a container.
- 2Button
- This class creates a labeled button.
- 3Check Box
- A check box is a graphical component that can be in either an **on** (true) or **off** (false) state.
- 4Check Box Group
- The CheckboxGroup class is used to group the set of checkbox.
- 5List
- The List component presents the user with a scrolling list of text items.

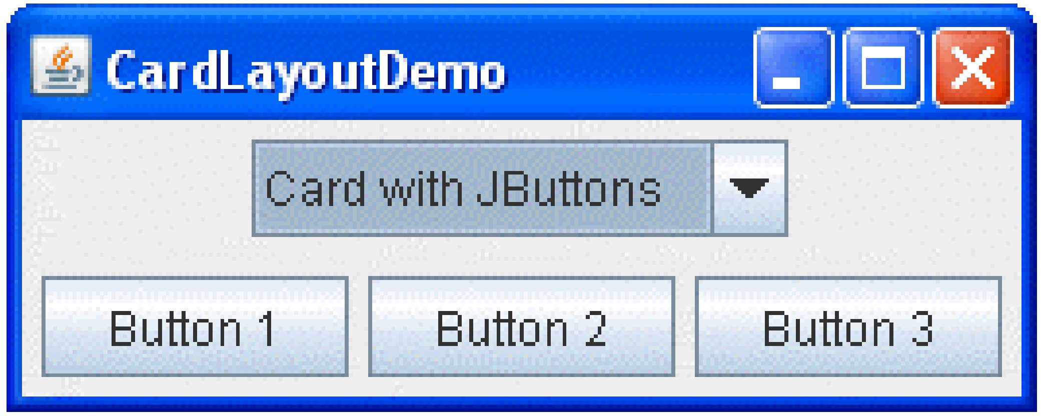
- Text Field
- A TextField object is a text component that allows for the editing of a single line of text.
- 7Text Area
- A TextArea object is a text component that allows for the editing of a multiple lines of text.
- 8Choice
- A Choice control is used to show pop up menu of choices. Selected choice is shown on the top of the menu.
- 9Canvas
- A Canvas control represents a rectangular area where application can draw something or can receive inputs created by user.
- 10Image
- An Image control is superclass for all image classes representing graphical images.

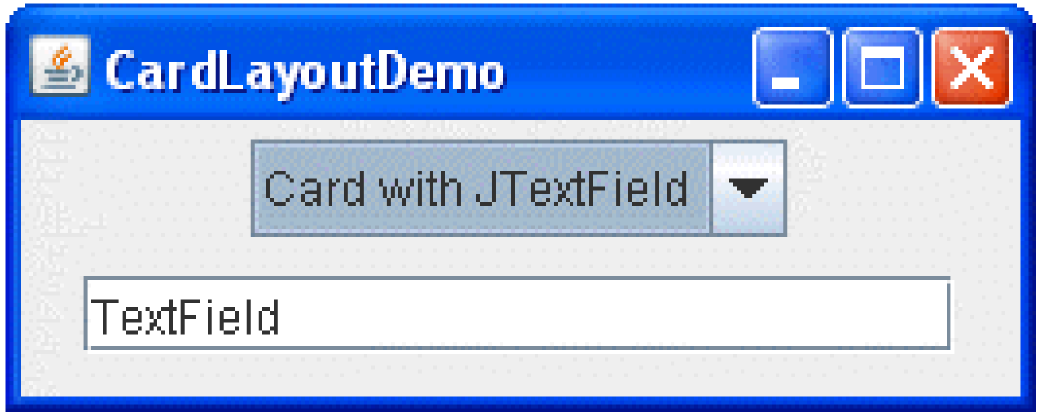
- **LayoutManagers:**
- The LayoutManagers are used to arrange components in a particular manner. LayoutManager is an interface that is implemented by all the classes of layout managers. There are following classes that represents the layout managers:
- java.awt.BorderLayout
- java.awt.FlowLayout
- java.awt.GridLayout
- java.awt.CardLayout
- java.awt.GridBagLayout
- javax.swing.BoxLayout
- javax.swing.GroupLayout
- javax.swing.ScrollPaneLayout
- javax.swing.SpringLayout etc.

- **BorderLayout**
- Every content pane is initialized to use a BorderLayout. (As Using Top-Level Containers explains, the content pane is the main container in all frames, applets, and dialogs.) A BorderLayout places components in up to five areas: top, bottom, left, right, and center. All extra space is placed in the center area. Tool bars that are created using JToolBar must be created within a BorderLayout container, if you want to be able to drag and drop the bars away from their starting positions.

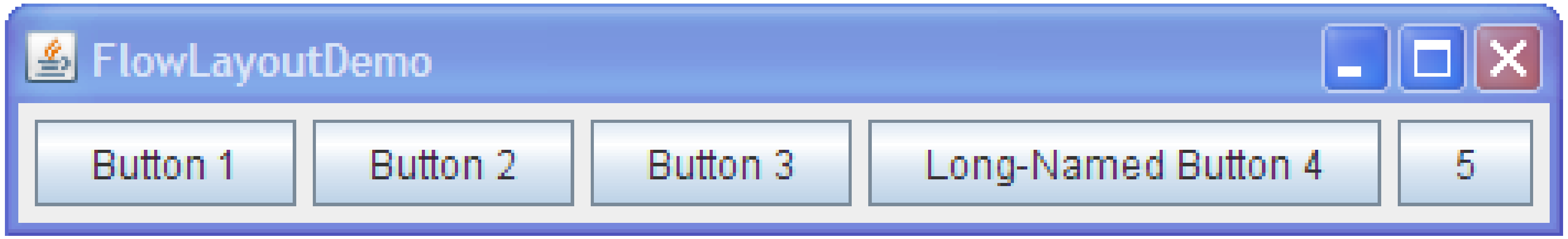

BorderLayoutDemo
Button 1 (PAGE_START)
Button 3 (LINE_START)
Button 2 (CENTER)
5 (LINE_END)
Long-Named Button 4 (PAGE_END)

- The BoxLayout class puts components in a single row or column. It respects the components' requested maximum sizes and also lets you align components

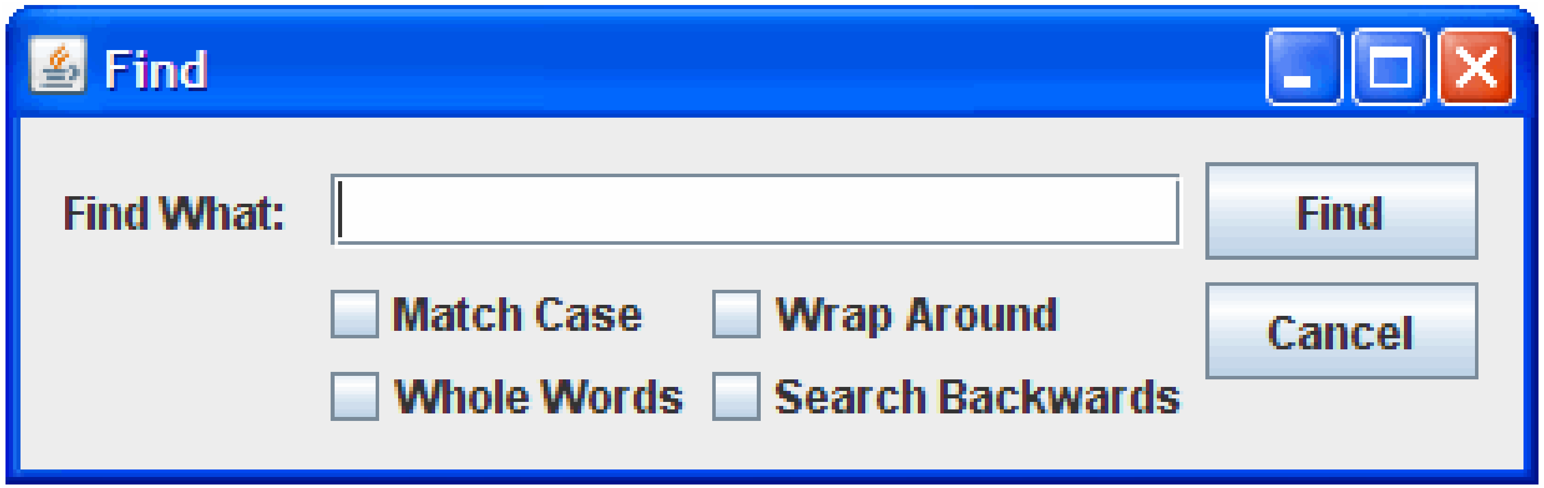




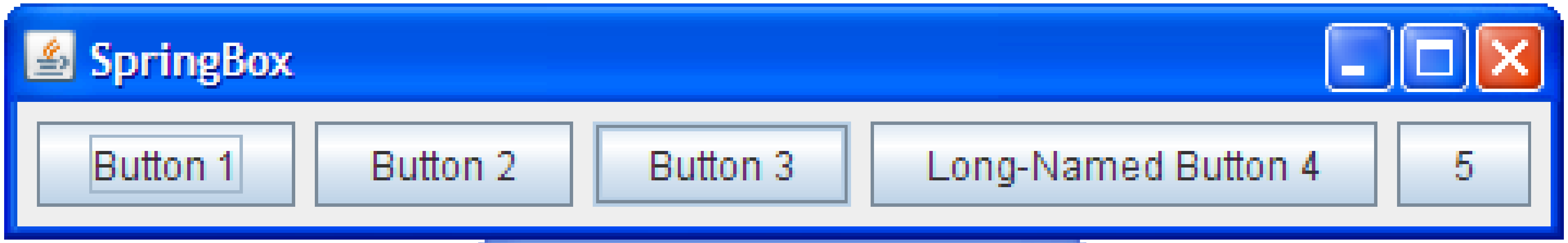
- FlowLayout is the default layout manager for every JPanel. It simply lays out components in a single row, starting a new row if its container is not sufficiently wide. Both panels in CardLayoutDemo, shown previously, use FlowLayout.

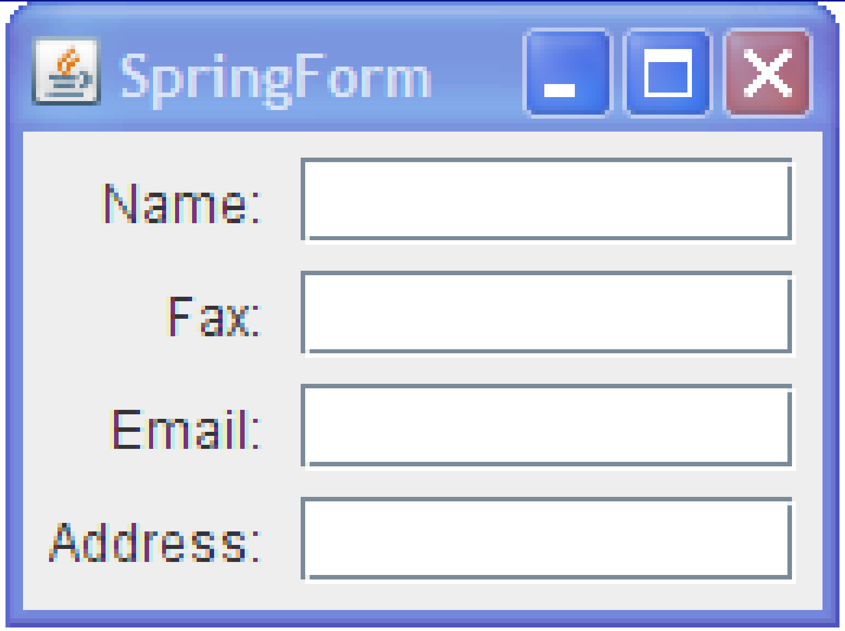


- GridLayout simply makes a bunch of components equal in size and displays them in the requested number of rows and columns.



- SpringLayout is a flexible layout manager designed for use by GUI builders. It lets you specify precise relationships between the edges of components under its control. For example, you might define that the left edge of one component is a certain distance (which can be dynamically calculated) from the right edge of a second component. SpringLayout lays out the children of its associated container according to a set of constraints


SpringBox
Button 1
Button 2
Button 3
Long-Named Button 4
5


SpringForm
Name:
Fax:
Email:
Address: