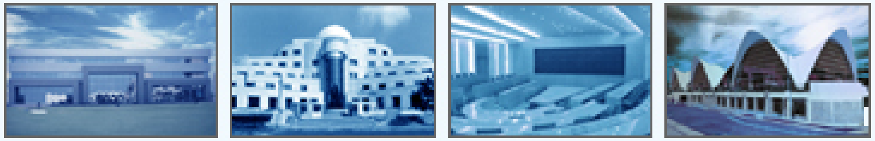
Infosys®

POWERED BY INTELLECT
DRIVEN BY VALUES

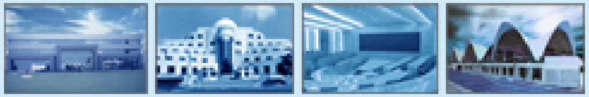


Java Programming

Education
and
Research

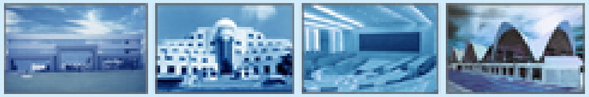
Our Agenda

- Introduction to OOPS
- What is Java?
- The building Blocks of Java-- JDK
- Java architecture
- The java.lang. package
- Basic program constructs
- Applets v/s Applications



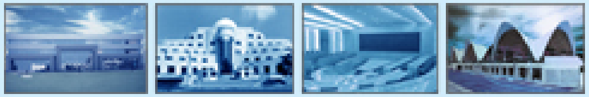
Agenda Continued

- The Utilities in util package
- User interface & Event Handling
- Exceptions Handling



Before the first Cup….

- Before we begin something on installation
  - Have jdk1.2.2 installed
  - you get an exe version
  - set path after installation
  - to test open a command window and type javac
  - if it gives a help for the command the installation is OK

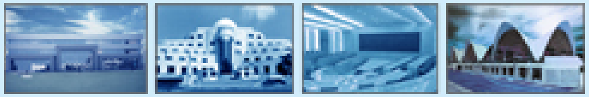


Introducing to OOPS

- You need to familiarize yourself with some terms like “Class”, “Object”, “Inheritance”, “Polymorphism” etc etc….

- The programming style that you use in C where importance is given for functions can be called as structured programming

- Importance is given only to functionality, no importance is associated with the Data.

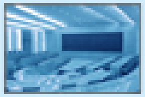


Class…..Objects….

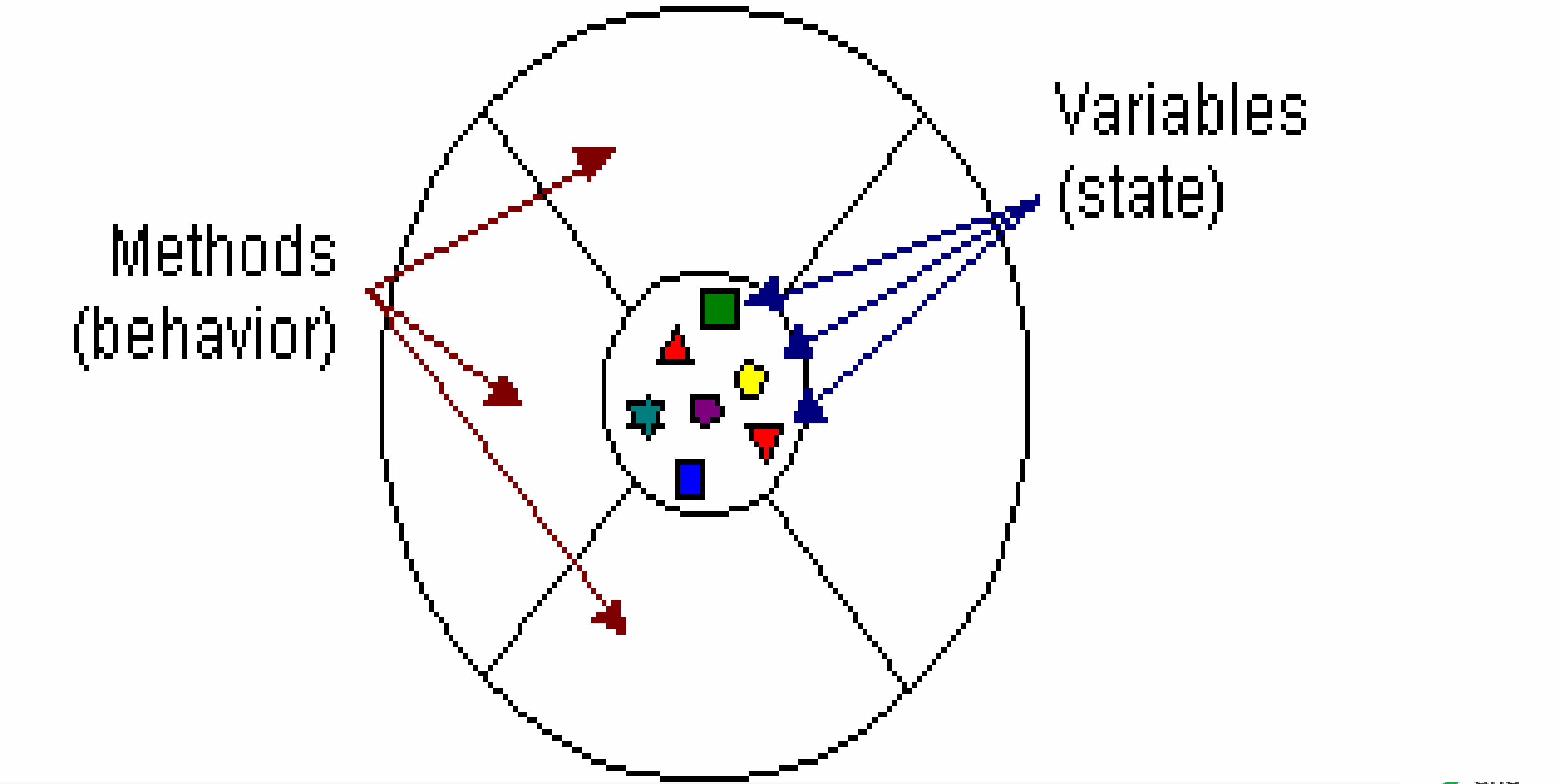
- What is a Class?
  - A class is a blueprint or prototype that defines the variables and the methods common to all objects of a certain kind.

- So what is an Object?
  - An object is a software bundle of related variables and methods. Software objects are often used to model real-world objects you find in everyday life.

Education and Research



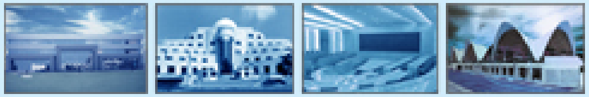
Representation of Object



The Object

- **The object diagrams show that the object's variables make up the center, or nucleus, of the object**

- **Methods surround and hide the object's nucleus from other objects in the program. Packaging an object's variables within the protective custody of its methods is called *encapsulation***

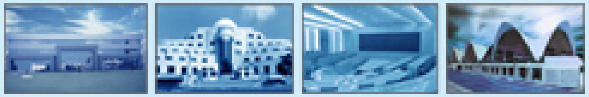


State & Behavior

- The functions are called methods and the variables attributes

- The “value” of the attributes is called the state.

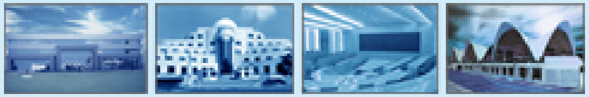
- Somebody calling a method on an object and the method getting executed by making use of current state is invoking the behavior



How do you write a class

- In Java we use a key word class
  - A class is defined as follows

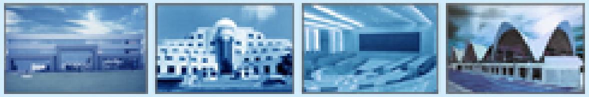
```
class Employ {
   String Name;//attribute
   int  Age;//attribute
//a behavior
void printDetails(){
   System.out.println(“Name is”+Name);
   System.out.println(“Age is”+Age);
   }
}
```



Inheritance

- **Now you have understood a Class let us look at what is inheritance.**

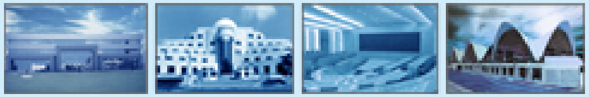
- **A class inherits state and behavior from its super-class. Inheritance provides a powerful and natural mechanism for organizing and structuring software programs.**



Inheritance

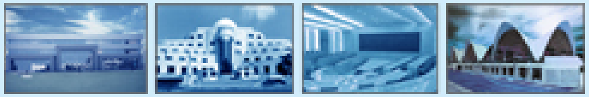
- However, subclasses are not limited to the state and behaviors provided to them by their superclass.

- Subclasses can add variables and methods to the ones they inherit from the superclass.



Overriding

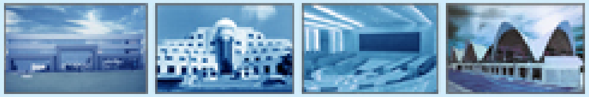
- Subclasses can also *override* inherited methods and provide specialized implementations for those methods.
- You are not limited to just one layer of inheritance. The inheritance tree, or class hierarchy, can be as deep as needed.
- Methods and variables are inherited down through the levels



Inheritance

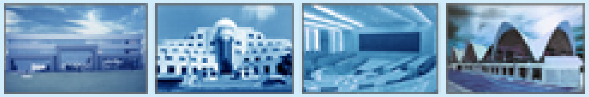
- Inheritance offers the following benefits:
  - Subclasses provide specialized behaviors from the basis of common elements provided by the super class

- Programmers can implement super-classes called ***abstract classes*** that define "generic" behaviors.
  - – The abstract superclass defines and may partially implement the behavior, but much of the class is undefined and unimplemented



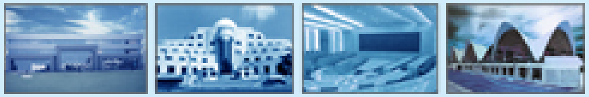
What is Java?

- A language developed at Sun Microsystems
- A general-purpose language
- High-level language
- Developed initially for consumer devices
- Help in building a dynamic Web
- Supported today by most of the big players like IBM, Netscape, Oracle, Inprise etc.



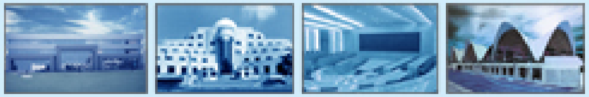
Features Of Java

- Object-oriented
- Simple
- Robust
- Secure
- Architecture Neutral / Portable
- Multithreaded
- Distributed



Java - The Basics

- Draws features from OO languages like Smalltalk, C++, Ada
- An interpreted language
- Uses a virtual machine called Java Virtual Machine (JVM)
- A very exhaustive OO library



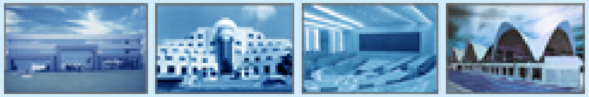
Hello World

- **We will have the source code first**

  Type this into any text editor

```
public class HelloWorldApp {

public static void main(String[]args){

   System.out.println(“Hello World!”);

   }

}
```

- **Save this as HelloWorldApp.java (take care case matters…..)**



Some Rules

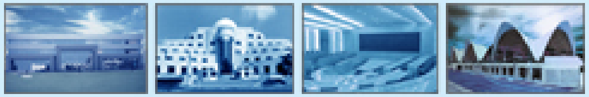
- **The name of the file must always be the name of the "public class"**
- **It is 100% case sensitive**
- **You can have only one public class in a file(i.e. in one .java file)**
- **Every "stand alone" Java program must have a public static void main defined**
  - **it is the starting point of the program.**

To Compile

- Open a command prompt

- Go to the directory you have saved your program.

- Type javac HelloWorldApp.java.
  - If it says bad command or file name set the path
  - If it does not say anything and get the prompt the compilation was successful.

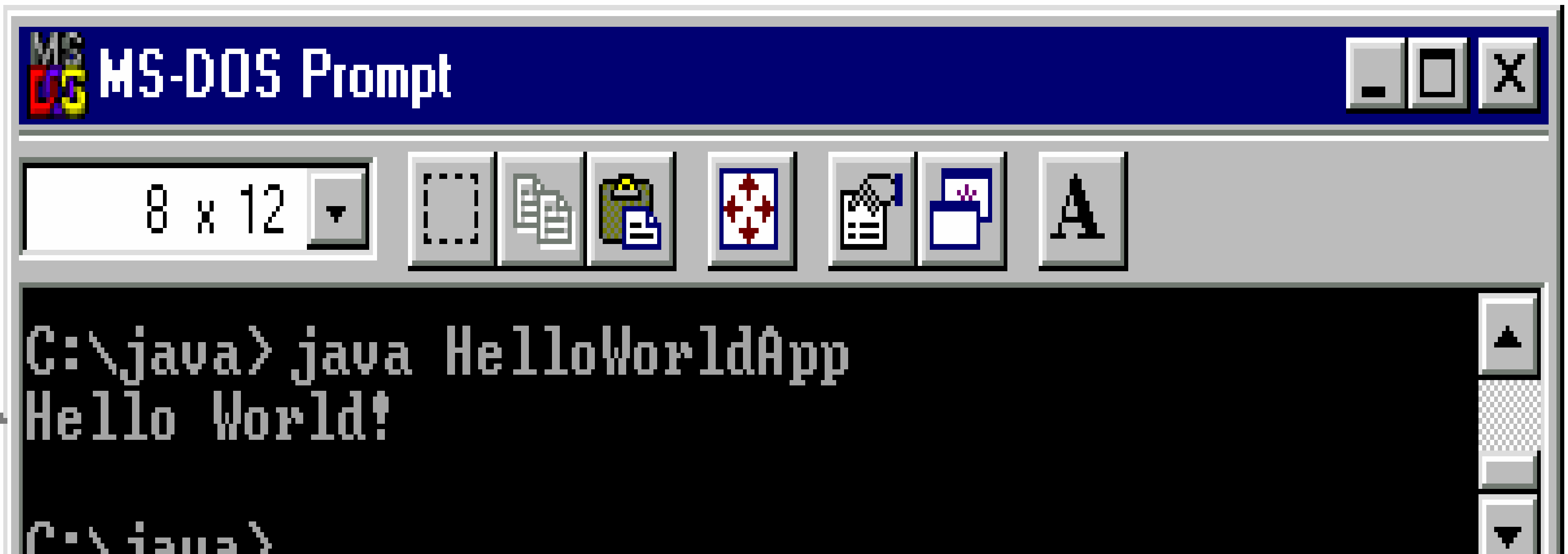


To execute

- Type in the command prompt

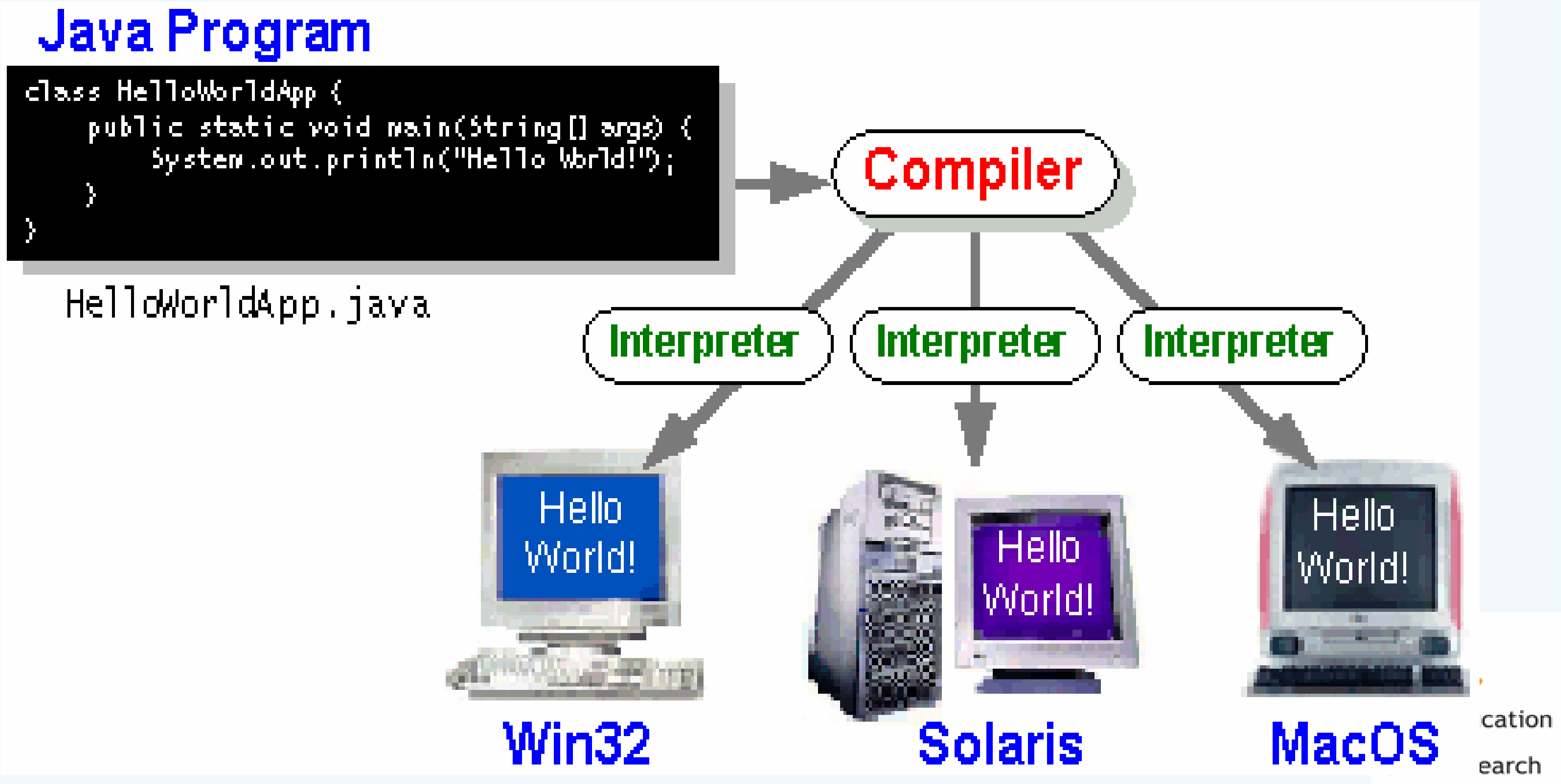
  "java HelloWorldApp"

- The result



```
C:\java>java HelloWorldApp
Hello World!

C:\java>
```

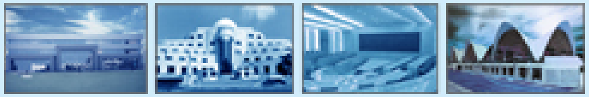
So How did this work…….



```
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

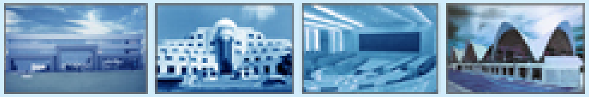
Platform independence……

- Java is a language that is platform independent.
- A *platform* is the hardware or software environment in which a program runs
- Once compiled code will run on any platform without recompiling or any kind of modification.
- This is made possible by making use of a Java Virtual Machine a.k.a. JVM



Java Virtual Machine

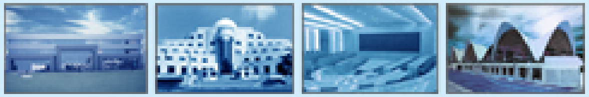
- JVM can be considered as a processor purely implemented with software.
- The .class file that is generated is the machine code of this processor.
- The interface that the JVM has to the .class file remains the same irrespective of the underlying platform .
- This makes platform independence possible



Platform independence

- The JVM interprets the .class file to the machine language of the underlying platform .

- The underlying platform processes the commands given by the JVM and returns the result back to JVM which displays it for you.

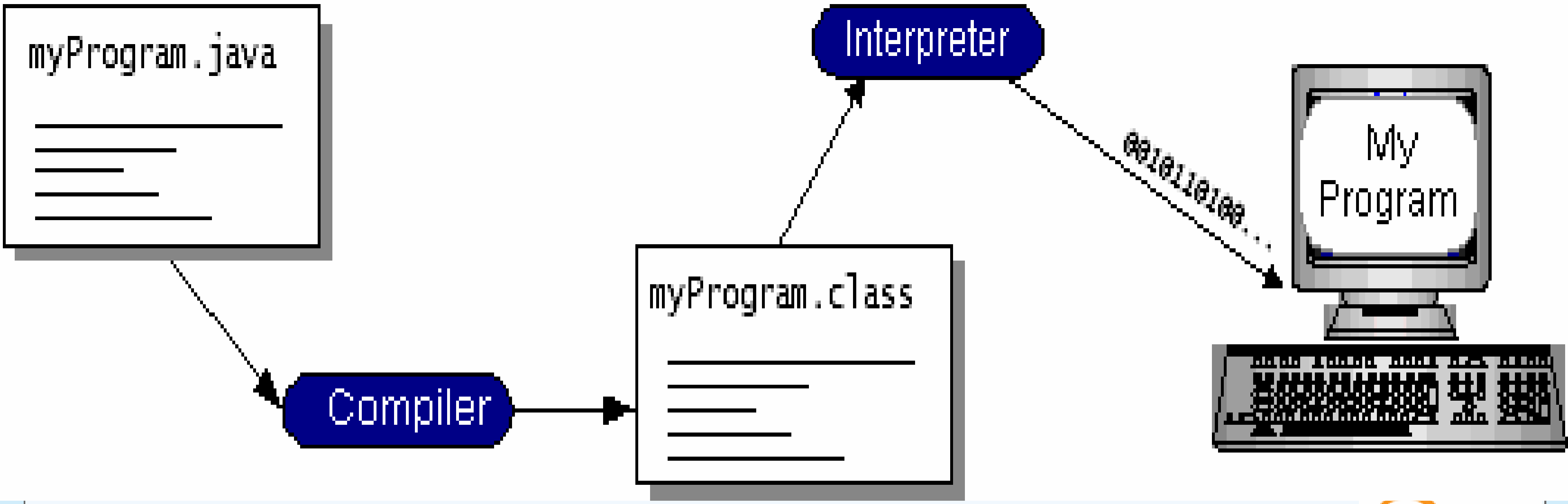


The life cycle

- The Java programming language is unusual in that a program is both compiled and interpreted

- With the compiler, first you translate a program into an intermediate language called *Java bytecodes*-the platform-independent codes interpreted by the interpreter on the Java platform

A Diagrammatic Representation

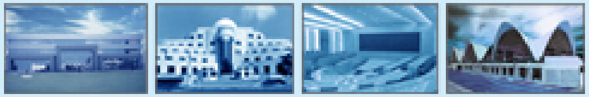


JDK

- JDK or Java Development Kit is a free software that can be used to write and compile Java programs.

- Currently version 1.3 has been released but we will be using version 1.2.2

- It has lots of examples and the Standard Java Class Library also called the API

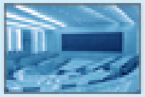


JDK

- We will be making use of the Classes defined in the standard library by creating objects or inheriting from those classes.

- We use the javac compiler provided with JDK

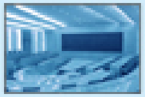
- We have tools like javadoc, rmiregistry, appletviewer etc which we may make use of



The Java Platform

- The Java platform has two components:
  - The *Java Virtual Machine* (Java VM)
  - The *Java Application Programming Interface* (Java API)

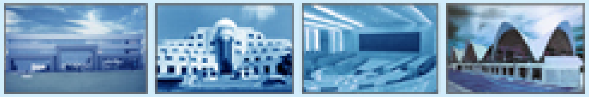
- The Java API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets.



The Java Definition

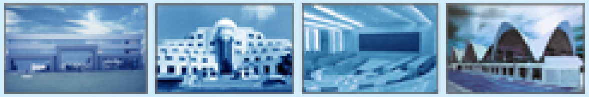
- The Java programming language is a high-level language that can be characterized by all of the following buzzwords:

Simple, Architecture-neutral, Object-oriented, Portable, Distributed, High-performance, Interpreted, Multithreaded, Robust, Dynamic and Secure



Constituents of a Class

- Variables or Data Members
- Constructors
- Functions or Methods
- Classes, also called Inner Classes
- Startup function, if it is a starter class

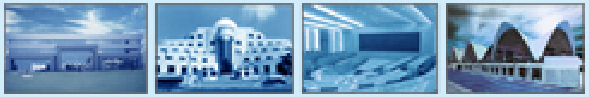


Data Types

- Strongly typed language

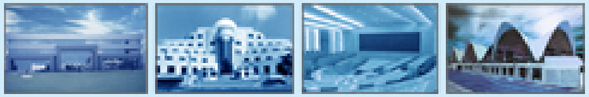
- Two types of variables
  - Primitive type
  - Reference type
  - `null` is a special type

Reference types cannot be cast to primitive types



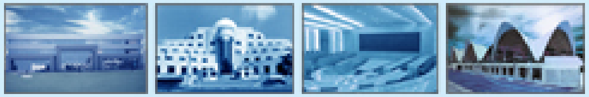
Primitive Data Types

`byte`	Byte-length integer	8-bit two's complement
`short`	Short integer	16-bit two's complement
`int`	Integer	32-bit two's complement
`long`	Long integer	64-bit two's complement*(real numbers)*



Primitive Data Types

`float`	Single-precision floating point	32-bit IEEE 754
`double`	Double-precision floating point	64-bit IEEE 754
`char`	A single character	16-bit Unicode character
`boolean`	A boolean value (`true or false`)	true or false

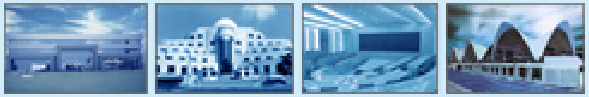


References…..

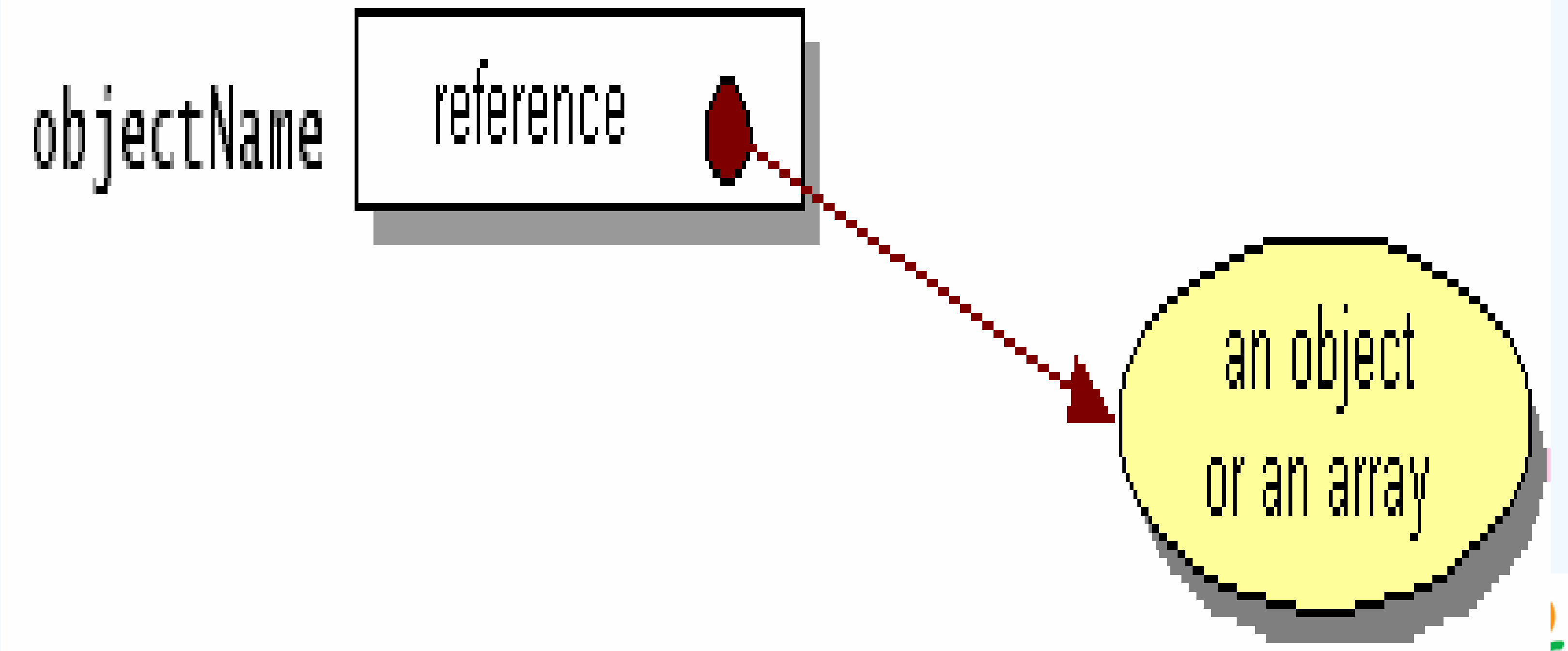
- Arrays, classes, and interfaces are *reference* types

- A reference is called a pointer, or a memory address in other languages

- **The Java programming language does not support the explicit use of addresses like other languages do**

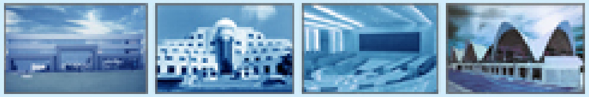


Reference...



Access Specifiers

- There are four access specifiers:
  - public
  - private
  - “ ” - package
  - protected



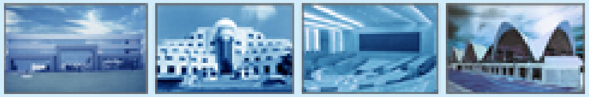
Access for Types

- Access specifiers could be used on classes in Java

- All classes belong to packages in Java

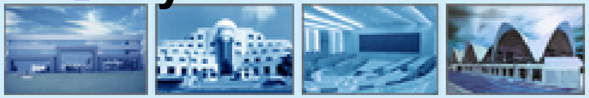
- “public” types are only accessible outside the package

- private and protected specifier are invalid for classes



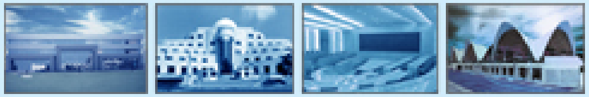
Modifiers in Java

- Access specifiers
- static
- final
- abstract
- native
- synchronized



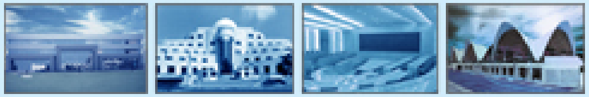
“final” Modifier

- “final” modifier has a meaning based on its usage
- For variable:
  - Primitives: read-only
  - Objects: reference is read-only
  - use all upper case letters
- For methods: no overriding
- For classes: no inheritance



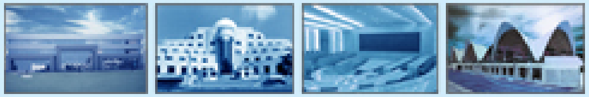
“abstract” Modifier

- “abstract” modifier is used to defer an operation
- Cannot be used for variables
- For methods: no implementation
- For classes: no instantiation
- A concrete class can be made abstract by using the modifier for the class



Rules to Follow

- The following cannot be marked with “abstract” modifier
  - Constructors
  - Static methods
  - Private methods
  - Methods marked with “final” modifier

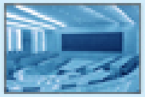


“native” Modifier

- “native” modifier is used to indicate implementation of the method in a non-Java language, like C/C++

- The library where the method is implemented should be loaded before invoking native methods

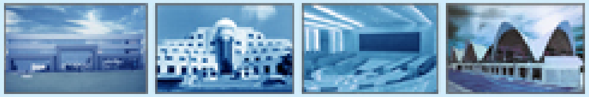
- “synchronized” Modifier
    - Discussed in the module on threading



Variables

- The Java programming language has two categories of data types: *primitive* and *reference.*

- A variable of primitive type contains a single value of the appropriate size and format for its type: a number, a character, or a boolean value

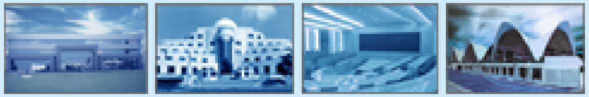


Scope of variables

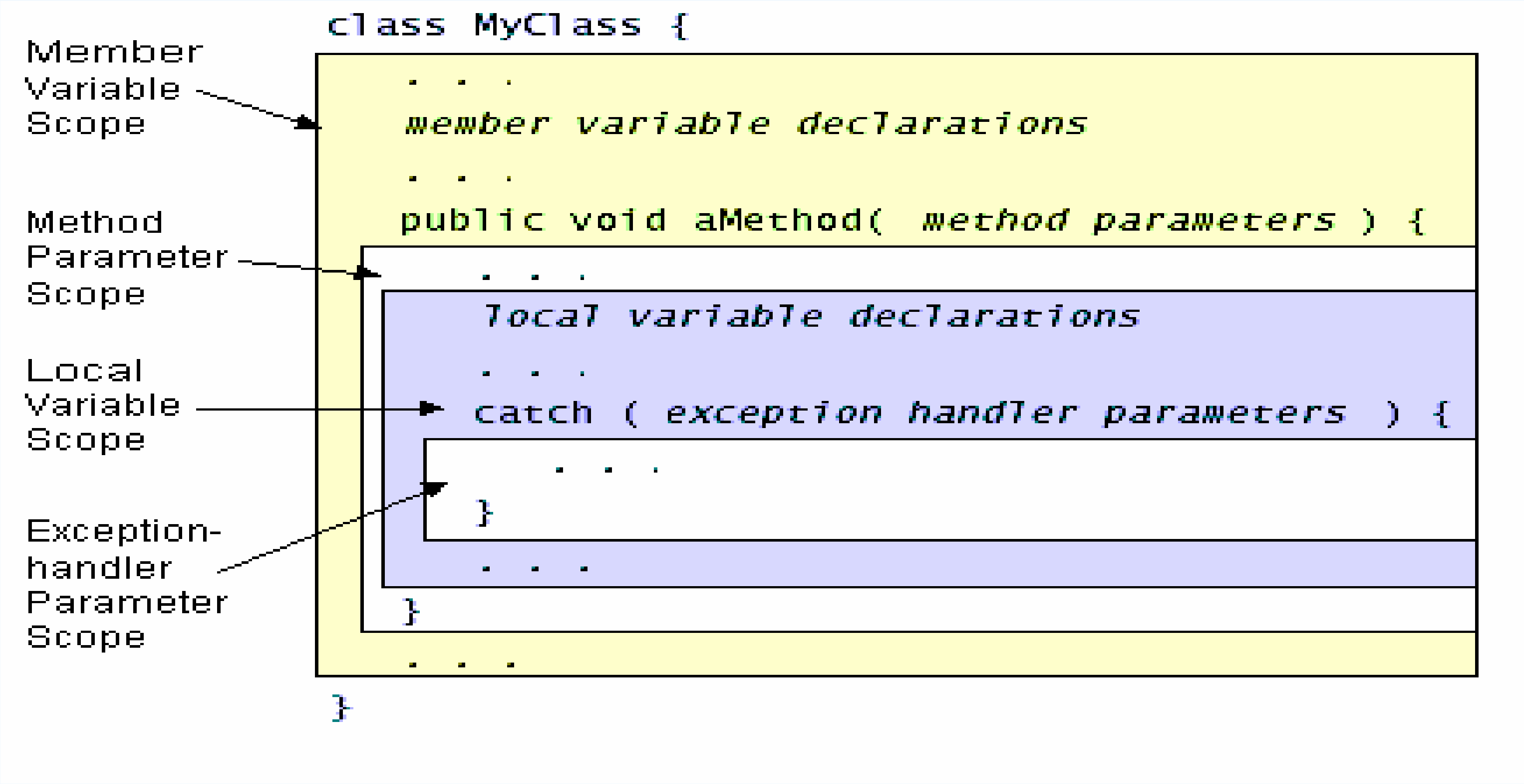
- A variable's scope is the region of a program within which the variable can be referred to by its simple name.

- Scope also determines when the system creates and destroys memory for the variable

- Don’t confuse Scope with Visibility



Scope….



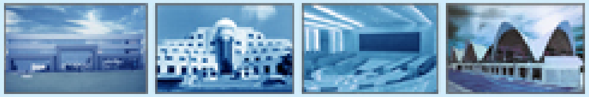
```
class MyClass {
    . . .
    member variable declarations
    . . .
    public void aMethod( method parameters ) {
        . . .
        local variable declarations
        . . .
        catch ( exception handler parameters ) {
            . . .
        }
        . . .
    }
    . . .
}
```

ion

:h

Member Variables

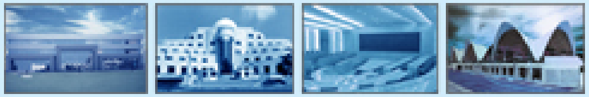
- A member variable is a member of a class or an object.
- It is declared within a class but outside of any method or constructor.
- A member variable's scope is the entire declaration of the class.
- The declaration of a member needs to appear before it is used



Local Variables

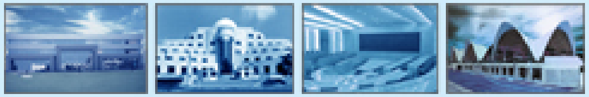
- You declare local variables within a block of code

- The scope of a local variable extends from its declaration to the end of the code block in which it was declared



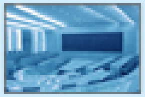
Parameter Scope

- Parameters are formal arguments to methods or constructors and are used to pass values into methods and constructors.
- The scope of a parameter is the entire method or constructor for which it is a parameter.
- Exception-handler parameters are similar to parameters but are arguments to an exception handler rather than to a method or a constructor



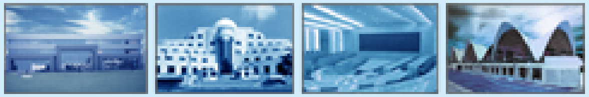
Final variables

- You can declare a variable in any scope to be final .
- The value of a final variable cannot change after it has been initialized.
- Such variables are similar to constants in other programming languages.
- To declare a final variable, use the final keyword in the variable declaration before the type: final int Var = 0;



Visibility

- Visibility is set with an access modifier
- Applies only to member variables and determines whether the variable can be used from outside of the class within which it is declared.
- The access modifiers are public, protected, private and default(when none specified)
- The default scope is Package.

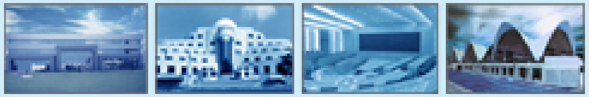


Public-Private

- Public variables and methods are those which can be accessed from any where i.e. From the class, outside the class and outside the package.

- Private variables are those which can be accessed only within the class.

- They are not visible outside that class.

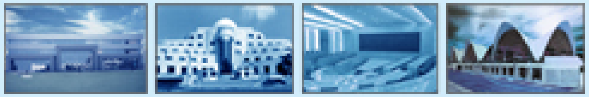


Protected

- Protected variables re those which are visible only inside the class and the children classes of that class.

- If your class extends a base class then your derived class will be able to access the variables and methods of the base class that are declared as protected

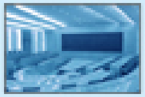
  (and public of course….)



Default Scope

- The default Scope i.e. if you don’t specify any access modifiers the scope is package scope.

- It means that within the package the class is it will be accessible but outside the package it is not accessible.

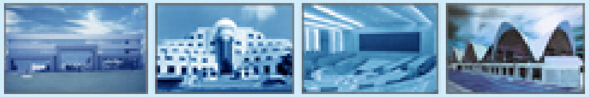


Class Member Access

	Private	Friendly	Protected	Public
Same class	Yes	Yes	Yes	Yes
Same Package subclass	No	Yes	Yes	Yes
Same Package non-subclass	No	Yes	Yes	Yes
Different Package subclass	No	No	Yes	Yes
Different Package non-subclass	No	No	No	Yes

ation

arch



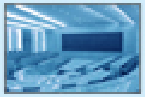
The syntax...

- Java follows exactly the syntax of C with some minor differences.

- A happy news --------

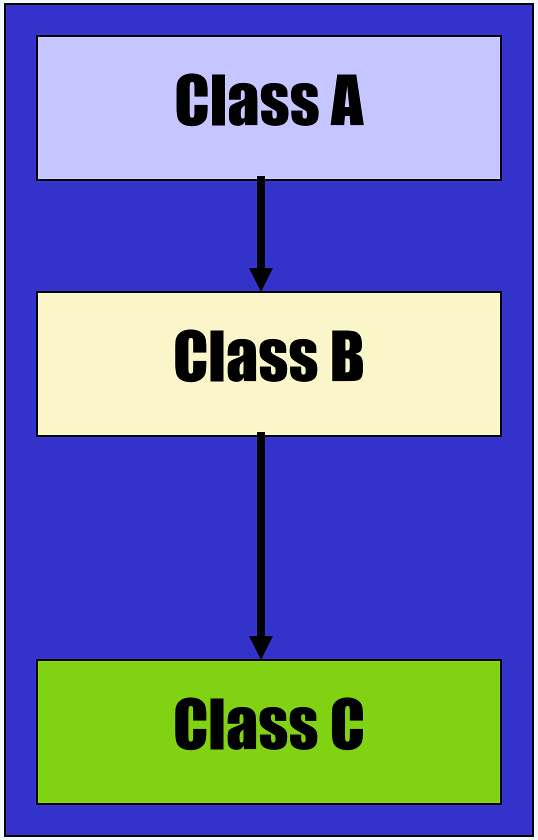
  THERE IS NO POINTERS IN JAVA

- But we have a concept called reference that we have discussed already

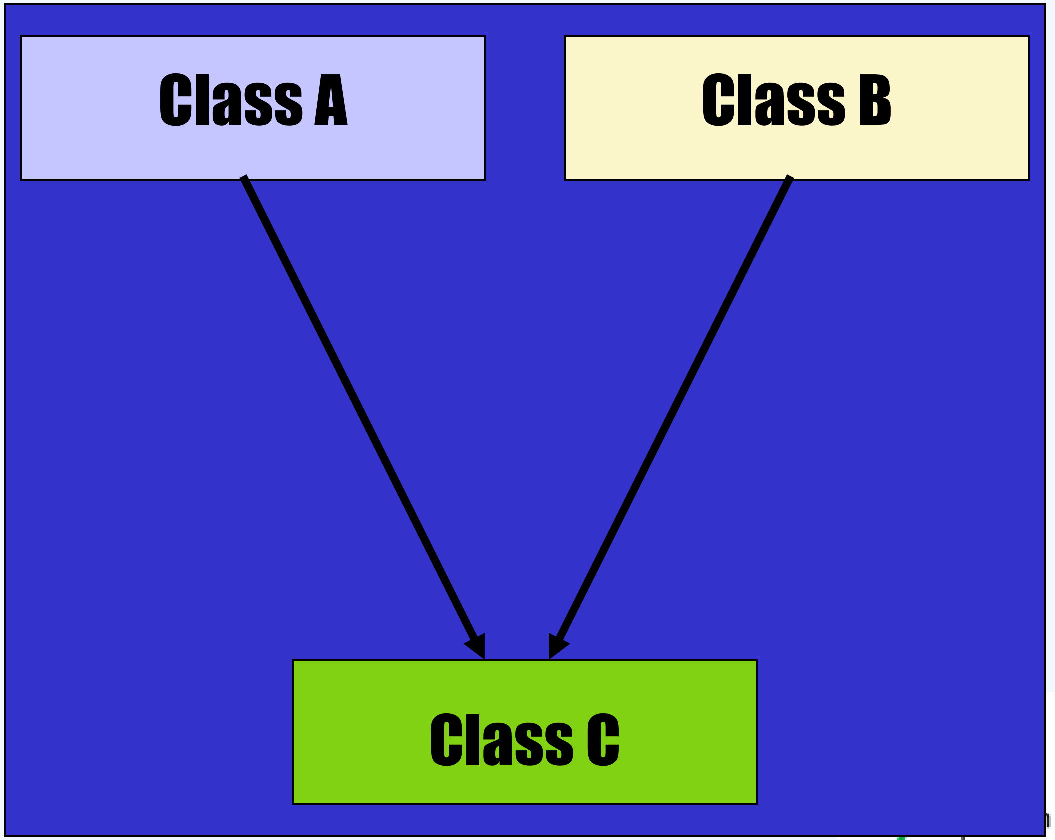


Interfaces

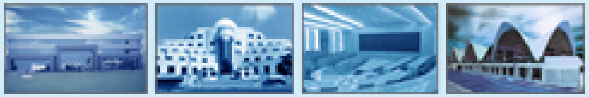
(Inheritance in Java]



Allowed in Java



Not Allowed in Java



Interface

Following are the code for the diagram in the slide shown before :

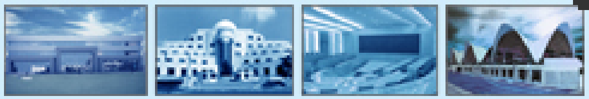
```
Class B extends A
{

 }

Class C extends B
{

 }
```

✓

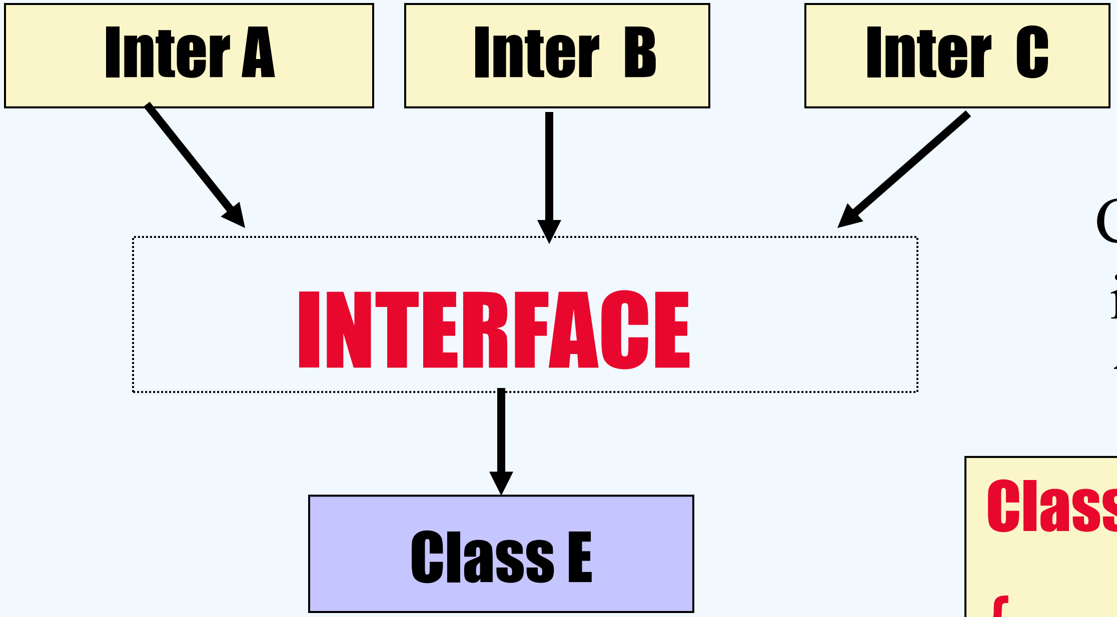
```
Class  C  extends A , B
{

 }
```

✗

The code written above is not acceptable by Java



Implementing Multiple Inheritance in Java



Inter A

Inter B

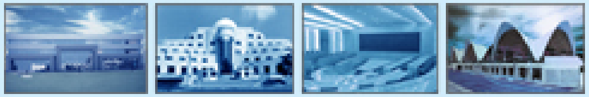
Inter C

INTERFACE

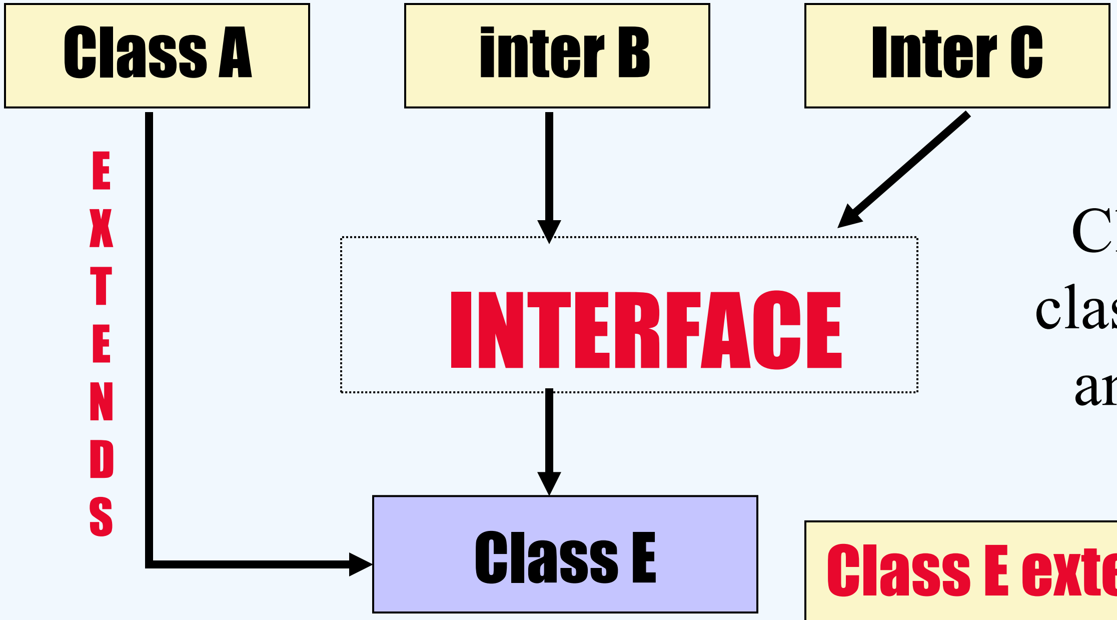
Class E

Class E can inherit from interface A, B and C in the following manner :

```
Class  E  implements  A, B, C
{
   ............;
}
```



Another way of implementing multiple Inheritance



Class E can inherit from classes A,& implements B and C in another way as shown here :

```
Class E extends A  implements B, C
{
   .............;
}
```

Creating an interface class

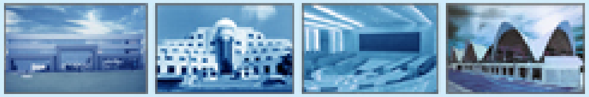
In Java interfacing is done in the following manner :

When the code is executed as given below, “myinterface”.class file will be created in the folder “JavaProgs”

```
public interface myinterface
{
  public void add(int x, int y) ;
}
```

When the code for interface is executed as given below :

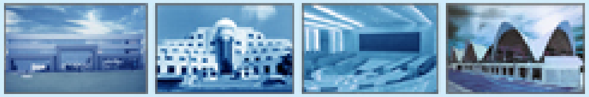
```
javac  –d  c:\JavaProgs\  myinterface . java
```



Using interface in Programs

Importing the folder where myinterface.class file is stored

```
import  java.io.* ;
import  mypackage.* ;
Class demo implements myinterface
{
                public void add(int x., int y)
                {
                                System.out.println("   " + ( x + y  );
                }
}
Public static void main(String args[ ])
{
                deno  d  =  new  demo ( )  ;
                 d.add (10 , 20 ) ;
}
```

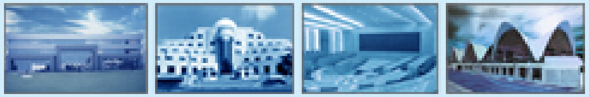


Interfaces Contd…

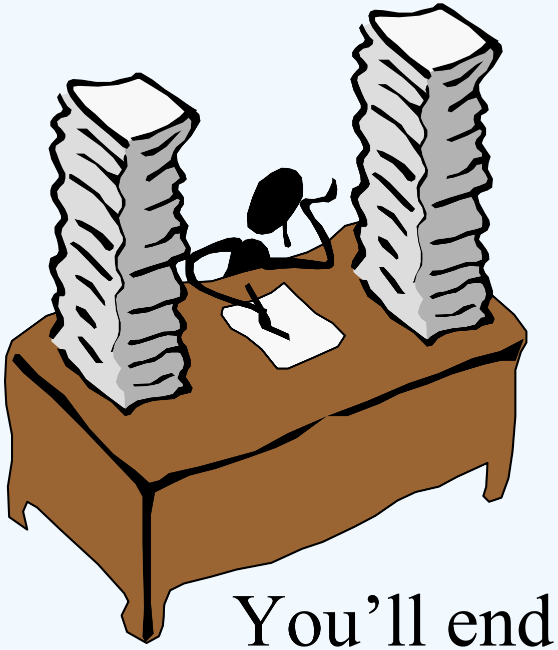
- Comparable to a pure abstract class

- Classes implement, by providing definitions to the methods of the interface

- Inheritance is possible in interfaces, even multiple inheritance is possible



Why use Packages ?



Just think of Writing the code from the scratch, each time you create an application

You’ll end up spending your precious time and energy and finally land up with a Huge code accumulated before you.

Reusing The Existing Code

Reusability of code is one of the most important requirements in the software industry.

Reusability saves time, effort and also ensures consistency.

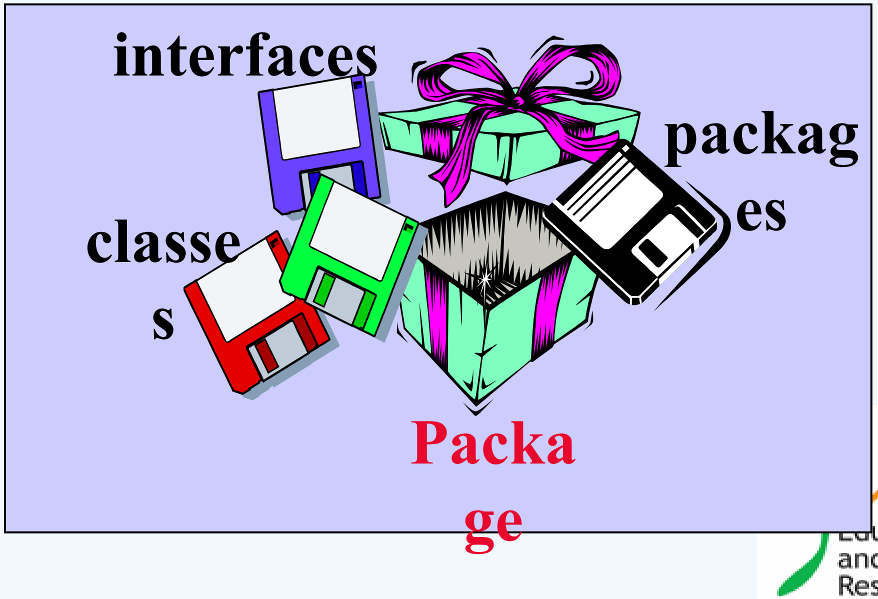
A class once developed can be reused by any number of programs wishing to incorporate the class in that particular program.

Concept of Packages

In Java, the codes which can be reused by other programs is put into a “Package”.

A Package is a collection of classes, interfaces and/or other packages.

Packages are essentially a means of organizing classes together as groups.



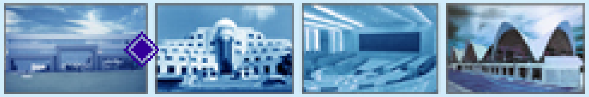
Features of Packages

Packages are useful for the following purposes: Packages allow you to organize your classes into smaller units (such as folders) and make it easy to locate and use the appropriate class file.

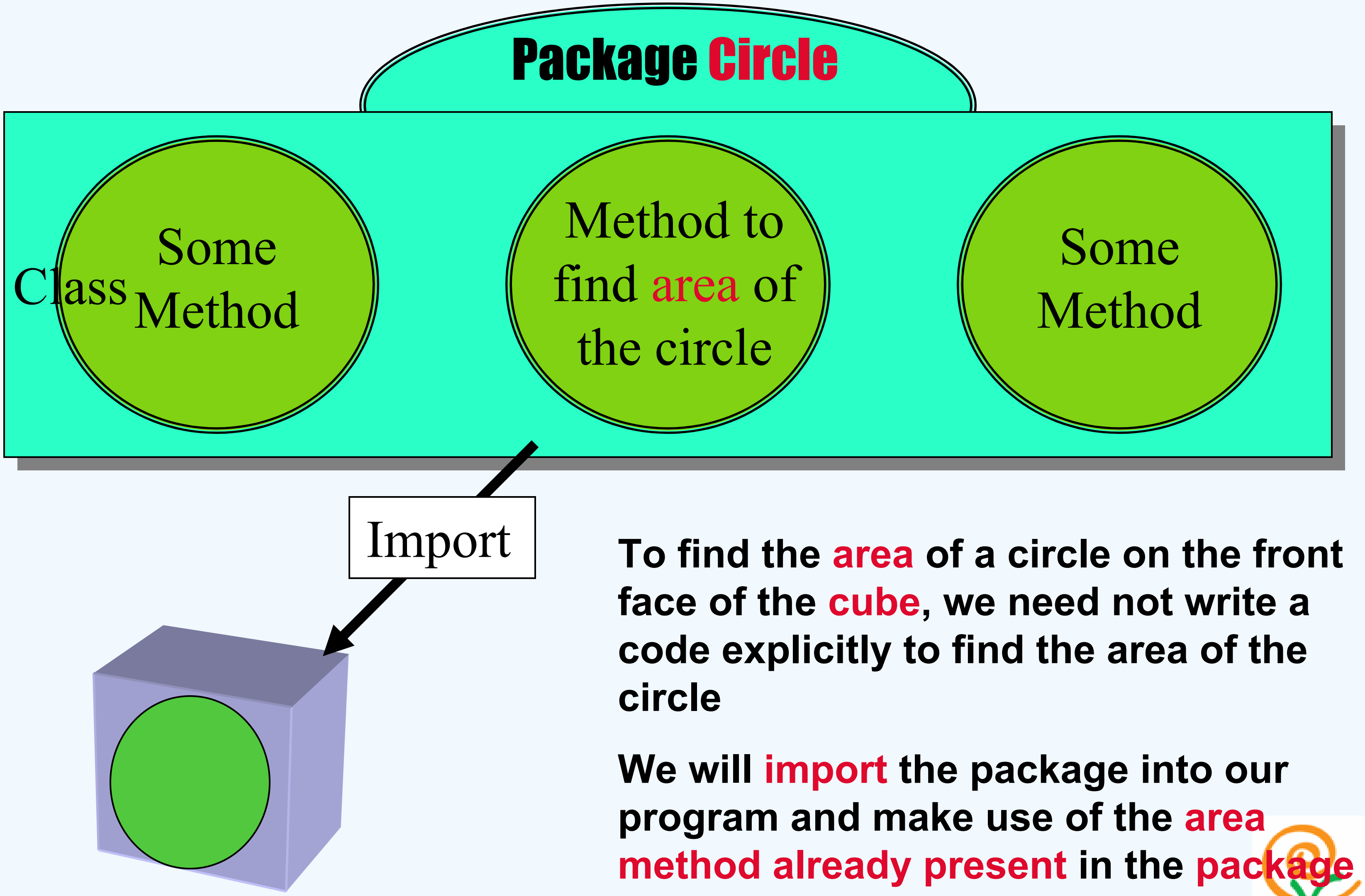
It helps to avoid naming conflicts. When you are working with a number of classes, it becomes difficult to decide on names of the classes & methods. At times you would want to use the same name, which belongs to an another class. Package, basically hides the classes and avoids conflicts in names.

Packages allow you to protect your classes, data and methods in a larger way than on a class-to-class basis.

Package names can be used to identify your classes.



An Example on the use of Packages



To find the **area** of a circle on the front face of the **cube**, we need not write a code explicitly to find the area of the circle

We will **import** the package into our program and make use of the **area method already present** in the **package** "**circle**".

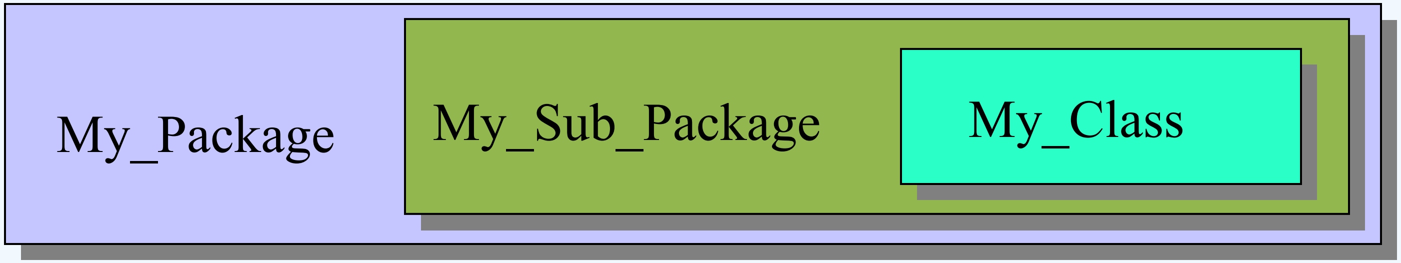
Importing a Package

In Java, the Packages (where the required method is already created) can be imported into any program where the method is to be used.

We can import a Package in the following manner :

import package_name . class_name ;

Suppose you wish to use a class say My_Class whose location is as follows :



My_Package

My_Sub_Package

My_Class

This class can be imported as follows :

import My_Package . MySub_Package . My_Class ;

Creating a Package

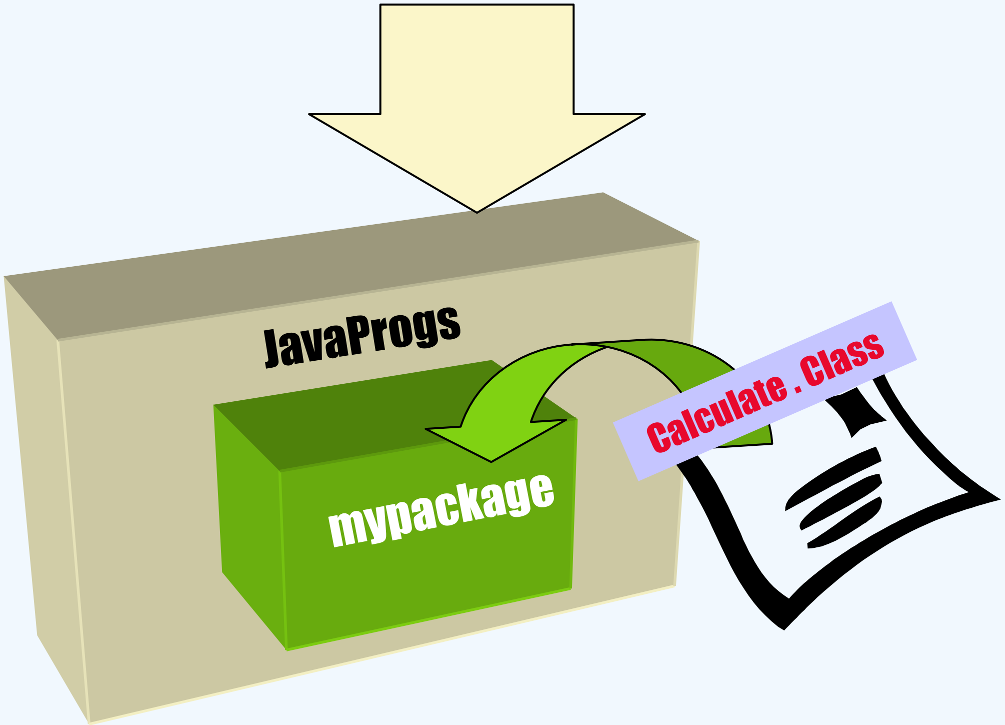
In Java Packages are created in the following manner :

Package package_name ;

```
package mypackage ;
public class calculate
{
      public int add(int x, int
      y)
      {
         return( x + y ) ;
      }
}
```

Compiling the package

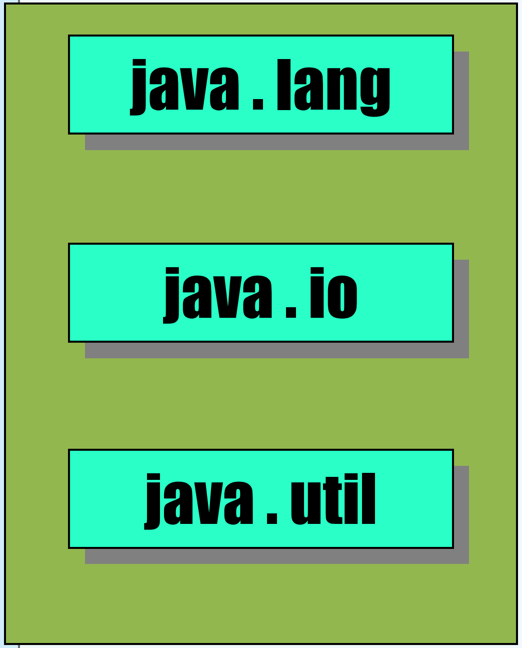
```
javac -d c:\ JavaProgs Calculate.java
```



When the above command is executed on the command prompt, the compiler creates a folder called “mypackage” in our JavaProgs directory and stores the “Calculate.class” into this folder

Standard Java Packages

The Three Java Packages that are essential to any Java program are :



java .lang

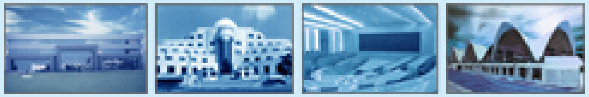
Contains classes that form the basis of the design of the programming language of Java

java .io

The use of *streams* for all input output operations in Java is handled by the java.io package

java . util

Contains classes and interfaces that provide additional utility but may not be always vital.



java.lang package

One of the most important classes defined in this package is Object and it represents the root of the java class hierarchy.

This package also holds the "wrapper" classes such as Boolean, Characters, Integer, Long, Float and Double.

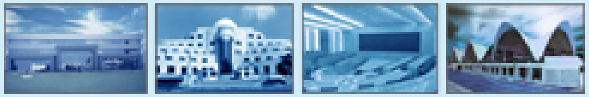
Many a times it is necessary to treat the non-object primitive datatypes of int, char, etc. as objects.

Thus Java defines "wrapper" classes that enable us to treat even primitive data types as objects.These wrapper classes are found in the package "java.lang".

Other classes found in this package are :

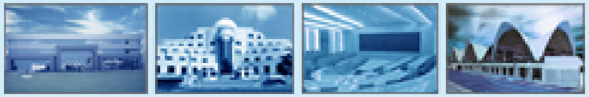
Math – which provides commonly used mathematical functions like sine, cosine and square root.

String & String Buffer – Encapsulate commonly used operations on character strings.



Some of the important methods of Math class

- int abs(int i) -- returns the absolute value of I
- long abs(long l) -- returns the absolute value of l
- float abs(float f) -- returns the absolute value of f
- double abs(double d) -- returns the absolute value of d
- double ceil(double d) -- returns as a double the smallest integer that is not less than d
- double floor(double d) --- returns as a double the largest integer



java.io package

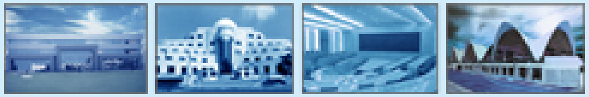
This package has two very important abstract classes :

Input Stream – This class defines the basic behavior required for input.

Output stream – This class is the basis of all the classes that deal with output operations in Java.

Since these are abstract classes, they cannot be used directly but must be inherited, so that the abstract methods can be implemented.

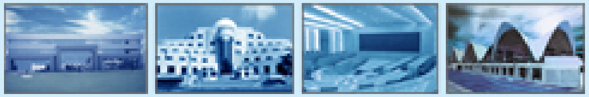
All I/O stream classes are derived from either of these classes.



java.io package

- The classes derived in Inputstream and Outputstream can only read from or write to the respective files.
- We cannot use the same class for both reading and writing operations.

  An exception to this rule is the class “RandomAccessFile”.
- This is the class used to handle files that allow random access and is capable of mixed reading and writing operations on a file.
- There are two additional interface to this package :
  - Data input
  - Data output
- •These classes are used to transfer data other than bytes or characters



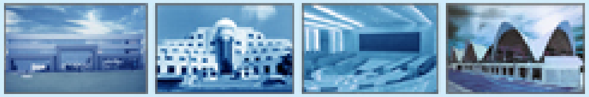
Java.util package

- One of the most important package in this package is the class “**Date**”, which can be used to **represent** or **manipulate** date and time information.

- In addition, the class also enable us to **account** for **time zones** .

- Java helps us to **change** the **size of an array** which is usually **fixed**, by making use of the class “**Vector**”. This class also enable us to **add**, **remove** and **search** for items in the array.

Education and Research



Tips on using packages

- The statement :

  import java.awt.* ;

- Will include all the classes available in the “awt” subdirectory present in the java directory.
- While creating a package, care should be taken that the statement for creating a package must be written before any other import statements

LEG ✓

```
package mypackage ;

import java . io;
```

ILLEGA ✗

```
import java . io;

package mypackage ;
```

Important Packages in Java

java.lang → You don’t need to explicitly import this package. It is always imported for you.

java.io → This package consists of classes that help you for all the Input and Output operations.

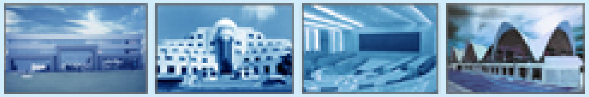
java.applet → This package consists of classes that you need, to execute an applet in the browser or an appletviewer.

java.awt → This package is useful to create GUI applications.

java.util → This package provides a variety of classes and interfaces for creating lists, calendar, date, etc.

java.net → This package provides classes and interfaces for TCP/IP network programming.

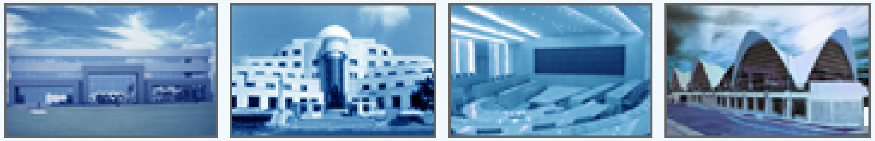
Education and Research



Infosys

Infosys®

POWERED BY INTELLECT
DRIVEN BY VALUES

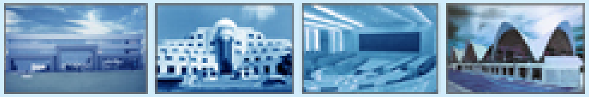


Declaring and Access Control

Education
and
Research

Arrays

- An array is a data structure which defines an ordered collection of a fixed number of homogeneous data elements
- The size of an array is fixed and cannot increase to accommodate more elements
- In Java, array are objects and can be of primitive data types or reference types
- All elements in the array must be of the same data type



Arrays

Declaring Arrays Variables

```
<elementType>[] <arrayName>;
```

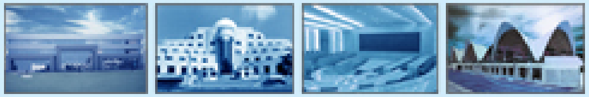
or

```
<elementType> <arrayName>[];
```

where <elementType> can be any primitive data type or reference type

Example:

```
int IntArray[];
Pizza[] mediumPizza, largePizza;
```



Arrays

Constructing an Array

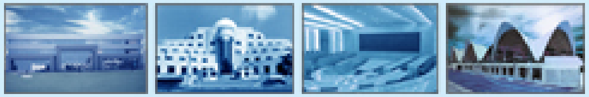
```
<arrayName> = new <elementType>[<noOfElements>];
```

Example:

```
IntArray = new int[10];
mediumPizza = new Pizza[5];
largePizza = new Pizza[2];
```

Declaration and Construction combined

```
int IntArray = new int[10];
Pizza mediumPizza = new Pizza[5];
```



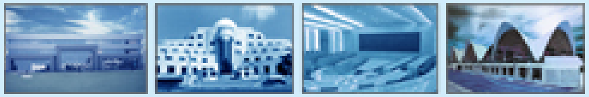
Arrays

Initializing an Array

```
<elementType>[] <arayName> = {<arrayInitializerCode>};
```

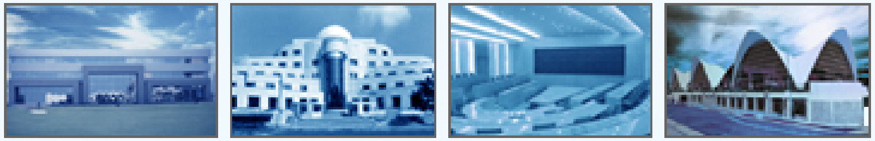
Example:

```
    int IntArray[] = {1, 2, 3, 4};

    char charArray[] = {‘a’, ‘b’, ‘c’};

    Object obj[] = {new Pizza(), new Pizza()};

    String pets[] = {“cats”, “dogs”};
```



Infosys®

POWERED BY INTELLECT
DRIVEN BY VALUES

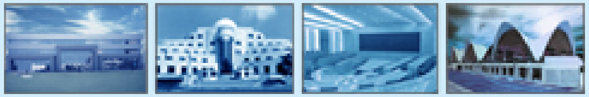


IO Facilities in Java

Education
and
Research

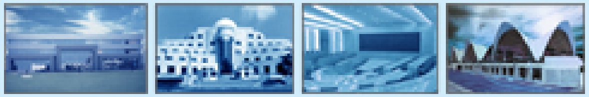
Overview

- IO Streams in Java
- Understanding some fundamental streams
- Creating streams for required functionality
- Some advanced streams

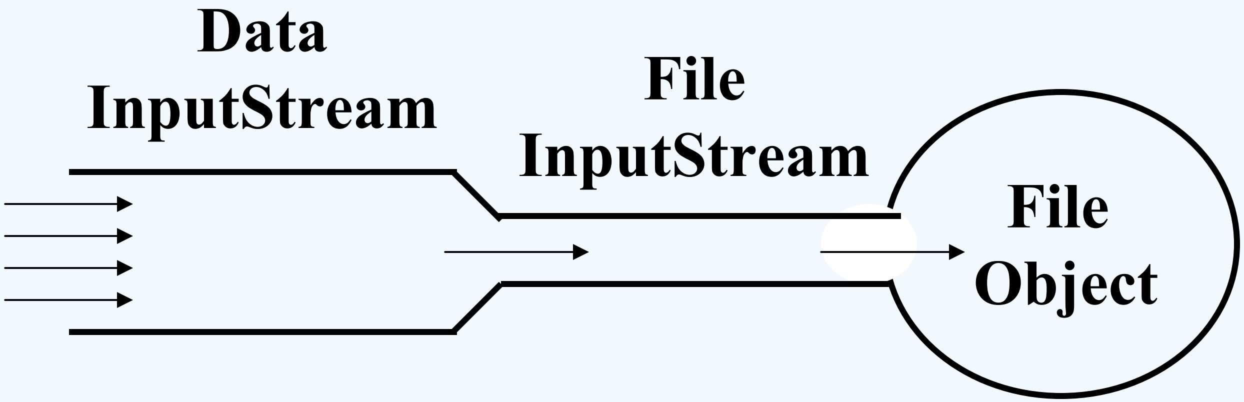


Streams

- Streams are channels of communication
- Provide a good abstraction between the source and destination
- Could also act as a shield to lower transport implementation
- Most of Java’s IO is based on streams
  - Byte-oriented streams
  - Character-oriented streams

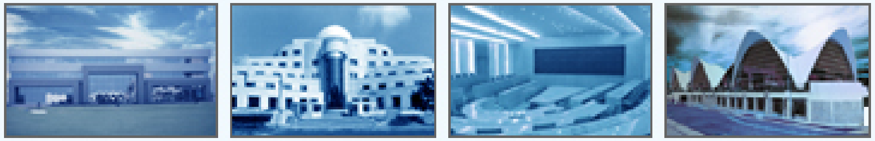


Concatenating Streams



Infosys®

POWERED BY INTELLECT
DRIVEN BY VALUES



Input & Output Streams

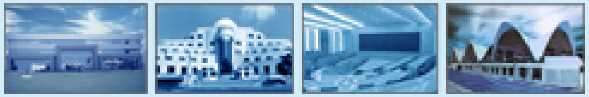
Education
and
Research

Streams

A stream can be thought of as a Conduit (pipe) for data between a source and the destination.

Two types of Streams are

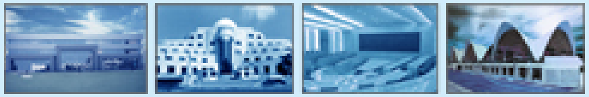
1. Low level streams
2. High level streams



Low level streams & High level streams

Streams which carries bytes are called low level streams. Examples are FileInputStream and FileOutputStream.

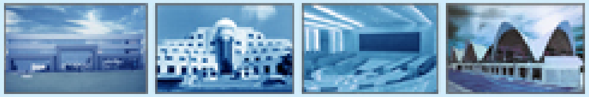
Streams which carries primitive data types are called high level streams. Examples are DataInputStream and DataOutputStream.

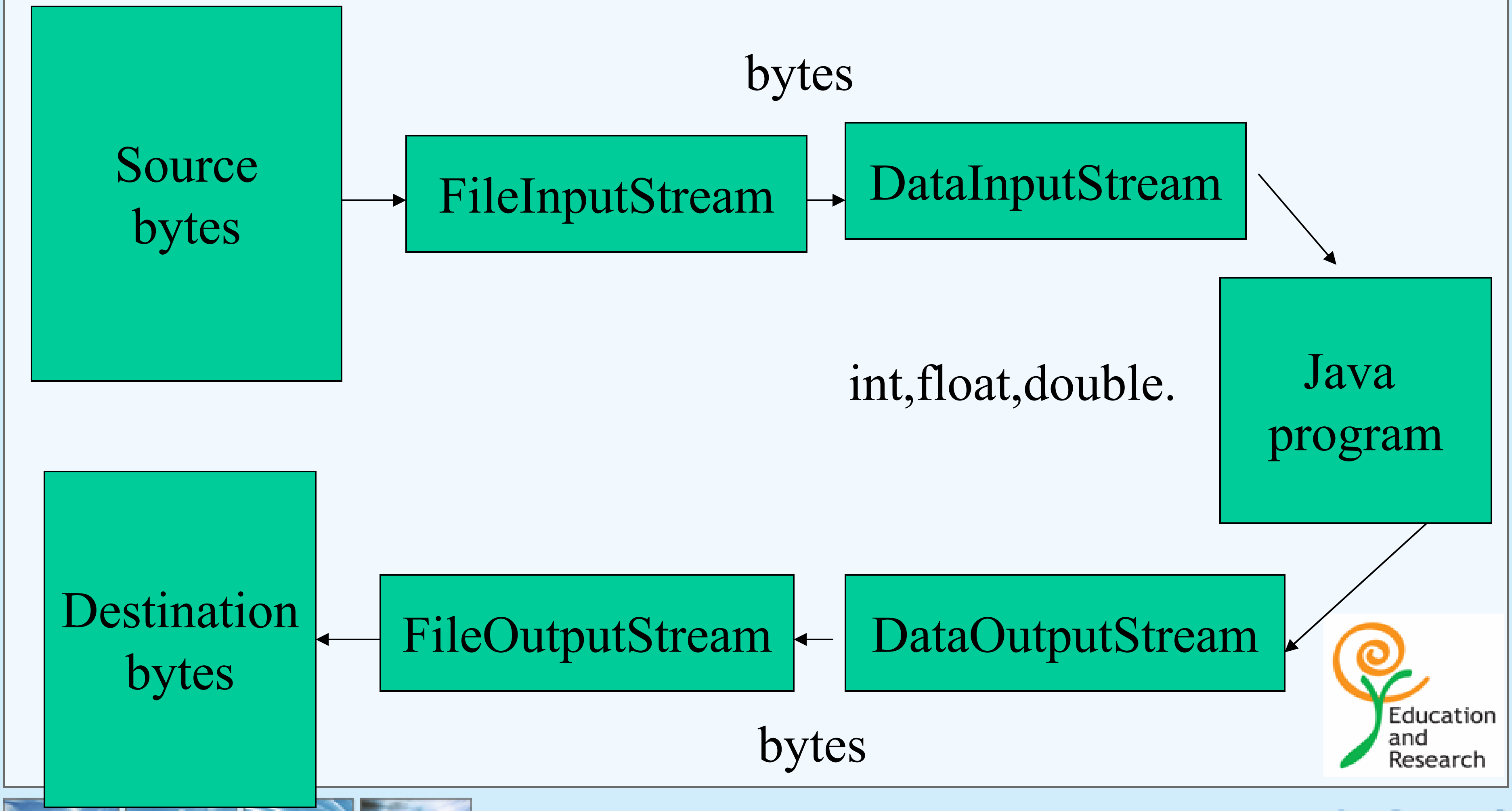


InputStream & OutputStream

InputStreams are used for reading the data from the source.

OutputStreams are used for writing the data to the destination.





bytes

Source bytes

FileInputStream

DataInputStream

Java program

int,float,double.

Destination bytes

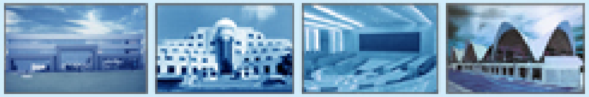
FileOutputStream

DataOutputStream

bytes

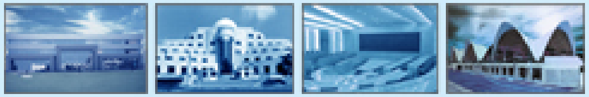
Writing Primitives to a File

```
DataOutputStream dos =
    new DataOutputStream( new
    FileOutputStream("item.dat"));
dos.writeFloat(itemPrice);
dos.writeInt(itemQty);
dos.writeChars(itemNo);
```



Filter Streams

- Filter contents as they pass through the stream
- Filters can be concatenated as seen before
- Some filter streams
  - Buffered Streams
  - LineNumberInputStream
  - PushBackInputStream
  - PrintStream

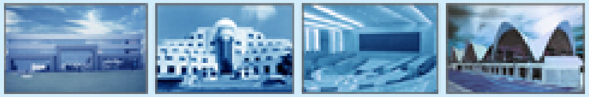


Conversion Streams

- **InputStreamReader: bridge from byte streams to character streams**

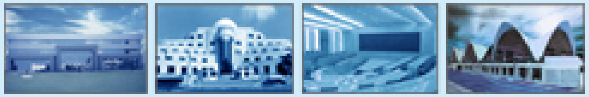
```
BufferedReader in = new
    BufferedReader( new
    InputStreamReader(System.in));
```

- **OutputStreamWriter: bridge from chararcter streams to byte streams**



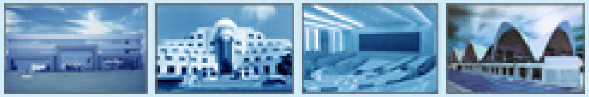
Review

- Streams are the basis of Java’s IO
- Pre-defined streams for many situations
- Streams need to be concatenated
- Conversion streams available for byte to character and vice-versa conversion

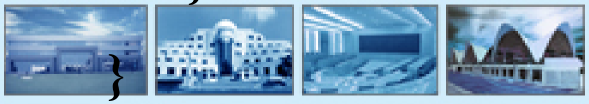


String class

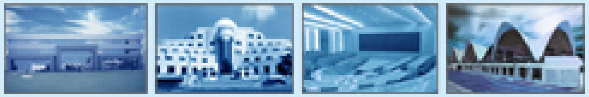
- A string is a collection of characters
- Has equals( ) method that should be used to compare the actual string values
- Lot of other methods are available which are for the manipulation of characters of the string



```
public class Stringcomparison
{
public static void main(String args[]) {
    String ss1=new String("Rafiq");
    String ss2=new String("Rafiq");
    String s1="Rafiq";
    String s2="Rafiq";
    System.out.println(" == comparison for StringObjects: "+(ss1==ss2));
    System.out.println(" == comparison for StringLiterals: "+(s1==s2));
    System.out.println(" equals( ) comparison for StringObjects:
"+(ss1.equals(ss2)));
    System.out.println(" equals( ) comparison for StringLiterals:
"+(s1.equals(s2)));
    }
}
```



```
class checkstring
{ public static void main(String args[]){
            String str="HELLO guys & girls";
            System.out.println("The String is:"+str);
            System.out.println("Length of the String is:"+str.length());
            System.out.println("Character at specified
    position:"+str.charAt(4));
            System.out.println("substring of the String
    is:"+str.substring(6,10));
            System.out.println("Index of the specified
    character:"+str.indexOf("g"));
            System.out.println("conversion to
    uppercase:"+str.toUpperCase());
            System.out.println("conversion to
    uppercase:"+str.toLowerCase());
    }
}
```



String Buffer

•The prime difference between String & StringBuffer class is that the stringBuffer represents a string that can be dynamically modified.

•StringBuffer's capacity could be dynamically increased eventhough it's capacity is specified in the run time.

Constructors

StringBuffer()

StringBuffer(int capacity)

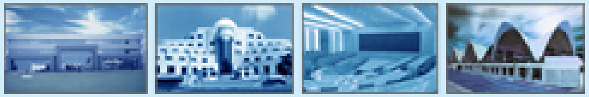
StringBuffer(String str)

Methods

int length()

int capacity()

void setLength(int len)



String Buffer

char charAt(int where)

void setCharAt(int where, char ch)

StringBuffer append(String str)

StringBuffer append(int num)

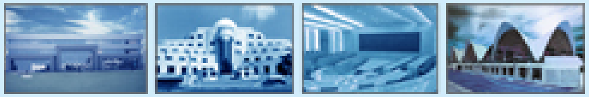
StringBuffer append(Object obj)

StringBuffer insert(int index,String str)

StringBuffer insert(int index,char ch)

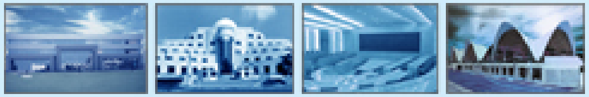
StringBuffer insert(int index,Object obj)

StringBuffer reverse()



java.util Package

- Has utility classes like Date
- Properties
- HashTable
- Vector e.t.c

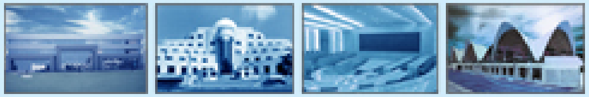


Vector

```
Vector( )
Vector(int size)
Vector(int size, int incr)
```

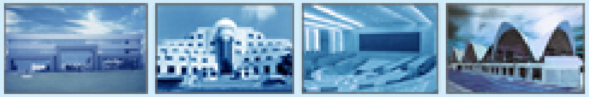
The following are few Vector methods:

```
final void addElement(Object element)
final int capacity()
final boolean contains(Object element)
final Object elementAt(int index)
final Object firstElement()
```



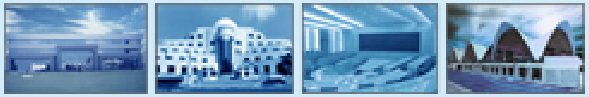
Vector

```
final void insertElementAt(Object element, int index)
final Object lastElement()
final boolean removeElement(Object element)
final void removeElementAt(int index)
final int size()
```



Date

```
import java.util.Date;
class DateDemo{
 public static void main(String args[])
   {
     //Instantiating a Date Object
       Date date=new Date();
     System.out.println("current date is"+date);
     System.out.println("current day is"+date.getDay());
     System.out.println("current month is"+date.getMonth());
     System.out.println("current Year is"+date.getYear());
     long msec=date.getTime();
     System.out.println("Milliseconds since Jan. 1,1970 ="+msec);
   }
}
```



Applications and Applets (The difference)

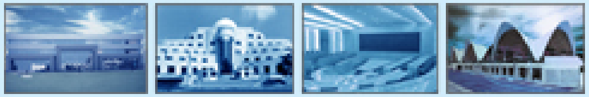
These are some of the difference between an Application and an Applet :

Java Application	Java Applet
1. Standalone program.	1. Used for Internet programming.
2. Applications can run by itself.	2. Applications cannot run by itself and requires a Browser software to run it.
3. Since it can be run independently, so utilities like event handling, user interface, etc. should be explicitly written by the programmer.	3. Since applets runs inside the Browser, it enjoys all the inbuilt facilities of some event handling.
. No Control over the low of the program	. There is some control over the execution of the program.

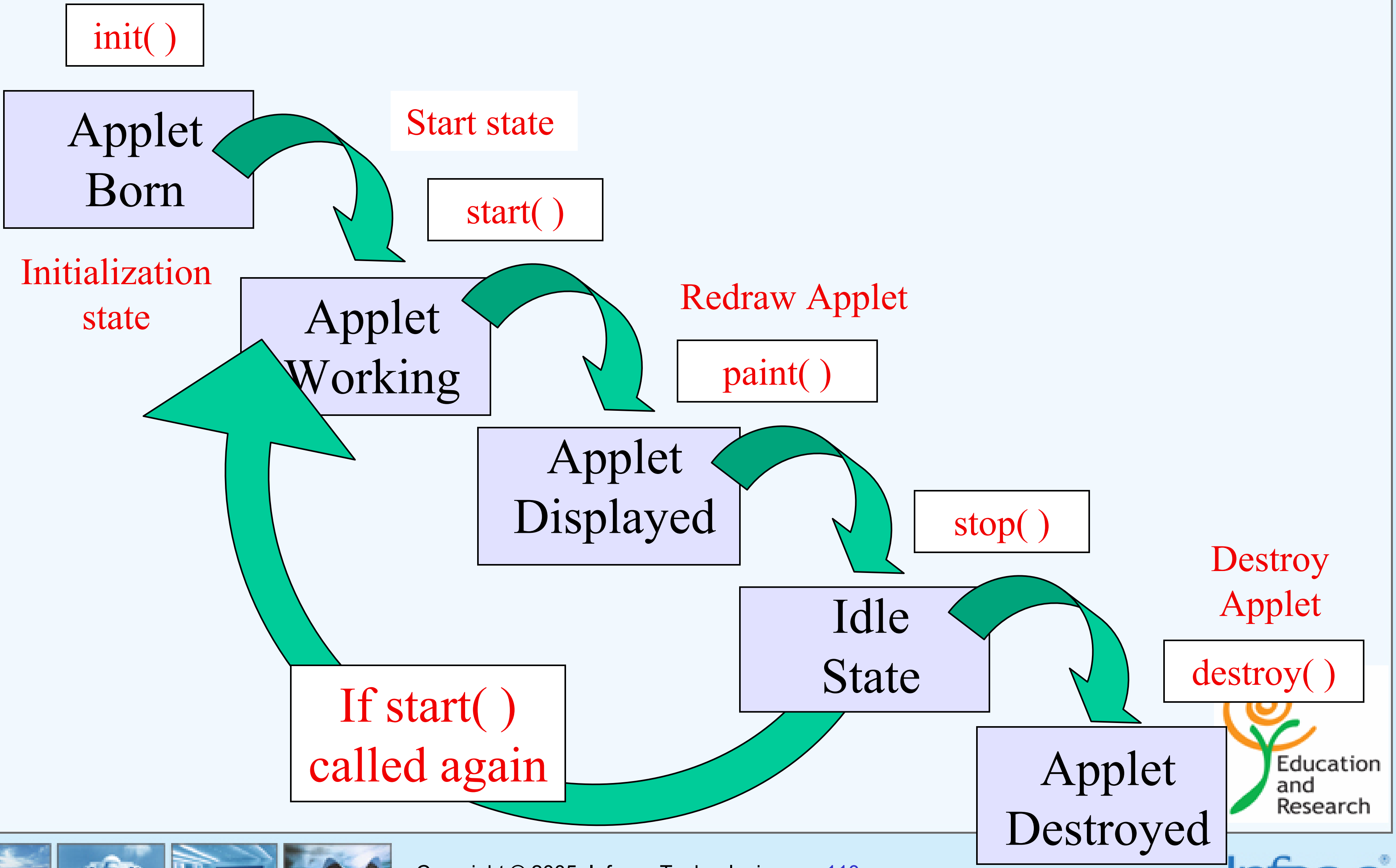
Applet

- What is an applet?
- A small Java program that runs on a Browser
- The applet runs on JVM embedded in the browser

-
  ```
  import java.applet.Applet;
  import java.awt.Graphics;

  public class HelloWorld extends Applet {
      public void paint(Graphics g) {
          g.drawString("Hello world!", 50, 25);
      }
  }
  ```



Life Cycle of an Applet



init( )

Applet Born

Initialization state

Start state

start( )

Applet Working

Redraw Applet

paint( )

Applet Displayed

stop( )

Idle State

Destroy Applet

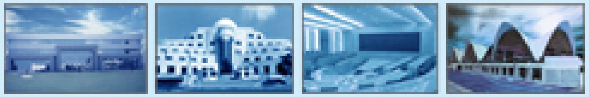
destroy( )

Applet Destroyed

If start( ) called again

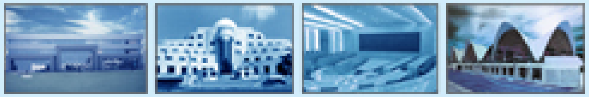
Applet Life Cycle..

- The Browser creates an Object of the applet and then it calls a sequence of methods on that object.
- The first method to be called is init()
- Then the start() method is called.
- Then repaint().
- Finally to remove destroy() is called.
- So there is no need for a public static void main in the applet to work because the browser creates the object and calls the method on that.



Life cycle Applet...

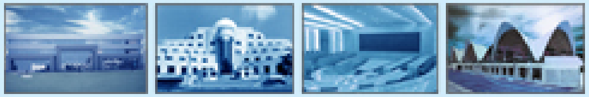
- Every time you move out of the HTML that contains your applet and then come back the start() is called.
- init() is called only once.
- When ever you lose scope and return the paint() is called.
- All these calls are done by the browser(not by you)



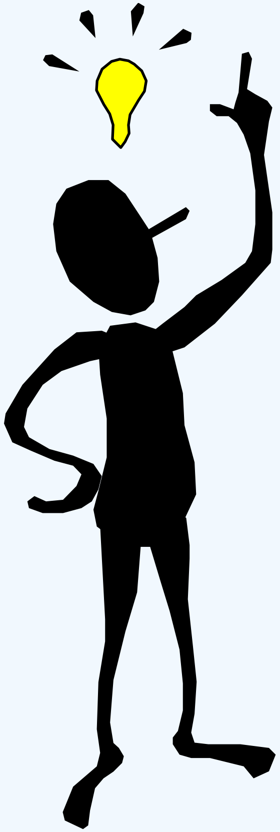
Applet………

- You need to override only the methods you want, the others are redirected to the base class.
- If you need to repaint the applet display you can do it by calling repaint().(You are not supposed to call directly-

  `public void paint(Graphics g)`)
- Why?????????????



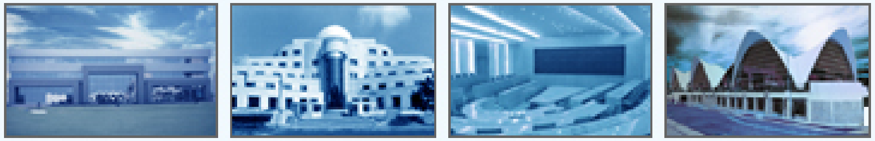
Some code review..



- We used import statements
- We inherited from Applet class
- We did some overriding
- We called some behavior
- We learnt to embed Java programs in HTML

Infosys®

POWERED BY INTELLECT
DRIVEN BY VALUES



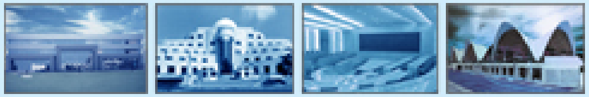
Exception Handling

Education
and
Research

Exceptions and Errors

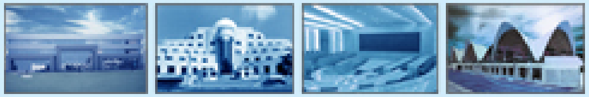
- Exceptions are situations within the control of an application, that it should try to handle

- Errors indicate serious problems and abnormal conditions that most applications should not try to handle



Overview

- What is an Exception?
- Errors and Exceptions
- The try-catch-finally block(s)
- Catching multiple exceptions

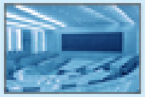


Exceptions

- An exception is an event that occurs during the execution of a program that disrupts the normal flow of instructions.

- The exception object contains information about the exception, including its type and the state of the program when the error occurred.

    - Helpful in separating the execution code from the error handler

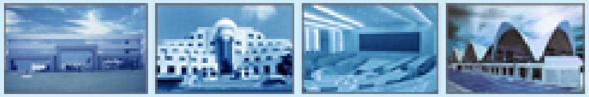


Exception’s

- The Java programming language provides a mechanism known as exceptions to help programs report and handle errors.

- When an error occurs, the program throws an exception.

- It means that the normal flow of the program is interrupted and that the runtime attempts to find an exception handler--a block of code that can handle a particular type of error



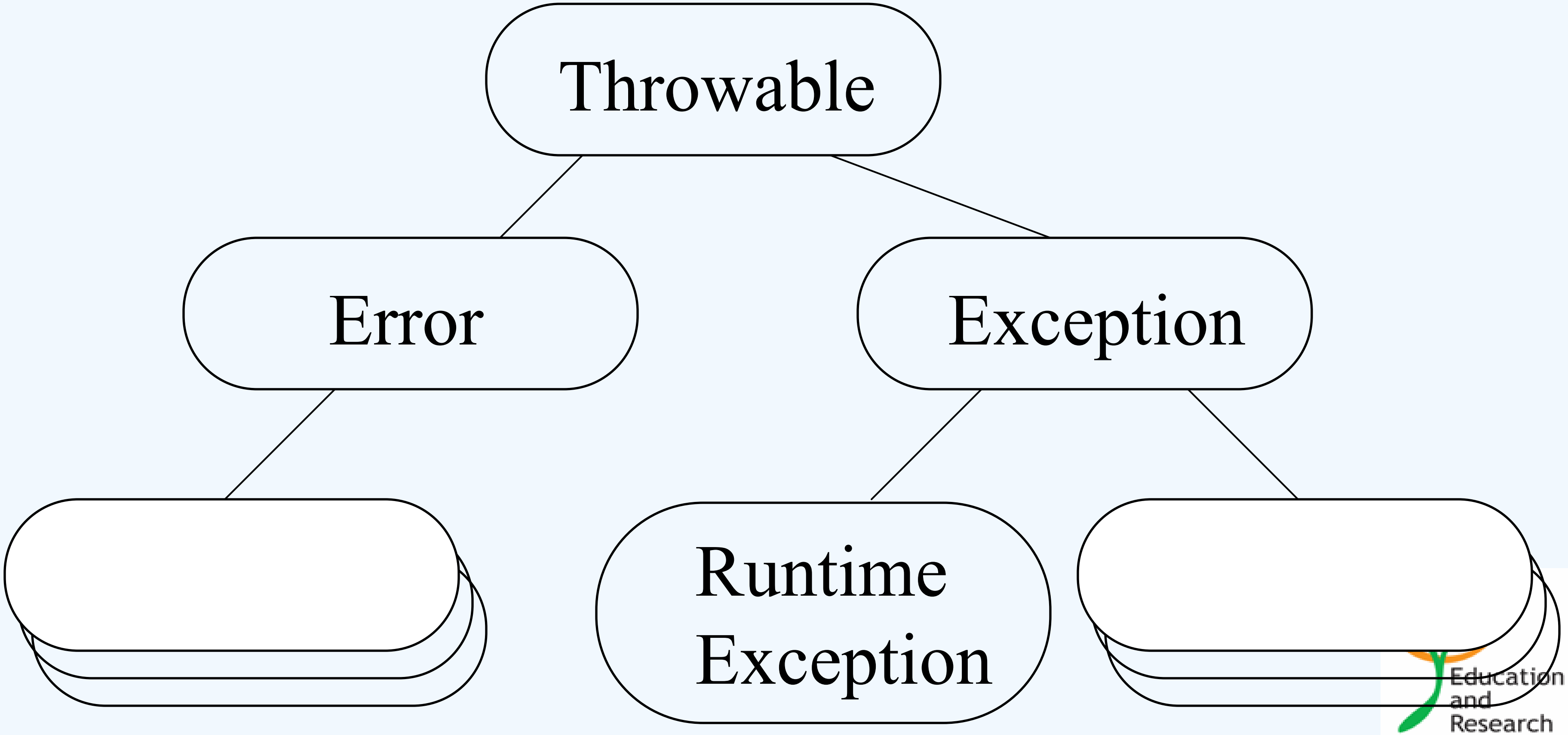
Exceptions

- The exception handler can attempt to recover from the error or, if it determines that the error is unrecoverable, provide a gentle exit from the program.

- Three statements play a part in handling exceptions: try-catch-finally

- Java enforces exceptions to be handled by the programmer, resulting in compile errors otherwise

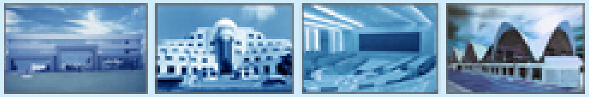
Java’s Exception Hierarchy



Try catch -finally

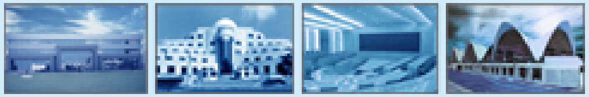
- The finally statement is associated with a try statement and identifies a block of statements that are executed regardless of whether or not an error occurs within the try block.

```
try {        statement(s)
     }
catch (exceptiontype name) {
     statement(s)        }
finally {           statement(s)
            }
```



The “finally” Block

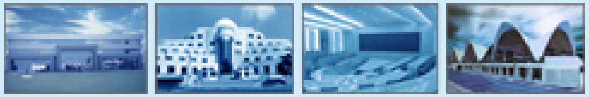
- Defines the code that is executed always
- In the normal execution it is executed after the try block
- When an exception, it is executed after the handler or before propagation as the case may be



Throwing Exceptions

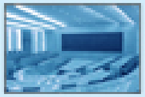
- Exceptions in Java are compulsorily of type Throwable
- Use the throw clause to throw an exception
- Can also be used to rethrow an exception

```
public void read() throws IOException
{
     // Some code that cause IO Exception
     throw new IOException();
}
```



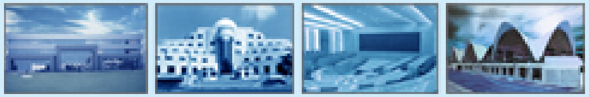
Some Java Exceptions

- ArithmeticException
- ClassCastException
- IllegalStateExecption
- IndexOutOfBoundsException
- InstantiationException
- NullPointerException
- SecurityException



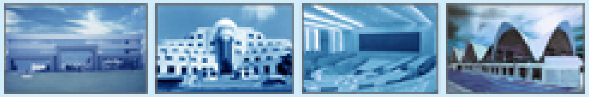
Some Java Errors

- ClassFormatError
- InternalError
- LinkageError
- OutOfMemoryError
- StackOverflowError
- VirtualMachineError
- UnknownError



Throws-Throw

- These are two key words that you may use
- The Throws keyword is used along with the declaration of a method that can throw an exception.
- This makes it mandatory for anyone calling the method to have it in try block.
- Else the compiler will give an error.

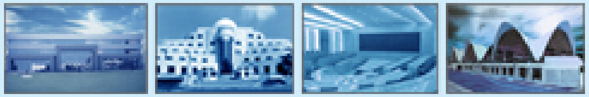


Throw

- All Java methods use the throw statement to throw an exception.

- The throw statement requires a single argument: a *throwable* object.

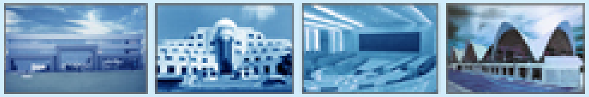
- If you attempt to throw an object that is not throwable, the compiler refuses to compile your program



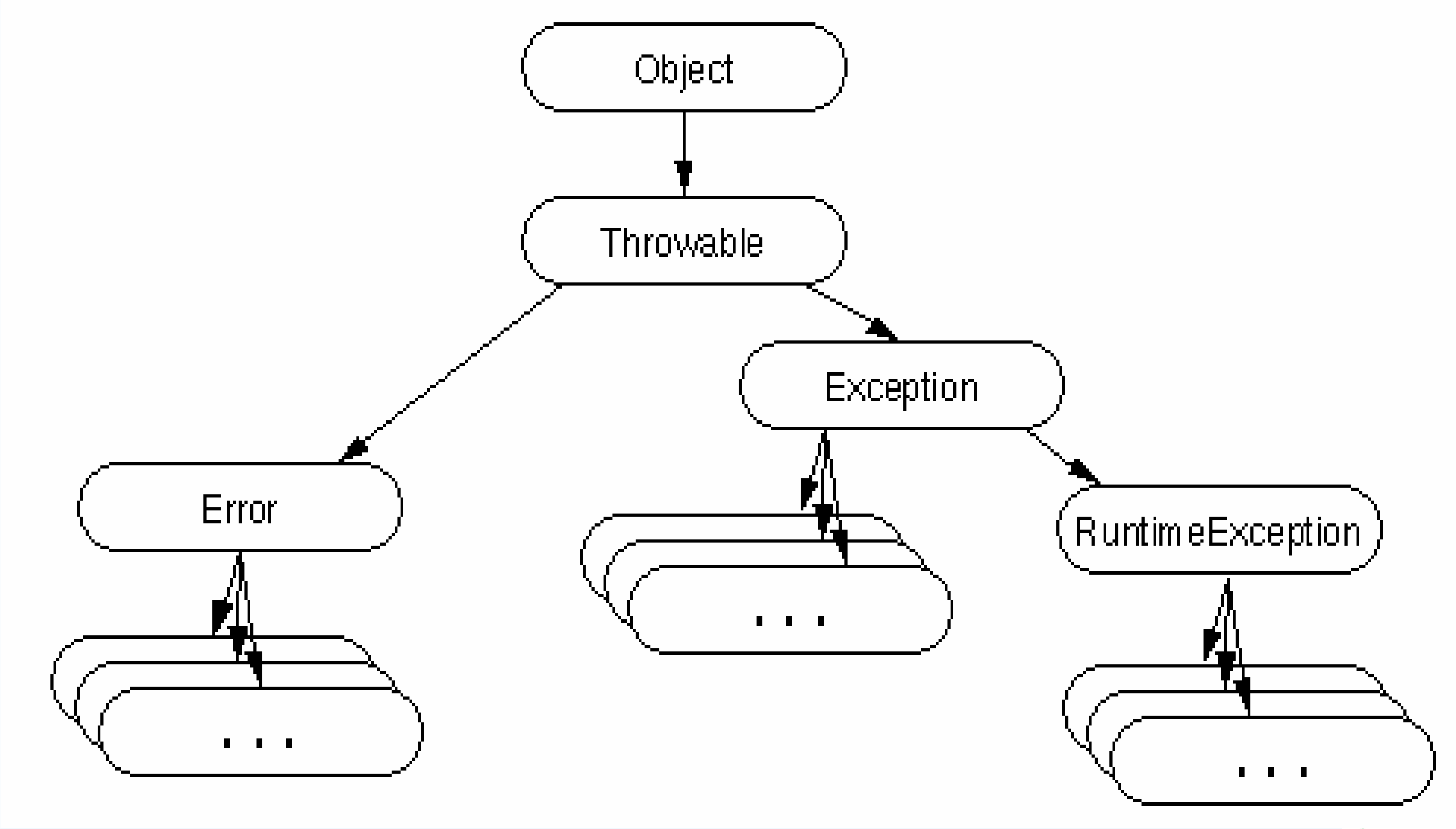
Throw

- It is used as follows- in a case where you want to throw an exception

```
if(accountIsValid()){
                   continue}
            else{
            throw new InvalidAccontException();
            }
```



The Hierarchy



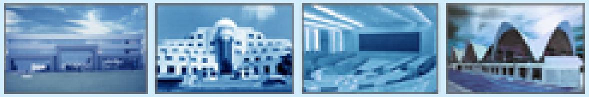
cation

earch

Errors

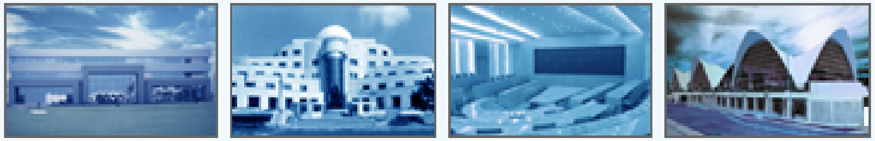
- When a dynamic linking failure or some other "hard" failure in the virtual machine occurs, the virtual machine throws an Error.

- Typical Java programs should not catch Errors. In addition, it's unlikely that typical Java programs will ever throw Errors either.



Infosys®

POWERED BY INTELLECT
DRIVEN BY VALUES



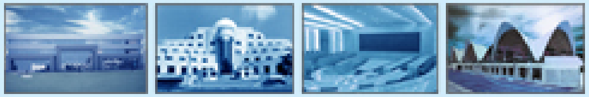
GUI Programming using AWT

Education and Research

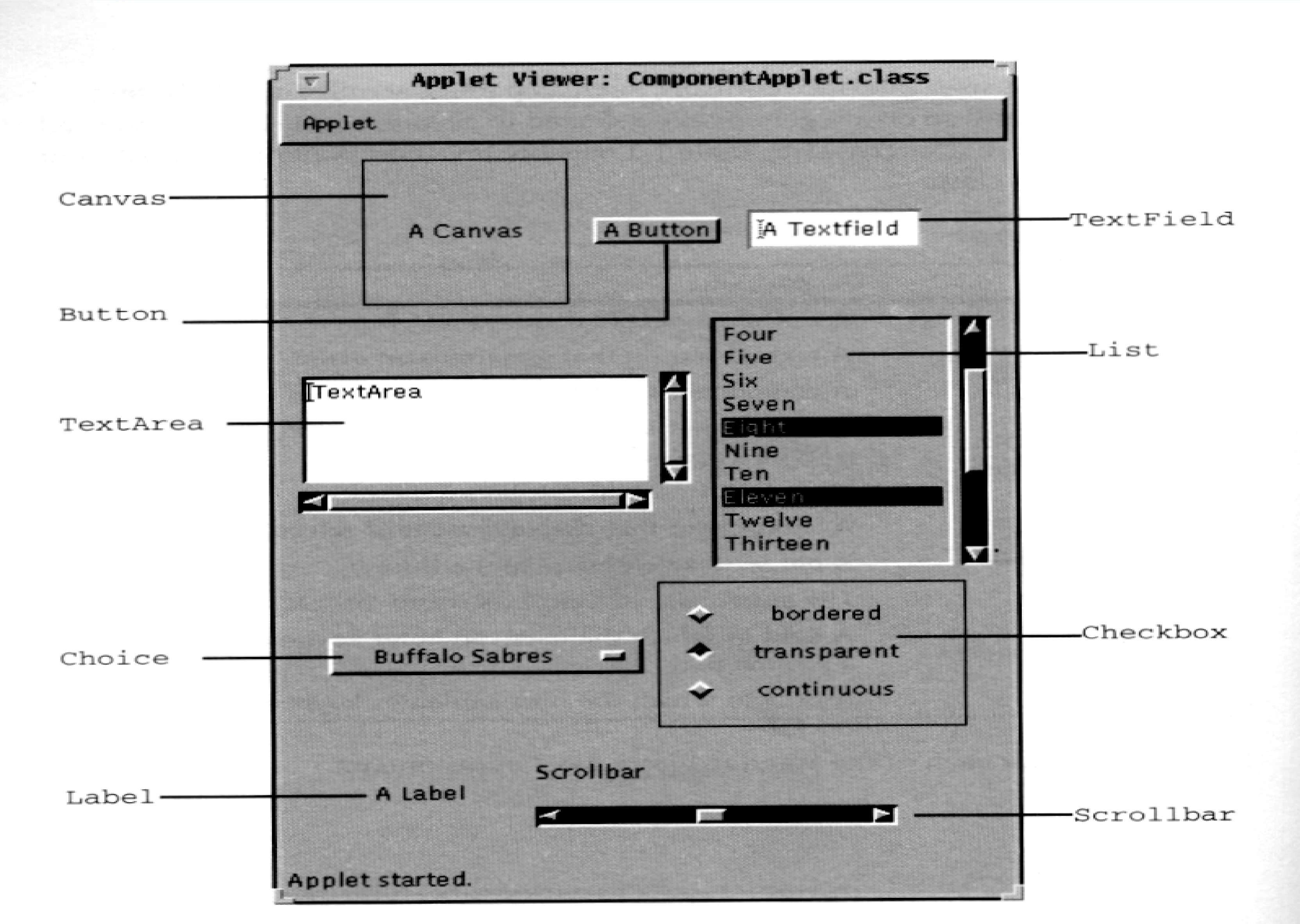
JAVA GUI Components

- AWT provides us with predefined classes and methods to help in creating various GUI components.
- Following are the various GUI components available in the ‘java.awt’ package :

Button	: push button
Canvas	: drawing surface
Checkbox	: option button
Choice	: combo box
Label	: static text
List	: list box
Scrollbar	: scrolling capability
TextComponent	: base class for text components
Container	: base class for all GUI containers

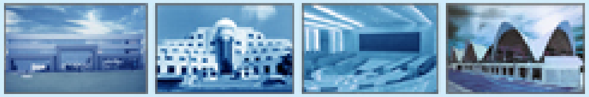


Java AWT components



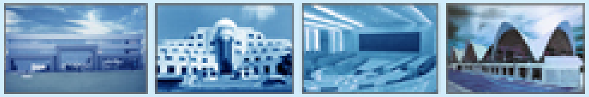
Building applications with graphical user interfaces

- Creating the interface
    - Creating the window
    - Adding components
    - Layout
    - Another Container
- Defining the behavior
    - WindowEvent and WindowListener
    - ItemEvent and ItemListener



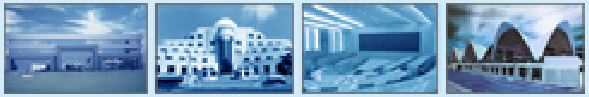
AWT hierarchy

- Component
  - Button
  - Canvas
  - Checkbox
  - Choice
  - Container
    - Panel
    - ScrollPane
    - Window
      - Dialog
        - FileDialog
      - Frame
  - Label
  - List
  - Scrollbar
  - TestComponent
    - TextArea
    - TextField



Creating the window

- Use Containers
  - containers hold the Components and helps to organize the components into manageable groups.
  - provides the basic window and dialog services.
  - top-level windows are represented by the ***Frame*** class.

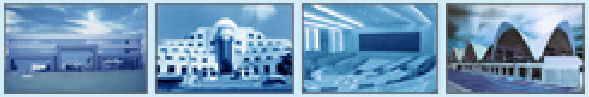


Defining the types of container

- *Frame* : It is a fully functioning window with its own title and icons.

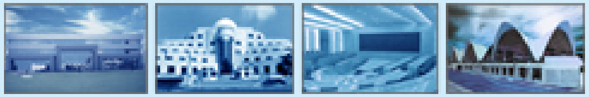
Other containers:

- *Panel* : A pure container and not a window in itself. Sole purpose is to organize the components on to a window.
- *Dialog* : pop-up window that pops out when an error message has to be displayed. Not a fully functioning window like the Frame.

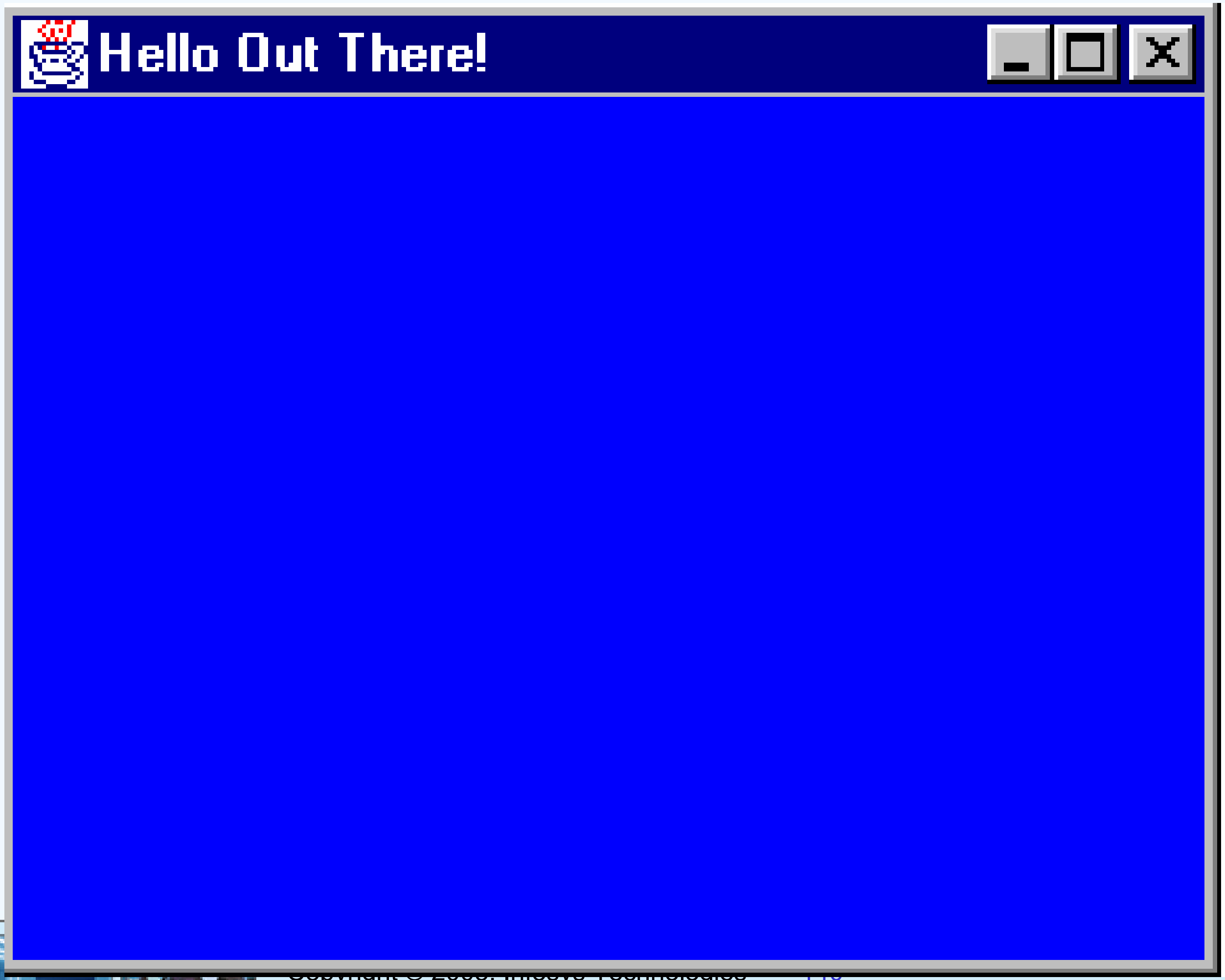


Frames

- Are subclasses of Window
- Have title and resize corner
- Inherit from Component class and add components with the *add* method
- Have Border Layout as the default layout manager
- Use the setLayout method to change the default layout manager

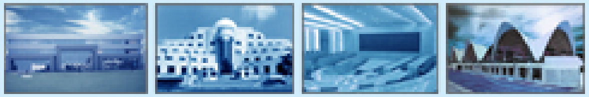


Frames

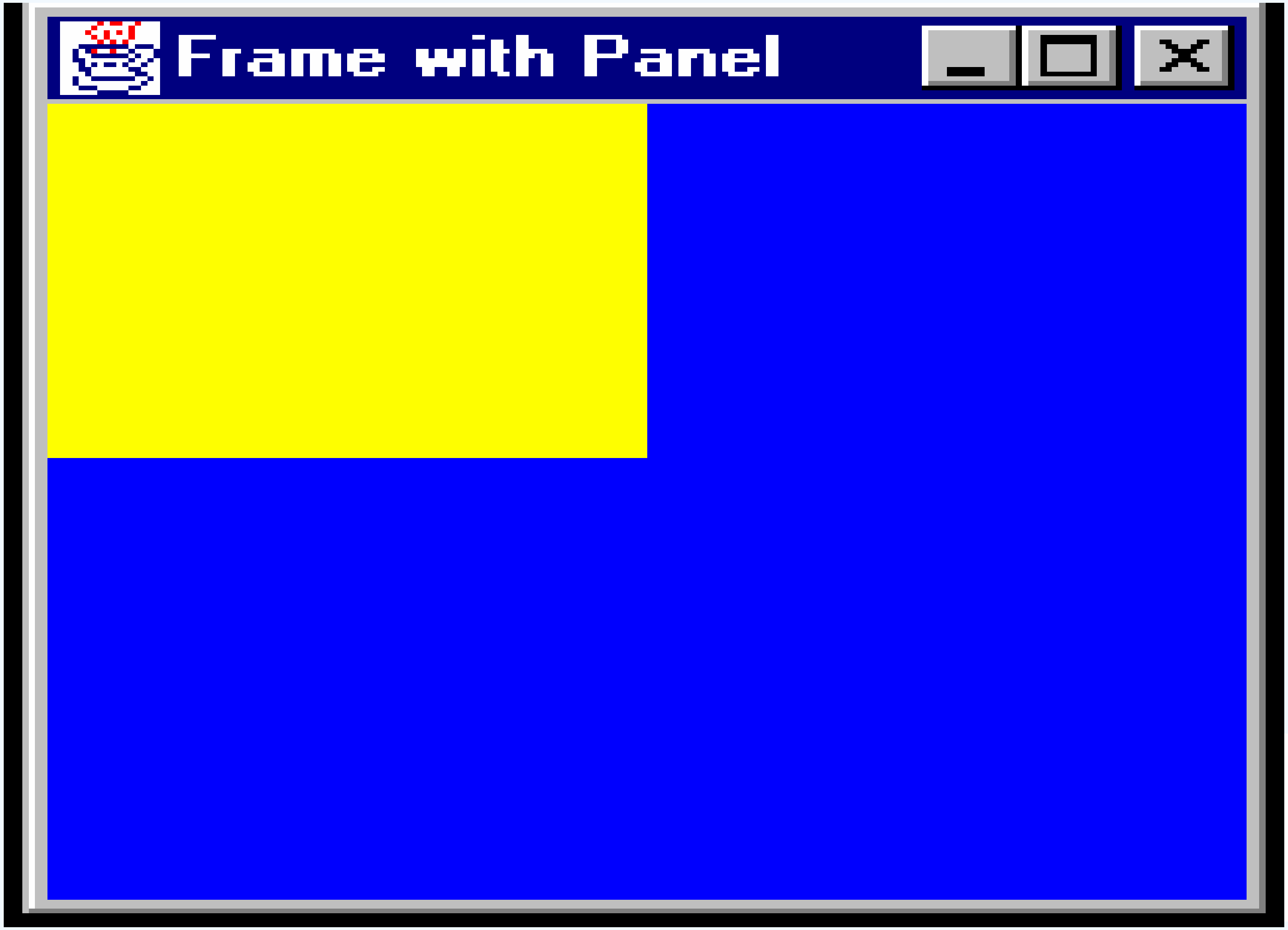


Panels

- Provide a space for components
- Allow subpanels to have their own layout manager
- Add components with the add method
- Default layout manager is the FlowLayout layout manager

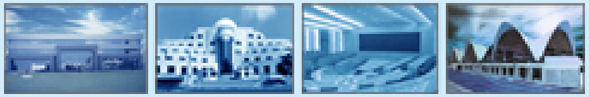


Panel



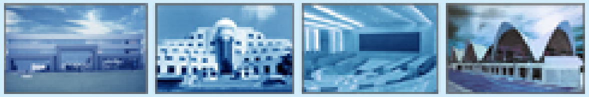
Adding components

- Components need to be added to a window
  - Create an instance
  - Add it to a window by calling add()
  - Removing with remove()
- Except labels all other components generate events when accessed.
  - You need to handle the events



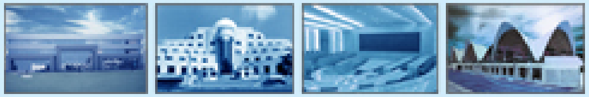
Layout Managers

- Used to display the components on the target screen.
- Java being platform independent, employs a unique way to display the components on the screen, independent of the platform.
- Java provides five different ways of sectioning the display area.
- Each of these ways of displaying components on different screen sections is handled by the Layout Manager.



Layout managers

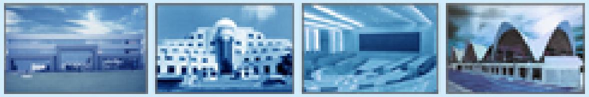
- The five Layout managers available are :
  - Flow Layout
  - Grid Layout
  - Border Layout
  - Card Layout
  - GridBag Layout



Flow Layout Manger

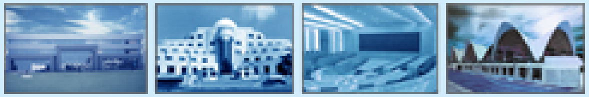
- Arranges components from left-to-right in top-to-down fashion.

1. The first component is placed at the Top-left corner.
2. The successive components will be placed next to the one before it till a border of the display is encountered.
3. The remaining components will be displayed in the next row in a similar fashion.



Flow Layout Manager

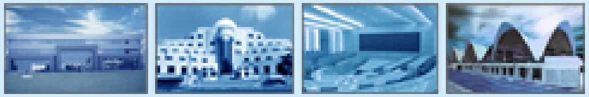
4. The horizontal alignment of components is possible
   - The options available for horizontal alignment are :
     - Left
     - Right
     - Center
   - By default, the components are center aligned.
5. It is also possible to specify the vertical and horizontal spacing between the components.



Grid Layout Manager

1. The display area is divided into a grid composing of rows and columns and further into number of cells.
2. Components are placed in the cells one after another in a row-wise fashion.
3. Relative placement of the components remain the same irrespective of the size of the Applet.
4. Possible to vary the space between components placed on a Grid Layout.

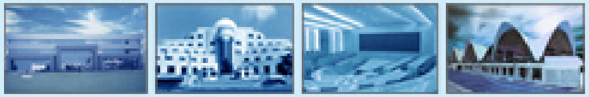
***The dimension of Applets in the HTML page do not affect the placement of components in the GridLayout*.**



Border Layout Manager

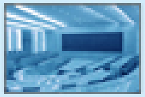
1. **It uses the Graphic directions of East, West, North, South and Center.**
2. **The components are arranged along the borders of the Layout area.**
3. **The space left in the center is given to the component with center as its position.**

- ***The Border Layout manager is the default layout manager for Dialog and Frame.***



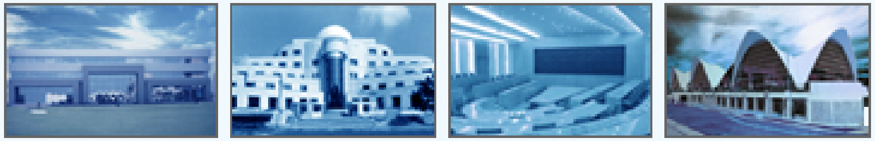
Card Layout Manager

1. The Card Layout components are arranged into individual cards.
2. All the components are not visible at the same time. These cards can only be viewed one at a time.
3. In Card Layout, the components are placed in different Panels.



GridBag Layout Manager

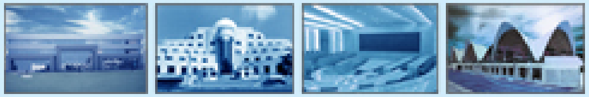
- Most powerful Layout Manager.
- Arranges the components in Grids.
- Most complex of all Layout Managers.
- Most flexible of all the five Layout Managers available.
- Some controls provided by GridBag Layout Manager are :
  - Span of Cells.
  - Arrangement of Components in the cells.
  - Space proportions between rows and columns.
- *These controls are managed by the class called GridBagConstraints.*



Event handling

Events

- An *event* is an object that represents some activity to which we may want to respond
- Example:
  - a mouse is moved
  - a mouse button is clicked
  - a mouse is dragged
  - a graphical button is clicked
  - a keyboard key is pressed
  - a timer expires
- Often events correspond to user actions, but not always



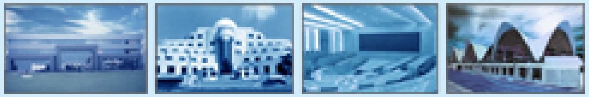
Events…

- The Java standard class library contains several classes that represent typical events

- Certain objects, such as an applet or a graphical button, generate (fire) an event when it occurs

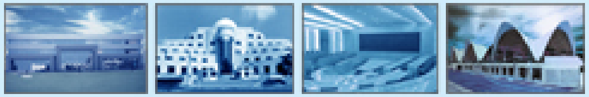
- Other objects, called *listeners*, respond to events

- We can write listener objects to do whatever we want when an event occurs



Events…

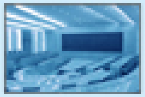
- The java.awt.event package defines classes to different type of events.
- Events correspond to :
  - Physical actions (eg.,mouse button down, Key press/release)
  - Logical events (e.g. gotfocus - receiving focus on a component)



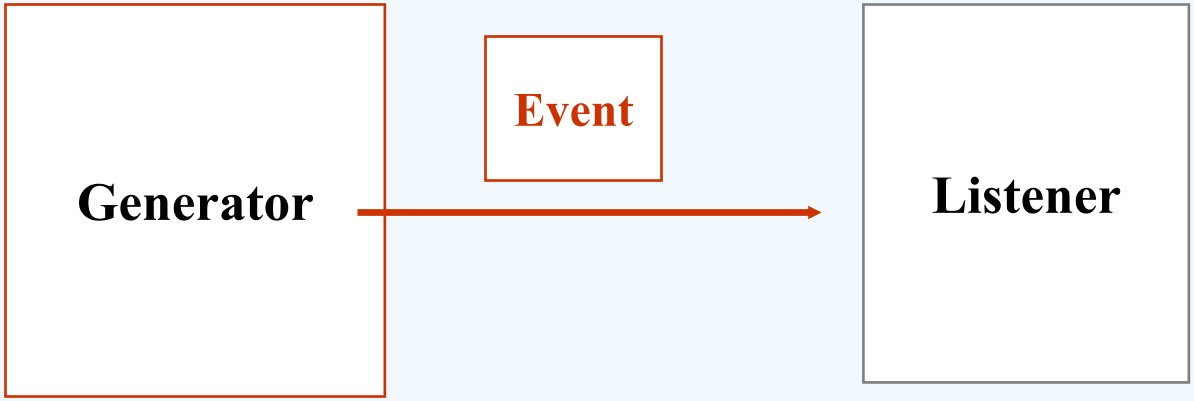
Some Event Classes

Present in java.awt.event class

- ***ActionEvent* – a button pressed, a menu item selected, etc.**
- ***ItemEvent* – a check box or list item is clicked, etc.**
- ***ComponentEvent* – a component is hidden, moved, resized, or becomes visible**
- ***KeyEvent* – keyboard events like key pressed, released, etc**
- ***MouseEvent* –mouse clicked, dragged, etc**
- ***WindowEvent* – minimized, maximized, resized, etc**



Events and Listeners



This object may generate an event

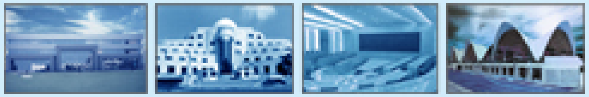
This object waits for and responds to an event

When an event occurs, the generator calls the appropriate method of the listener, passing an object that describes the event

Delegation Event Model

- **Events are fired by event sources.**
- **An event listener registers with an event source and receives notifications about the events of a particular type.**

- ***The java.awt.event package defines events and event listeners, as well as event listener adapters***



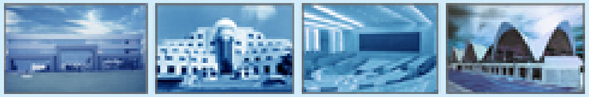
Listener Interfaces

- We can create a listener object by writing a class that implements a particular *listener interface*

- The Java standard class library contains several interfaces that correspond to particular event categories

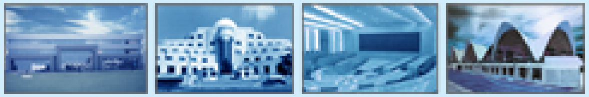
  – E.g. the MouseListener interface contains methods that correspond to mouse events

- After creating the listener, we *add* the listener to the component that might generate the event to set up a formal relationship between the generator and listener



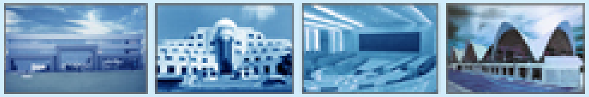
Some Listener Interfaces

- ActionListener – for receiving action events
  - E.g. mouse creates action on being clicked
- ItemListener – for receiving item events
  - E.g. list can be selected or deselected
- KeyListener- for receiving keyboard events (keystrokes).
- MouseListener
- MouseMotionListener
- WindowListener



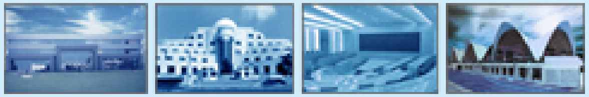
Mouse Events

- The following are *mouse events*:
  - *mouse pressed* - the mouse button is pressed down
  - *mouse released* - the mouse button is released
  - *mouse clicked* - the mouse button is pressed and released
  - *mouse entered* - the mouse pointer is moved over a particular component
  - *mouse exited* - the mouse pointer is moved off of a particular component
- Any given program can listen for some, none, or all of these



Event Based Programming

- Create a listener class implementing the appropriate listener interface.
- Register the listener with the required components.
  - The event listeners will be listening for the events.
- When any event occurs, the event source creates an appropriate event object & invokes the appropriate method of the registered listener.

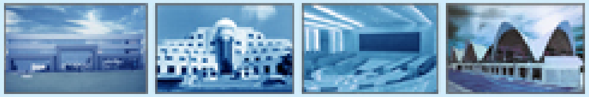


Event Hierarchy

AWTEvent

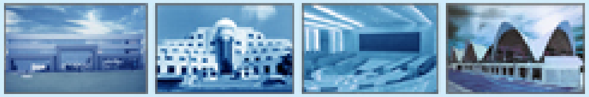
Abstract Class

- ActionEvent
- AdjustmentEvent
- ComponentEvent
  - ContainerEvent
  - FocusEvent
  - InputEvent
    - MouseEvent
    - KeyEvent
  - WindowEvent
- ItemEvent
- TextEvent



Event classes

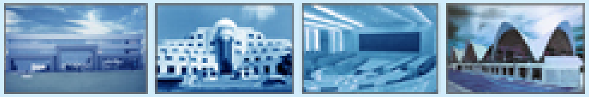
- ActionEvent
  - Generated when a button is pressed, a list is double-clicked, or a menu item is selected.
- AdjustmentEvent
  - Generated when a scroll bar is manipulated.
- ComponentEvent
  - Generated when a component is hidden, moved, resized or becomes visible
  - ContainerEvent
    - Generated when a component is added to or removed from a container.
  - FocusEvent
    - Generated when a component gains or loses keyboard focus.



Event classes

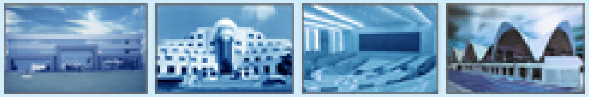
- InputEvent
  - Abstract super class for all component input event classes.
    - KeyEvent
      - Generated when input is received from the **keyboard**.
    - MouseEvent
      - Generated when the **mouse** is dragged, moved, clicked, pressed, or released;also generated when a mouse enters or exits a component.
- WindowEvent
  - Generated when a **window** is activated, closed, deactivated, deiconified, iconified, opened, or quit.

- ItemEvent
  - Generated when an item is selected or deselected
- TextEvent
  - Generated when the value of a **text area** or **text field** is changed.



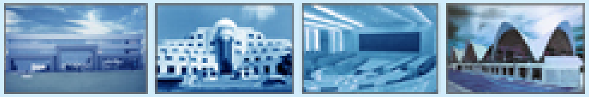
Event Sources

- Button
  - Generates action events when the button is pressed
- CheckBox
  - Generates item events when the check box is selected or deselected
- Choice
  - Generates item events when the choice is changed.
- List
  - Generates action events when an item is double-clicked
  - Generates item events when an item is selected or deselected



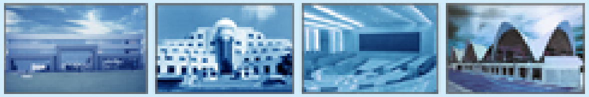
Event Sources

- Menu Item
  - Generates action events when a menu item is selected
  - Generates item events when a checkable menu item is selected or deselected
- Scrollbar
  - Generates adjustment events when the scroll bar is manipulated
- Text components
  - Generates text events when the user enters a character
- Window
  - Generates window events when a window is activated, closed, deactivated, deiconified, opened,or quit.



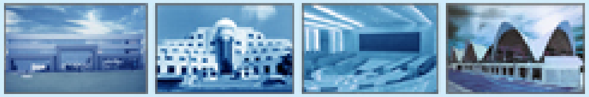
Listener Interfaces

- ActionListener
  - void actionPerformed(ActionEvent)
- AdjustmentListener
  - void adjustmentValueChanged(AdjustmentEvent)
- ComponentListener
  - void componentResized(ComponentEvent e)
  - void componentMoved(ComponentEvent e)
  - void componentShown(ComponentEvent e)
  - void componentHidden(ComponentEvent e)
- ContainerListener
  - void componentAdded(ContainerEvent e)
  - void componentRemoved(ContainerEvent e)



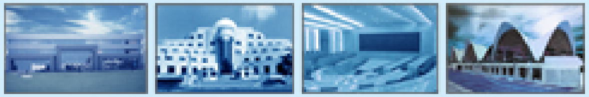
Listener Interfaces…

- FocusListener
  - void focusGained(FocusEvent e)
  - void focusLost(FocusEvent e)
- ItemListener
  - void itemStateChange(ItemEvent e)
- KeyListener
  - void keyPressed(KeyEvent e)
  - void keyReleased(KeyEvent e)
  - void keyTyped(KeyEvent e)
- MouseMotionListener
  - void mouseDragged(MouseEvent e)
  - void mouseMoved(MouseEvent e)
- TextListener
  - void textChanged(TextEvent e)



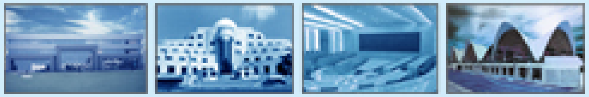
Listener Interfaces…

- **MouseListener**
  - void mouseClicked(MouseEvent e)
  - void mouseEntered(MouseEvent e)
  - void mouseExited(MouseEvent e)
  - void mousePressed(MouseEvent e)
  - void mouseReleased(MouseEvent e)
- **WindowListener**
  - void windowActivated(WindowEvent e)
  - void windowClosed(WindowEvent e)
  - void windowClosing(WindowEvent e)
  - void windowDeactivated(WindowEvent e)
  - void windowDeiconified(WindowEvent e)
  - void windowIconified(WindowEvent e)
  - void windowOpened(WindowEvent e)



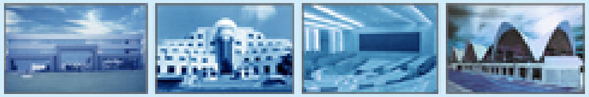
A Simple Event Handler

```
import java.awt.*;

public class TestButton{
    public static void main(String args[])
    {
            Frame f = new Frame(“Test”);
            Button b = new Button(“Press Me”);
            b.addActionListener(new ButtonHandler());
            f.add(b);
            f.pack();
            f.setVisible(true);
    }     }
```



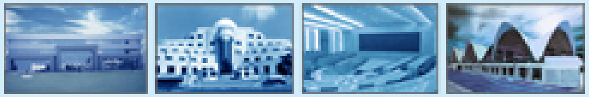
A Simple Event Handler

```
import java.awt.event.*;

public class ButtonHandler implements ActionListener {
    public void actionPerformed(ActionEvent e)
    {
            System.out.println(“Action occurred”);
            System.out.println(“Button’s label is” +
                        e.getActionCommand());
    }
}
```



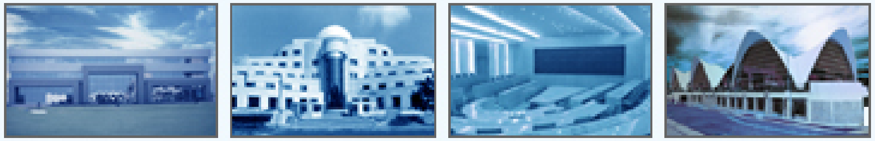
Recap

- Why Java
- Basic programming constructs
- Interfaces & packages
- Applets
- Exception Handling
- Event Handling



Infosys®

POWERED BY INTELLECT
DRIVEN BY VALUES



End of session

Education
and
Research