PROGRAMMING IN JAVA AND VB (502303)

Dr.RM. Vidhyavathi
Assistant Professor
Dept. of Bioinformatics
Alagappa University
Karaikudi-4.

Outline

- Introduction to Java
- Sample Program
- Data Types & Variables
- printf()
- Arithmetic & Logical Operations
- Conditionals
- Loops
- Arrays & Strings
- Applets
- Animation and Threads
- Interface and Packages
- Exception and Error Handling
- AWT-Windows Tool Kit
- Introduction to Visual Basic
- Connecting to Oracle Database using Visual Basic.
- Introduction to XML and HTML

UNIT-I-Introduction to JAVA programming

JAVA

- A programming language specifies the words and symbols that we can use to write a program
- A programming language employs a set of rules that dictate how the words and symbols can be put together to form valid program statements
- The Java programming language was created by Sun Microsystems, Inc.
- It was introduced in 1995 and it's popularity has grown quickly since

JAVA PROGRAM STRUCTURE

```
public class MyProgram
{
```

classheader

Class header

Comments can be placed almost anywhere

```
}
```

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



                    method header
 {
          method body
     }



}
```

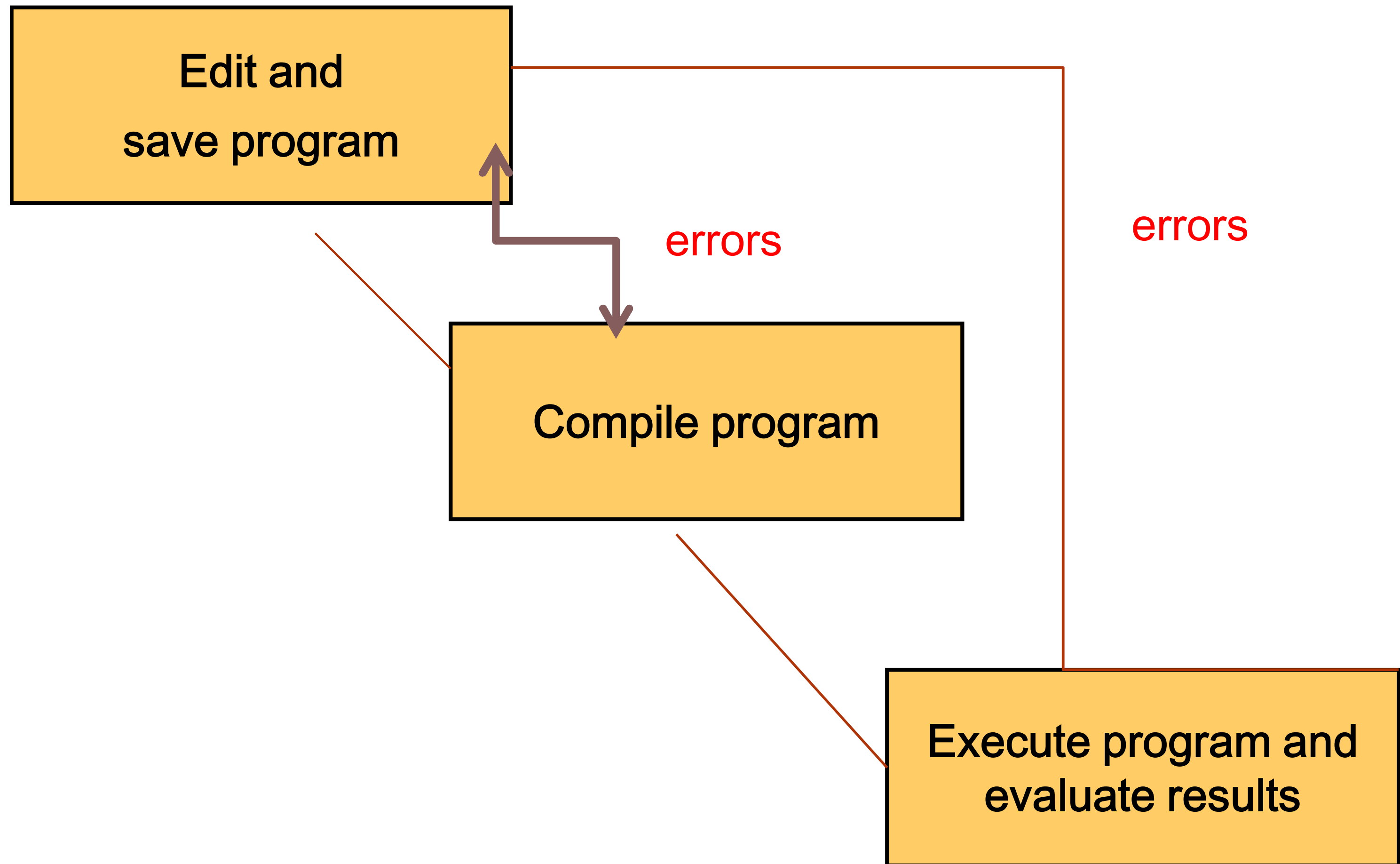
Sample Program

```
#include <stdio.h>
 int main()
{
        printf("Hello World\n");
        return(0);
}
```

JAVA OOP'S CONCEPT

- Java is a true OO language and therefore the underlying structure of all Java programs is classes.
- Anything we wish to represent in Java must be encapsulated in a class that defines the "state" and "behavior" of the basic program components known as objects.
- Classes create objects and objects use methods to communicate between them. They provide a convenient method for packaging a group of logically related data items and functions that work on them.
- A class essentially serves as a template for an object and behaves like a basic data type "int". It is therefore important to understand how the fields and methods are defined in a class and how they are used to build a Java program that incorporates the basic OO concepts such as encapsulation, inheritance, and polymorphism.

Program development



OBJECT

- An entity that has state and behavior is known as an object .Unique programming entity that has *methods*, has *attributes* and can react to *events*.

An object contains both data and methods that manipulate that data

An object is *active,* not passive; it *does* things

An object is *responsible* for its own data

But: it can *expose* that data to other objects

Object has three characteristics:

- **state:** represents data (value) of an object.
- **behavior:** represents the behavior (functionality) of an object such as deposit, withdraw etc.
- **identity:** Object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. But, it is used internally by the JVM to identify each object uniquely

CLASS IN JAVA

A class is a group of objects that has common properties. It is a template or blueprint from which objects are created. A class in java can contain:

- **data member**
- **method**
- **constructor**
- **block**
- **class and interface**

Syntax to declare a class:

```
class <class_name>{
    data member;
    method;
}
```

EXAMPLE OF OBJECT AND CLASS

```
class Student1
{
 int id;//data member (also instance variable)
 String name;//data member(also instance variable)
   public static void main(String args[])
{
  Student1 s1=new Student1();//creating an object of Student
  System.out.println(s1.id);
  System.out.println(s1.name);
 }
}
```

INHERITANCE IN JAVA

- It is a mechanism in which one object acquires all the properties and behaviors of parent object.
- Used in method overriding
- Used in code reusability.

```
class subclass-name extends Superclass-name
{
//mehods and fields
}
```

```
class Employee{

Float salary=4000;

}

class programmer extends Employee

{ int bonus=1000;

PublicStaticvoid main(string args[]){

Programmer p= new programmer();

System.out.println("programmer salary is."

+p.salary);

System.out.println("Bonus of programmer is."+p.bonus);

}

}
```

POLYMORHISM IN JAVA

- Polymorphism - perform a single action by different ways. It has two types;
  - compile time polymorphism
  - run time polymorphism

In runtime polymorphism or dynamic method dispatch is a process in which a call to an overridden method is resolved at runtime rather than compile time. An overridden method is called through the reference variable of a super class.

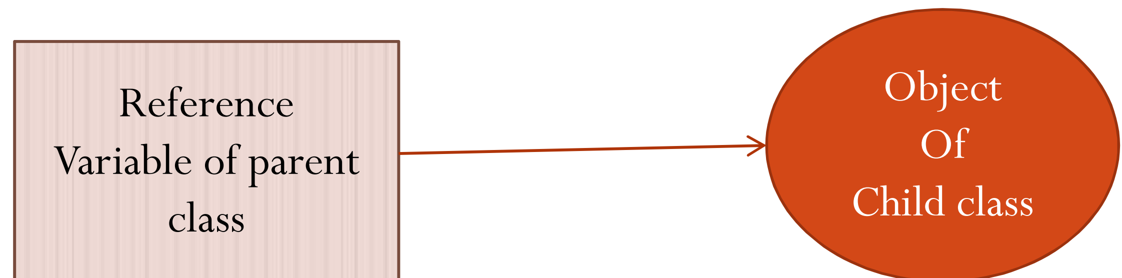
UPCASTING

- Reference variable of parent class refers to the object of child class, it is known as up casting.

Class A{}

Class B extends A{}

A a=new B();//upcasting



```
Class Bike{

Void run()system.out.println("running");}

}

Class splender extends bike{

Void run()system.out.println("running safely with 60km");}

Publicstaticvoid main(string args[]){

Bike b=new Splender();//upcasting

b.run(();

}

}
```

ENCAPSULATION IN JAVA

- It is process of wrapping code and data together into a single unit.

```
pakage com.javapoint;
publicclass student{
private string name;
public string getname(){
return name
}
publicvoid setName(string name){
This.name=name
}
}
```

```
pakage com.javapoint;

classTest

{

publicstaticvoid main(string args[])

{

Student s=new student();

s.setName("vijay");

System.out.println(s.getName());

}

}
```

ARRAYS, CONDITIONALS AND LOOPS

ARRAY

Arrays in Java are different than they are in other languages.

- Arrays in Java are actual objects that can be passed around and treated just like other objects.
- Arrays are a way to store a list of items.
- Each slot of the array holds an individual element
- Arrays can contain any type of element value, but can't store different types in a single array.
- The first step to creating an array is creating a variable that will hold the array

Declaring Array Variables

String difficultWords[];

Point hits[];

int temps[];

String[] difficult Words;

Point[] hits;

int[] temps;

CREATING ARRAY OBJECTS

- **The second step is to create an array object and assign it to that variable. There are two ways to do this:**
  - Using new
  - Directly initializing the contents of that array

```
String[] names = new String[10];
int[] temps = new int[99];
String[] chiles = { "jalapeno", "anaheim",
"serrano", "habanero", "thai"
};
```

ACCESSING ARRAY ELEMENTS

Once you have an array with initial values, you can test and change the values in each slot of that array. To get at a value stored within an array, use the array subscript expression:

- myArray[subscript]; // Object in location subscript
- int len = arr.length; // length of the array

CHANGING ARRAY ELEMENTS

- To assign an element value to a particular array slot, merely put an assignment statement after the array access expression:

```
myarray[1] = 15;
sentence[0] = "Abcdef";
sentence[10] = sentence[2];
```

MULTIDIMENSIONAL ARRAYS

Java does not support multidimensional arrays

```
int coords[][] = new int[12][12];

coords[0][0] = 1;

coords[0][1] = 2;
```

IF CONDITIONALS

The if conditional

```
 if (x < y)
 System.out.println("x is smaller than y");
```

An optional else keyword provides the statement to execute if the test is false:

```
if (x < y)
 System.out.println("x is smaller than y");
else System.out.println("y is bigger");
if (engineState == true )
 System.out.println("Engine is already on.");
```

```
else
{
 System.out.println("Now starting Engine.");
 if (gasLevel >= 1)
 engineState = true;
 else System.out.println("Low on gas!
Can't start engine.");
}
if (engineState)
 System.out.println("Engine is on.");
else System.out.println("Engine is off.");
```

SWITCH CONDITIONALS

```
if (oper == '+')
 addargs(arg1, arg2);
else if (oper == '-')
 subargs(arg1, arg2);
else if (oper == '*')
 multargs(arg1, arg2);
else if (oper == '/')
 divargs(arg1, arg2);
```

The switch or case statement; in Java it behaves as it does in C:

```
switch (test)
 {
 case valueOne:
 resultOne;
 break;
 case valueTwo:
 resultTwo;
 break;
 case valueThree:
 resultThree;
 break;
 ...
 default: defaultresult;
}
```

Here's a simple example of a switch statement similar to the nested if shown earlier:

```
switch (oper) {
 case '+':
 addargs(arg1, arg2);
 break;
 case '*':
 subargs(arg1, arg2);
 break;
 case '-':
 multargs(arg1, arg2);
 break;
divargs(arg1, arg2);
 break;
}
```

EXAMPLE

```
switch (x)
{
 case 2:
 case 4:
 case 6:
 case 8:
 System.out.println("x is an even number.");
 break;
 default: System.out.println("x is an odd number.");
}
```

FOR LOOPS

- The for loop, as in C, repeats a statement or block of statements some number of times until a condition is matched.

  The for loop in Java looks roughly like this:

```
for (initialization; test; increment)
{
 statements;
}
String strArray[] = new String[10];
int i; // loop index
for (i = 0; i < strArray.length; i++)
 strArray[i] = "";
for (i = 4001; notPrime(i); i += 2)
 ;
```

WHILE AND DO LOOPS

Finally, there are while and do loops.

while and do loops, like for loops, enable a block of Java code to be executed repeatedly until a specific condition is met.

while Loops

```
while (condition)
{
  bodyOfLoop;
 }
int count = 0;
  while ( count < array1.length && array1[count] !=0) {
array2[count] = (float) array1[count++];
}
```

LABELED LOOPS

- Both break and continue can have an optional label that tells Java where to break to. Without a label, break jumps outside the nearest loop, and continue restarts the enclosing loop.

- Using labeled breaks and continues enables you to break outside nested loops or to continue a loop outside the current loop.

- To use a labeled loop, add the label before the initial part of the loop:

```
out:
 for (int i = 0; i <10; i++) {
 while (x < 50) {
 if (i * x == 400)
 break out;
 ...
 }
 ...
 }
foo:
 for (int i = 1; i <= 5; i++)
 for (int j = 1; j <= 3; j++)
{
System.out.println("i is " + i + ", j is " + j);
 if ((i + j) > 4)
 break foo;
 }
System.out.println("end of loops");
```

OUTPUT:

```
i is 1, j is 1
i is 1, j is 2
i is 1, j is 3
i is 2, j is 1
i is 2, j is 2
i is 2, j is 3
end of loops
```

UNIT-II

Classes and Objects

- A Java program consists of one or more classes
- A class is an abstract description of objects
- Here is an example class:
  - class Dog { *...description of a dog goes here...* }
- Here are some objects of that class:

Classes contain data definitions

- Classes describe the data held by each of its objects
- Example:
  - class Dog {
    String name;
    int age;
    ...rest of the class...
    }

Data usually goes first in a class

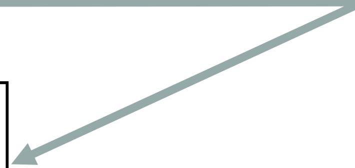
- A class may describe any number of objects
  - Examples: "Fido", 3; "Rover", 5; "Spot", 3;
- A class may describe a single object, or even no objects at all

Classes contain methods

- A class may contain methods that describe the behavior of objects
- Example:
  - class Dog {

```
class Dog {
    ...
    void bark() {
        System.out.println("Woof!");
    }
}
```

Methods usually go after the data



- When we ask a *particular* Dog to bark, it says “Woof!”
- Only Dog *objects* can bark; the *class* Dog cannot bark

Methods contain statements

- A statement causes the object to do something
  - (A better word would be "command"—but it isn't)
- Example:
  - System.out.println("Woof!");
  - This causes the particular Dog to "print" (actually, display on the screen) the characters Woof!

Methods may contain temporary data

- Data described in a class exists in all objects of that class
  - Example: Every Dog has its own name and age
- A method may contain local *temporary* data that exists only until the method finishes
- Example:
  - ```
    void wakeTheNeighbors( ) {
        int i = 50;     // i is a temporary variable
        while (i > 0) {
            bark( );
            i = i – 1;
        }
    }
    ```
    ```

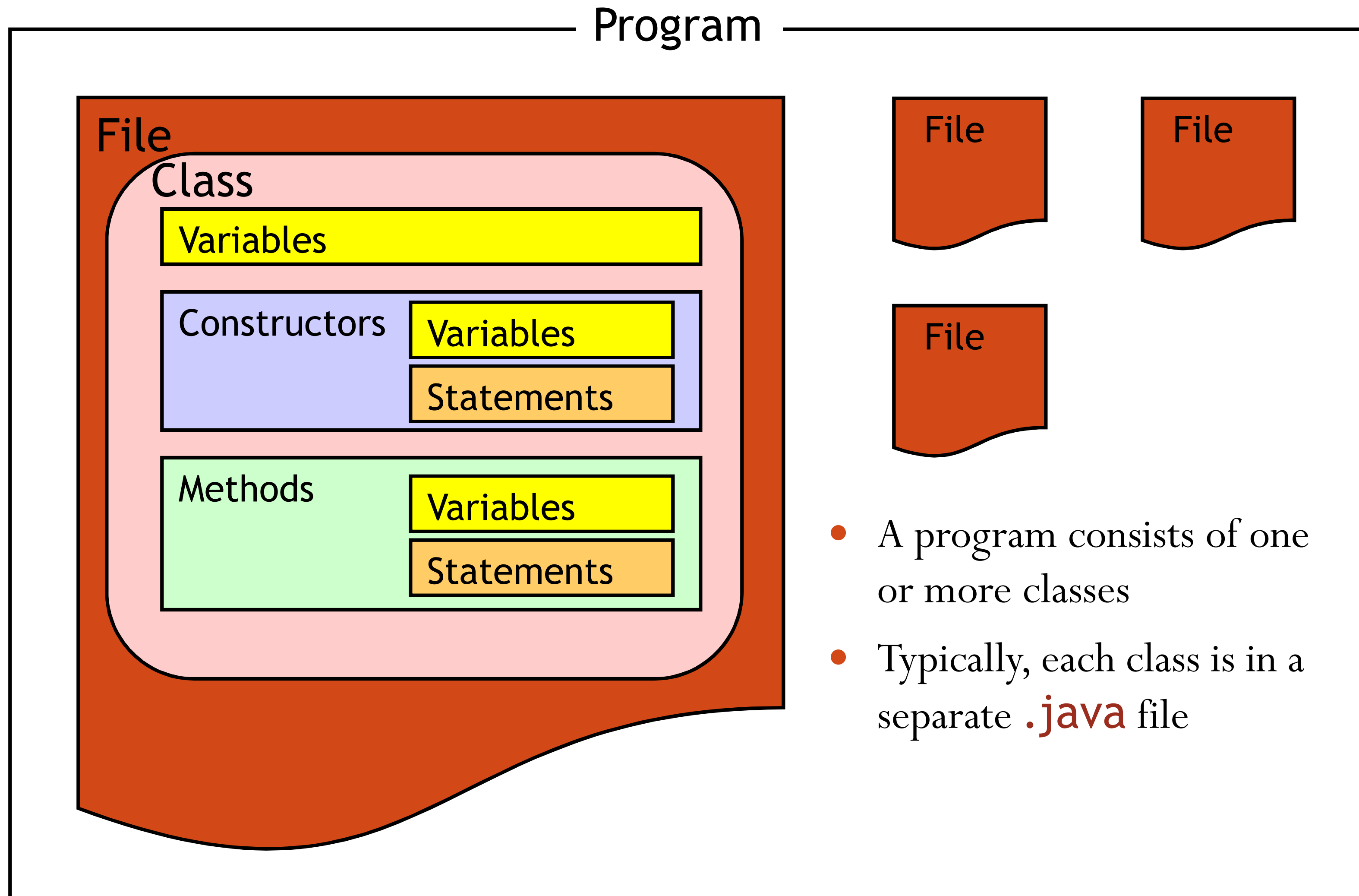
Classes always contain constructors

- A constructor is a piece of code that "constructs," or creates, a new object of that class
- If you don't write a constructor, Java defines one for you (behind the scenes)
- You can write your own constructors
- Example:
  - class Dog {

```
class Dog {
    String name;
    int age;
    Dog(String n, int age) {
        name = n;
        this.age = age;
    }
}
```

(This part is the constructor)

Diagram of program structure

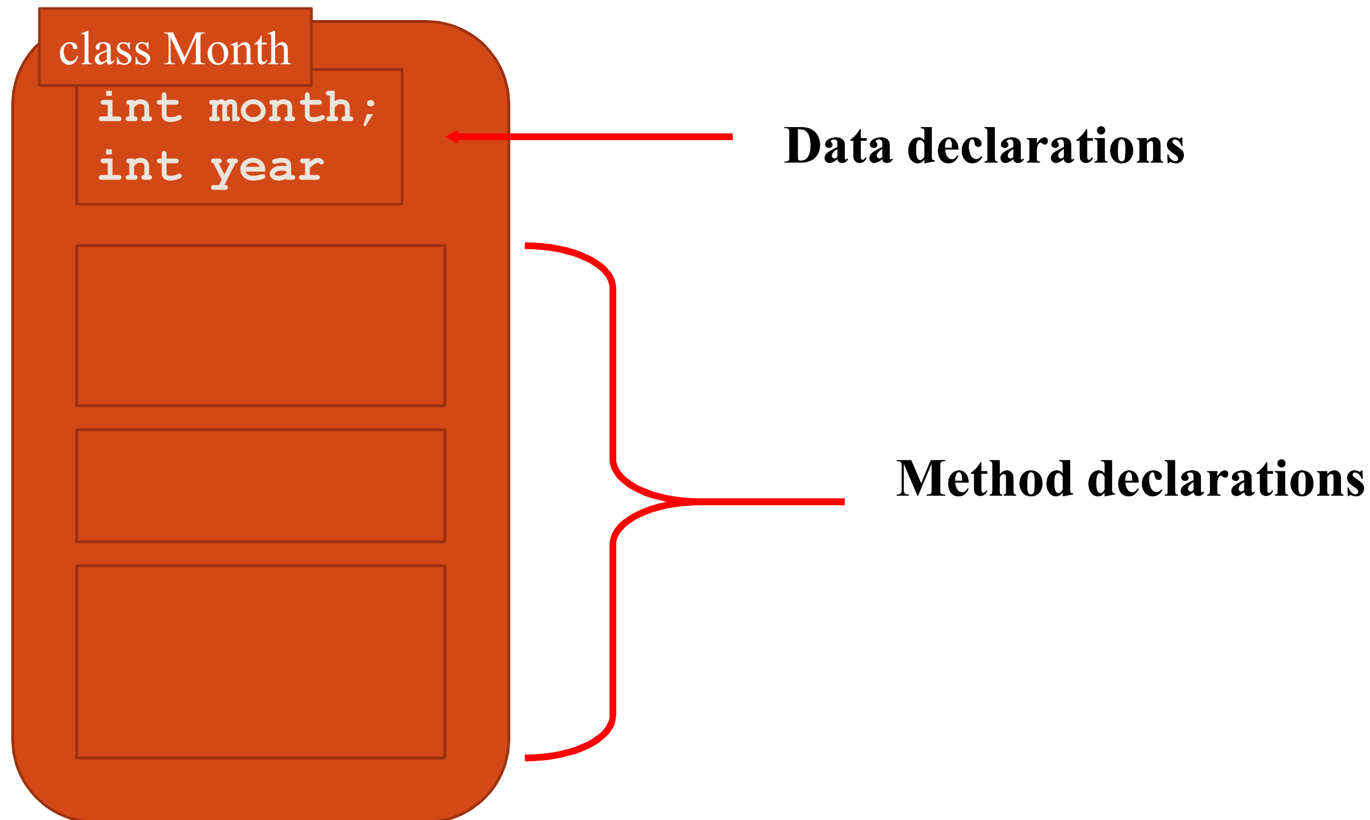


- A program consists of one or more classes
- Typically, each class is in a separate `.java` file

Methods

Defining Classes

- A class contains data declarations (static and instance variables) and method declarations (behaviors)



Methods

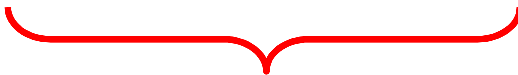
- A program that provides some functionality can be long and contains many statements

- A method groups a sequence of statements and should provide a well-defined, easy-to-understand functionality
  - a method takes input, performs actions, and produces output
- In Java, each method is defined within specific class

Method Declaration: Header

- A method declaration begins with a *method header*

```
class MyClass
{…
   static  int    min ( int num1, int num2 )
```



properties

return type

method name

parameter list

The parameter list specifies the type and name of each parameter

The name of a parameter in the method declaration is called a *formal argument*

Method Declaration: Body

The header is followed by the *method body:*

```
class MyClass
{
     …
     static int min(int num1, int num2)
     {
        int minValue = num1 < num2 ? num1 : num2;
        return minValue;
     }


     …
 }
```

The return Statement

- The *return type* of a method indicates the type of value that the method sends back to the calling location
  - A method that does not return a value has a void return type
- The *return statement* specifies the value that will be returned
  - Its expression must conform to the return type

Calling a Method

- Each time a method is called, the values of the *actual arguments* in the invocation are assigned to the *formal arguments*

```
int  num = min (2, 3);
```

```
static int min (int num1, int num2)
{
    int minValue = (num1 < num2 ? num1 : num2);
    return minValue;
}
```

Method Overloading

- A class may define multiple methods with the same name---this is called method overloading
  - usually perform the same task on different data types
- Example: The PrintStream class defines multiple println methods, i.e., println is overloaded:

```
println (String s)
println (int i)
println (double d)
   …
```

- The following lines use the System.out.print method for different data types:

```
System.out.println ("The total is:");
double total = 0;
System.out.println (total);
```

Method Overloading: Signature

- The compiler must be able to determine which version of the method is being invoked
- This is by analyzing the parameters, which form the *signature* of a method
  - the signature includes the type and order of the parameters
    - if multiple methods match a method call, the compiler picks the best match
    - if none matches exactly but some implicit conversion can be done to match a method, then the method is invoke with implicit conversion.
  - the return type of the method is not part of the signature

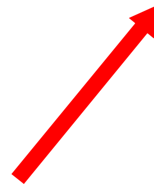
Method Overloading

Version 1

```
double tryMe (int x)
{
   return x + .375;
}
```

Version 2

```
double tryMe (int x, double y)
{
   return x * y;
}
```



Invocation

```
result = tryMe (25, 4.32)
```

More Examples

```
double tryMe ( int x )
{
   return x + 5;
}
```

```
double tryMe ( double x )
{
   return x * .375;
}
```

```
double tryMe (double x, int y)
{
   return x + y;
}
```

Which tryMe will be called?

```
tryMe( 1 );

tryMe( 1.0 );

tryMe( 1.0, 2);

tryMe( 1, 2);

tryMe( 1.0, 2.0);
```

Variable Scoping

Variables

- At a given point, the variables that a statement can access are determined by the scoping rule
  - the scope of a variable is the section of a program in which the variable can be accessed (also called visible or in scope)
- There are two types of scopes in Java
  - class scope
    - a variable defined in a class but not in any method
  - block scope
    - a variable defined in a block {} of a method; it is also called a local variable

Java Scoping Rule

- A variable with a class scope
  - *class/static variable*: a variable defined in class scope and has the static property
    - it is associated with the class
    - and thus can be accessed (in scope) in all methods in the class
  - *instance variable*: a variable defined in class scope but not static
    - it is associated with an instance of an object of the class,
    - and thus can be accessed (in scope) only in instance methods, i.e., those non-static methods
- A variable with a block scope
  - can be accessed in the enclosing block; also called local variable
  - a local variable can shadow a variable in a class scope with the same name
  - a variable may exist but is not accessible in a method,
    - e.g., method A calls method B, then the variables declared in method A exist but are not accessible in B.

Scoping Rules (cont.): Variables in a method

- There are can be three types of variables accessible in a *method* :
  - class and instance variables
    - static and instance variables of the class
  - local variables
    - those declared in the method
  - formal arguments

Example1:

```
public class Box
{
  private int length, width;
  …

  public int widen (int extra_width)
  {
         private int temp1;
         size += extra_width;
         …
  }
  public int lenghten (int extra_lenth)
  {
         private int temp2;
         size += extra_length;
         …
  }

  …

}
```

- instance variables
- formal arguments
- local variables

Scope of Variables

```
public class Box
{
  private int length, width;
  …

  public int widen (int extra_width)
  {
          private int temp1;
          size += extra_width;
          …
  }
  public int lenghten (int extra_lenth)
  {
          private int temp2;
          size += extra_length;
          …
  }

  …

}
```

- Instance variables are accessible in all methods of the class
- formal arguments are valid within their methods
- Local variables are valid from the point of declaration to the end of the enclosing block

Two Types of Parameter Passing

If a modification of the *formal argument* has no effect on the *actual argument*,

- it is call by value

If a modification of the *formal argument* can change the value of the *actual argument*,

- it is call by reference

Call-By-Value and Call-By-Reference in Java

- Depend on the type of the formal argument
- If a formal argument is a primitive data type, a modification on the formal argument has no effect on the actual argument
  - this is call by value, e.g. **num1 = min(2, 3);**
    num2 = min(x, y);
- This is because primitive data types variables *contain their values*, and procedure call trigger an assignment:

  <formal argument> = <actual argument>

```
int x = 2; int y = 3;
int num = min (x, y);
...
static int num( int num1, int num2)
{ ... }
```

```
int x = 2;
int y = 3;
int num1 = x;
int num2 = y;
{ … }
```

Call-By-Value and Call-By-Reference in Java

- If a formal argument is not a primitive data type, an operation on the formal argument can change the actual argument
  - this is call by reference
- This is because variables of object type *contain* ***pointers*** to the data that represents the object
- Since procedure call triggers an assignment

  <formal argument> = <actual argument>

  it is the pointer that is copied, not the object itself

```
MyClass x = new MyClass();
MyClass y = new MyClass();
MyClass.swap( x, y);
...
static void swap( MyClass x1, MyClass x2)
{ ... }
```

```
x = new MC();
y = new MC();
x1 = x;
x2 = y;
{ … }
```

Classes define Objects

- In object-oriented design, we group methods together according to objects on which they operate
- An object has:
  - *state* - descriptive characteristics
  - *behaviors* - what it can do (or be done to it), may depend on the state, and can change the state
- For example, a calendar program needs Month objects:
  - the state of a Month object is a (month,year) pair of numbers
  - these are stored as *instance variables* of the Month class
  - the Month class can also have *class variables,* e.g. the day of the week that Jan 1, 2000 falls on…
  - some behaviors of a month object are:
    - get name, get first weekday, get number of days,
    - print
    - set month index, set year

JAVA APPLET BASICS

INTRODUCTION :

- An applet is a Panel that allows interaction with a Java program
- A applet is typically embedded in a Web page and can be run from a browser
- You need special HTML in the Web page to tell the browser about the applet

We write an applet by extending the class Applet

Applet is just a class like any other; you can even use it in applications if you want

When you write an applet, you are only writing *part* of a program

The browser supplies the main method

The simplest reasonable applet

```
import java.awt.*;
import java.applet.Applet;

public class HelloWorld extends Applet {
    public void paint( Graphics g ) {
        g.drawString( "Hello World!", 30, 30 );
    }
}
```

Applet methods

public void init ()

public void start ()

public void stop ()

public void destroy ()

public void paint (Graphics)

Also:

public void repaint()

public void update (Graphics)

public void showStatus(String)

public String getParameter(String)

public void init ( )

- This is the first method to execute
- It is an ideal place to initialize variables
- It is the best place to define the GUI Components
- (buttons, text fields, scrollbars, etc.), lay them out, and add listeners to them
- Almost every applet you ever write will have an init( ) method

public void start ( )

- Not always needed
- Called after init( )
- Called each time the page is loaded and restarted
- Used mostly in conjunction with stop( )
- start() and stop( ) are used when the Applet is doing time-consuming calculations that you don’t want to continue when the page is not in front

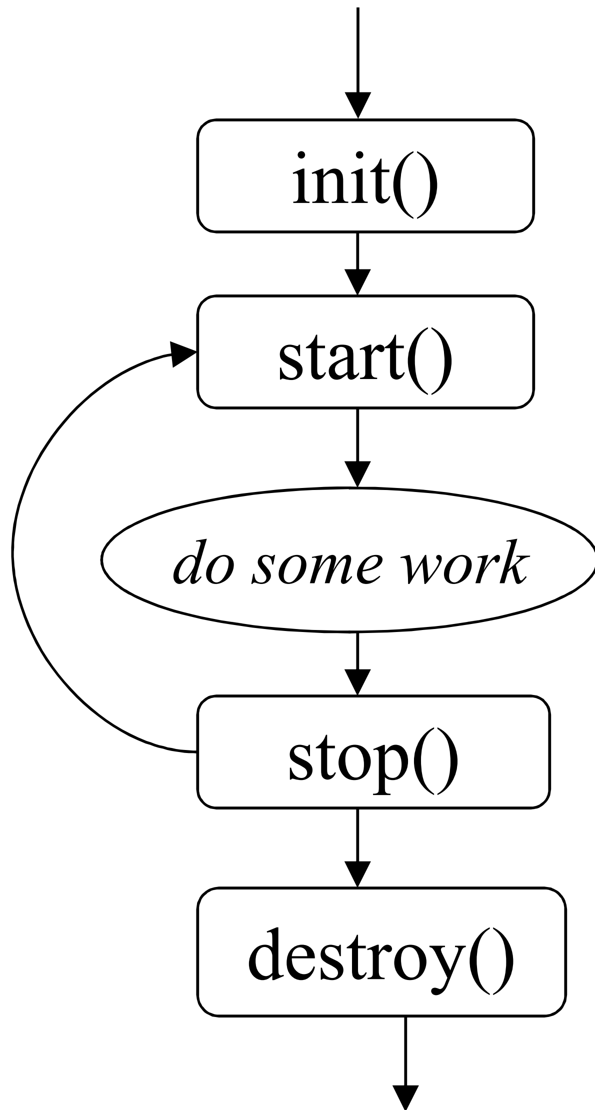
public void stop( )

- Not always needed
- Called when the browser leaves the page
- Called just before destroy( )
- Use stop( ) if the applet is doing heavy computation that you don't want to continue when the browser is on some other page
- Used mostly in conjunction with start()

public void destroy( )

- Seldom needed
- Called after stop( )
- Use to explicitly release system resources (like threads)
- System resources are usually released automatically

Methods are called in this order



- init and destroy are only called once each
- start and stop are called whenever the browser enters and leaves the page
- *do some work* is code called by your *listeners*
- paint is called when the applet needs to be repainted

public void paint(Graphics g)

- Needed if you do any drawing or painting other than just using standard GUI Components
- Any painting you want to do should be done here, or in a method you call from here
- Painting that you do in other methods may *or may not* happen
- *Never call* paint(*Graphics*)*, c*all repaint( )

repaint( )

- Call repaint( ) when you have changed something and want your changes to show up on the screen
- repaint( ) is a *request*--it might not happen
- When you call repaint( ), Java schedules a call to update(Graphics g)

```
import java.awt.*;
import java.applet.*;


public class WelcomeApplet extends Applet {

public void init() {
 }

public void paint(Graphics g) {
    g.drawString("Welcome to Java Programming!",                    25, 25 );
 }
}
```

`extends` allows us to inherit the capabilities of class `Applet`.

Method `paint` is guaranteed to be called in all applets. Its first line must be defined as above.

Simple Java Applet: Drawing a String

```
import java.awt.*;      // import package with class Graphics
import javax.applet.*;  // import class Applet
```

- Import predefined classes grouped into packages
  - When you create applets, import Applet class (in the package java. applet)
  - import the Graphics class (package java.awt) to draw graphics
    - Can draw lines, rectangles ovals, strings of characters
  - import specifies directory structure

Simple Java Applet: Drawing a String

```
g.drawString( "Welcome to Java Programming!", 25, 25 );
```

- Body of paint
  - Method drawString (of class Graphics)
  - Called using Graphics object g and dot (.)
  - Method name, then parenthesis with arguments
    - First argument: String to draw
    - Second: x coordinate (in pixels) location
    - Third: y coordinate (in pixels) location
- Java coordinate system
  - Measured in pixels (picture elements)
  - Upper left is (0,0)

Simple Java Applet: Drawing a String

- Running the applet
  - Compile
    - a regular file in JCreator
      - or
    - javac WelcomeApplet.java (from the command line)
    - If no errors, bytecodes stored in WelcomeApplet.class
  - Create an HTML file
    - Loads the applet into appletviewer or a browser
    - Ends in .htm or .html
  - To execute an applet
    - Create an HTML file indicating which applet the browser (or appletviewer) should load and execute

Simple Java Applet: Drawing a String

```
<html>
<applet code = "WelcomeApplet.class" width = "300" height = "45">
</applet>
</html>
```

- Simple HTML file (WelcomeApplet.html)
  - Usually in same directory as **.class** file
  - Remember, .class file created after compilation
- HTML codes (tags)
  - Usually come in pairs
  - Begin with < and end with >
- Lines 1 and 4 - begin and end the HTML tags
- Line 2 - begins <applet> tag
  - Specifies code to use for applet
  - Specifies width and height of display area in pixels
- Line 3 - ends <applet> tag

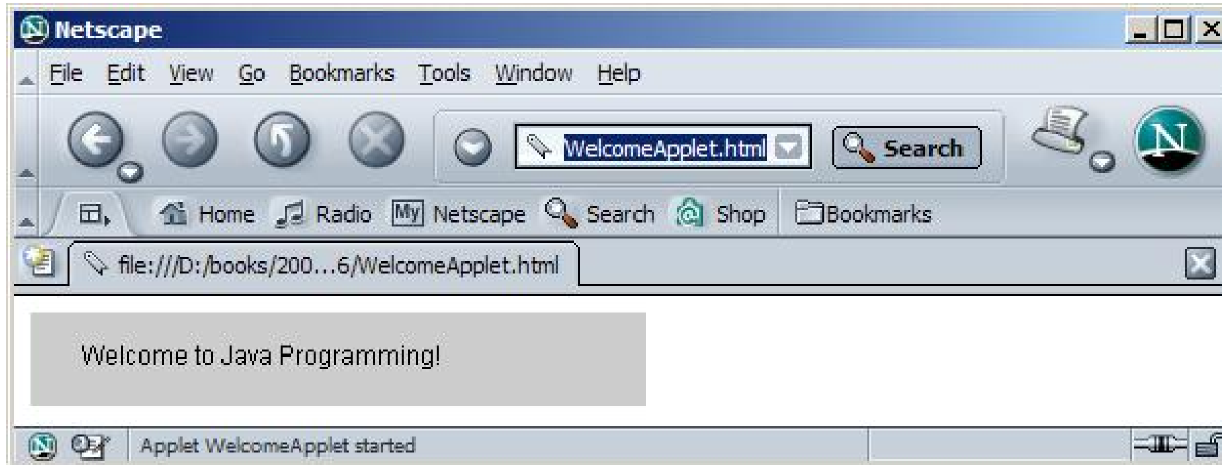
Simple Java Applet: Drawing a String

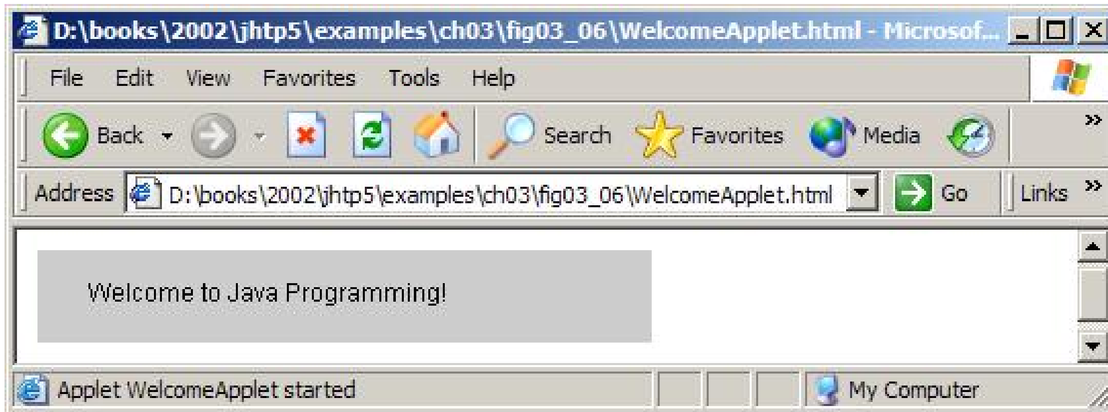
```html
<html>
<applet code = "WelcomeApplet.class" width = "300" height = "45">
</applet>
</html>
```

- appletviewer only understands <applet> tags
  - Ignores everything else
  - Minimal browser
- Executing the applet
  - appletviewer WelcomeApplet.html
  - Perform in directory containing .class file

Simple Java Applet: Drawing a String

Running the applet in a Web browser





ANIMATION AND THREADS

Moving pictures

- Animation—making objects that appear to move on the screen—is done by displaying a series of still pictures, one after the other, in rapid succession.

- Eg: cartoon on television, bouncing ball etc…

- Animation in java is accomplished by through various interrelated parts of the java abstract windowing toolkit(awt).

CREATING ANIMATION IN JAVA

- Involves two basic step:

- 1.Constructing a frame of animation.
- 2.Asking java to paint that frame.

PAINTING AND REPAINTING

Java applet can contain many different components that all need to be painted (paint())and in fact, applets can be embedded inside a larger Java application that also paints to the screen in similar ways, when you call repaint()

Starting and Stopping an Applet's Execution

- The start() method triggers the execution of the applet. You can either do all the applet's work inside that method, or you can call other object's methods in order to do so. Usually, start() is used to create and begin execution of a thread so the applet can run in its own time.

- stop(), on the other hand, suspends an applet's execution so when you move off the page on which the applet is displaying, it doesn't keep running and using up system resources. Most of the time when you create a start() method, you should also create a corresponding stop().

Adding sound

- It is easy to add sound to an animation.
- can use getaudioclip() method to get an audio clip object.

Threads

- A Thread is a single flow of control
  - When you step through a program, you are following a Thread
- Most simple non-GUI programs use a single "main Thread"
- A Thread is an Object you can create and control

Sleeping

- Every program uses at least one Thread
- Thread.sleep(int ***milliseconds***);
  - A millisecond is 1/1000 of a second
- try { Thread.sleep(1000); }

  catch (InterruptedException e) { }
- sleep only works for the current Thread

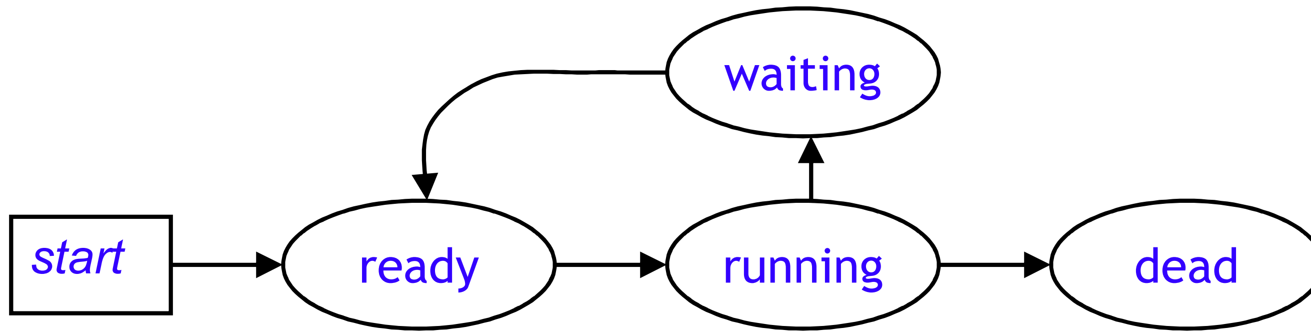
States of a Thread

- A Thread can be in one of four states:
  - **Ready:** all set to run
  - **Running:** actually doing something
  - **Waiting,** or **blocked:** needs something
  - **Dead:** will never do anything again

Two ways of creating Threads

- You can extend the Thread class:
  - class Animation extends Thread {…}
  - Limiting, since you can only extend one class
- Or you can implement the Runnable interface:
  - class Animation implements Runnable {…}
  - requires public void run( )

State transitions



Extending Thread

- class Animation extends Thread {

  public void run( ) { ***code for this thread*** }

  Anything else you want in this class

  }
- Animation anim = new Animation( );
  - A newly created Thread is in the **Ready** state
- To start the anim Thread running, call anim.start( );
- start( ) is a *request* to the scheduler to run the Thread --it may not happen right away
- The Thread should eventually enter the **Running** state

Implementing Runnable

- class Animation implements Runnable {…}
- The Runnable interface requires run( )
  - This is the "main" method of your new Thread
- Animation anim = new Animation( );
- Thread myThread = new Thread(anim);
- To start the Thread running, call myThread.start( );
  - You do not write the start() method—it's provided by Java
- As always, start( ) is a *request* to the scheduler to run the Thread--it may not happen right away

Starting a Thread

- Every Thread has a start( ) method
- *Do not* write or override start( )
- You *call* start( ) to request a Thread to run
- The scheduler then (eventually) calls run( )
- You must supply public void run( )
  - This is where you put the code that the Thread is going to run

Implementing Runnable: summary

```
class Animation extends Applet
                    implements Runnable {
    public void run( ) {
        while (okToRun) { ... }
    }
}

Animation anim = new Animation( );

Thread myThread = new Thread(anim);

myThread.start( );
```

How to animate

- Create your buttons and attach listeners in your first (original) Thread
- Create a second Thread to run the animation
- Start the animation
- The original Thread is free to listen to the buttons

DIGITAL CLOCK

- import java.awt.Graphics;
- import java.awt.Font;
- import java.util.Date;
- public class DigitalClock extends java.applet.Applet
- implements Runnable {
- Font theFont = new Font("TimesRoman",Font.BOLD,24);
  Date theDate;
- Thread runner;
- public void start() {
- if (runner == null) {
- runner = new Thread(this);

```
runner.start();
}
}
public void stop() {
if (runner != null) {
runner.stop();
runner = null;
}
}
public void run() {
while (true) {
theDate = new Date();
repaint();
try { Thread.sleep(1000); }
 catch (InterruptedException e) { }
}
}
```

- public void run() {
- while (true) {
- theDate = new Date();
- repaint();
- try { Thread.sleep(1000); }
- catch (InterruptedException e) { }
- }
- }
- public void paint(Graphics g) {
- g.setFont(theFont); g.drawString(theDate.toString(),10,50);
- }
- }

Image & Sound

Image

- In the world of computers, there are many types of images, each of which is associated with a specific file format.
- These image types are usually identified by their file extensions, which include PCX, BMP, GIF, JPEG (or JPG), TIFF (or TIF), TGA, and more.
- Each of these file types was created by third-party software companies for use with their products, but many became popular enough to grow into standards.

Object of Image

The java.applet.

Applet class provides getImage() method that returns the object of Image.

Syntax:

1. public Image getImage(URL , String image)

{}

Loading and Displaying an Image

The first step in displaying an image in your applet is to load the image from disk.

create an object of Java's Image class.

create an URL object that holds the location of the graphics file.

A better way to create the image's URL object is to call either the

getDocumentBase() or getCodeBase() method.

getDocumentBase() Method

getDocumentBase() method returns the URL of the directory from which the HTML document was loaded.

the getCodeBase() Method

The getCodeBase() method works similarly to getDocumentBase(), except that it returns the URL of the directory from which the applet was loaded.

If you're storing your images in the same directory (or a subdirectory of that directory) as your CLASS files, you'd want to call getCodeBase() to obtain an URL for an image.

CLASS files in a directory called CLASSES and the image you want (still called IMAGE.gif) is stored in a subdirectory of CLASSES called IMAGES.

Example of displaying image in applet

```
1. import java.awt.*;
2. import java.applet.*;
3. public class DisplayImage extends Applet {
4. Image picture;
5. public void init() {
6. picture = getImage(getDocumentBase(),"sonoo.jpg");
7.   }
8.  public void paint(Graphics g) {
9.   g.drawImage(picture, 30,30);
10.  }
11.  }
```

Sound

The Java Sound API, the word Sound takes on a somewhat different meaning.

However, it is probably fair to say that the ultimate purpose of the Sound API is to assist the user in writing programs that will cause sound pressure waves impinge upon the ears of targeted individuals at specific times.

Sound Packages

Two significantly different types of audio (or sound) data are supported by the API:

Sampled audio data

Musical Instrument Digital Interface (MIDI) data

Sampled audio data

Sampled audio data can be thought of as a series of digital values that represent the amplitude or intensity of sound pressure waves. This will be the primary topic of the first several lessons in this miniseries.

This type of audio data is supported by the following two Java packages:

javax.sound.sampled

javax.sound.sampled.spi

According to Sun, the first of these two packages "specifies interfaces for capture, mixing, and playback of digital (sampled) audio." I will have more to say about the second (spi) package shortly.

MIDI data

MIDI data can be thought of as sound (usually musical sound or special sound effects) created from a recipe.

This type of audio data is covered by the following two Java packages:

javax.sound.midi
javax.sound.midi.spi

According to Sun, the first of these two packages "provides interfaces for MIDI synthesis, sequencing, andevent transport."

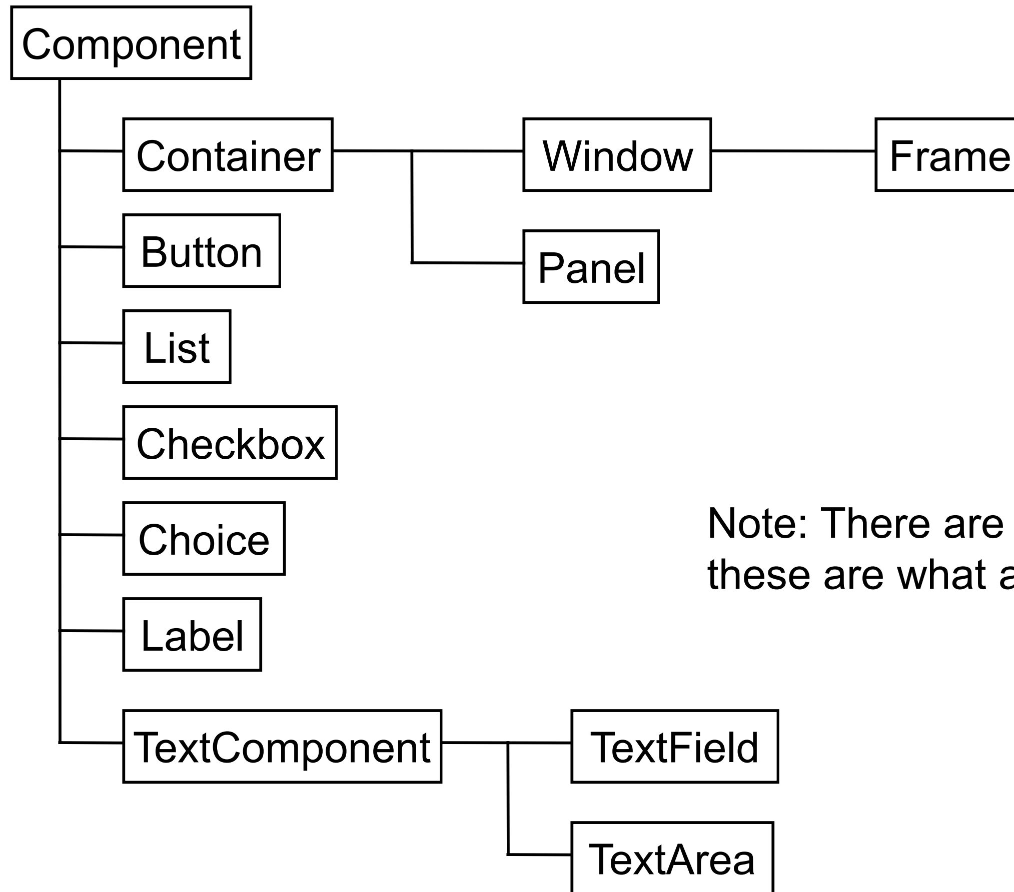
UNIT-III-ABSTRACT WINDOW TOOLKIT

AWT (Abstract Windowing Toolkit)

- The AWT is roughly broken into three categories
  - Components
  - Layout Managers
  - Graphics

- Many AWT components have been replaced by Swing components
- It is generally not considered a good idea to mix Swing components and AWT components. Choose to use one or the other.

AWT – Class Hierarchy



Note: There are more classes, however, these are what are covered in this chapter

Component

- Component is the superclass of most of the displayable classes defined within the AWT. Note: it is abstract.
- MenuComponent is another class which is similar to Component except it is the superclass for all GUI items which can be displayed within a drop-down menu.
- The Component class defines data and methods which are relevant to all Components

  setBounds

  setSize

  setLocation

  setFont

  setEnabled

  setVisible

  setForeground -- colour

  setBackground -- colour

Container

- Container is a subclass of Component. (ie. All containers are themselves, Components)
- Containers contain components
- For a component to be placed on the screen, it must be placed within a Container
- The Container class defined all the data and methods necessary for managing groups of Components

  add

  getComponent

  getMaximumSize

  getMinimumSize

  getPreferredSize

  remove

  removeAll

Windows and Frames

- The Window class defines a top-level Window with no Borders or Menu bar.
  - Usually used for application splash screens

- Frame defines a top-level Window with Borders and a Menu Bar
  - Frames are more commonly used than Windows

- Once defined, a Frame is a Container which can contain Components

```
Frame aFrame = new Frame("Hello World");
aFrame.setSize(100,100);
aFrame.setLocation(10,10);
aFrame.setVisible(true);
```

Panels

- When writing a GUI application, the GUI portion can become quite complex.
- To manage the complexity, GUIs are broken down into groups of components. Each group generally provides a unit of functionality.
- A Panel is a rectangular Container whose sole purpose is to hold and manage components within a GUI.

```
Panel aPanel = new Panel();
aPanel.add(new Button("Ok"));
aPanel.add(new Button("Cancel"));

Frame aFrame = new Frame("Button Test");
aFrame.setSize(100,100);
aFrame.setLocation(10,10);
aFrame.add(aPanel);
```

Buttons

- This class represents a push-button which displays some specified text.
- When a button is pressed, it notifies its Listeners. (More about Listeners in the next chapter).
- To be a Listener for a button, an object must implement the ActionListener Interface.

```
Panel aPanel = new Panel();
Button okButton = new Button("Ok");
Button cancelButton = new Button("Cancel");
aPanel.add(okButton));
aPanel.add(cancelButton));
okButton.addActionListener(controller2);
cancelButton.addActionListener(controller1);
```

Labels

- This class is a Component which displays a single line of text.
- Labels are read-only. That is, the user cannot click on a label to edit the text it displays.
- Text can be aligned within the label

```
Label aLabel = new Label("Enter password:");
aLabel.setAlignment(Label.RIGHT);

aPanel.add(aLabel);
```

List

- This class is a Component which displays a list of Strings.
- The list is scrollable, if necessary.
- Sometimes called Listbox in other languages.
- Lists can be set up to allow single or multiple selections.
- The list will return an array indicating which Strings are selected

```
List aList = new List();
aList.add("Calgary");
aList.add("Edmonton");
aList.add("Regina");
aList.add("Vancouver");
aList.setMultipleMode(true);
```

Checkbox

- This class represents a GUI checkbox with a textual label.
- The Checkbox maintains a boolean state indicating whether it is checked or not.
- If a Checkbox is added to a CheckBoxGroup, it will behave like a radio button.

```
Checkbox creamCheckbox = new CheckBox("Cream");
Checkbox sugarCheckbox = new CheckBox("Sugar");
[…]
if (creamCheckbox.getState())
{
coffee.addCream();
}
```

Choice

- This class represents a dropdown list of Strings.
- Similar to a list in terms of functionality, but displayed differently.
- Only one item from the list can be selected at one time and the currently selected element is displayed.

```
Choice aChoice = new Choice();
aChoice.add("Calgary");
aChoice.add("Edmonton");
aChoice.add("Alert Bay");
[…]

String selectedDestination= aChoice.getSelectedItem();
```

TextField

- This class displays a single line of optionally editable text.
- This class inherits several methods from TextComponent.
- This is one of the most commonly used Components in the AWT

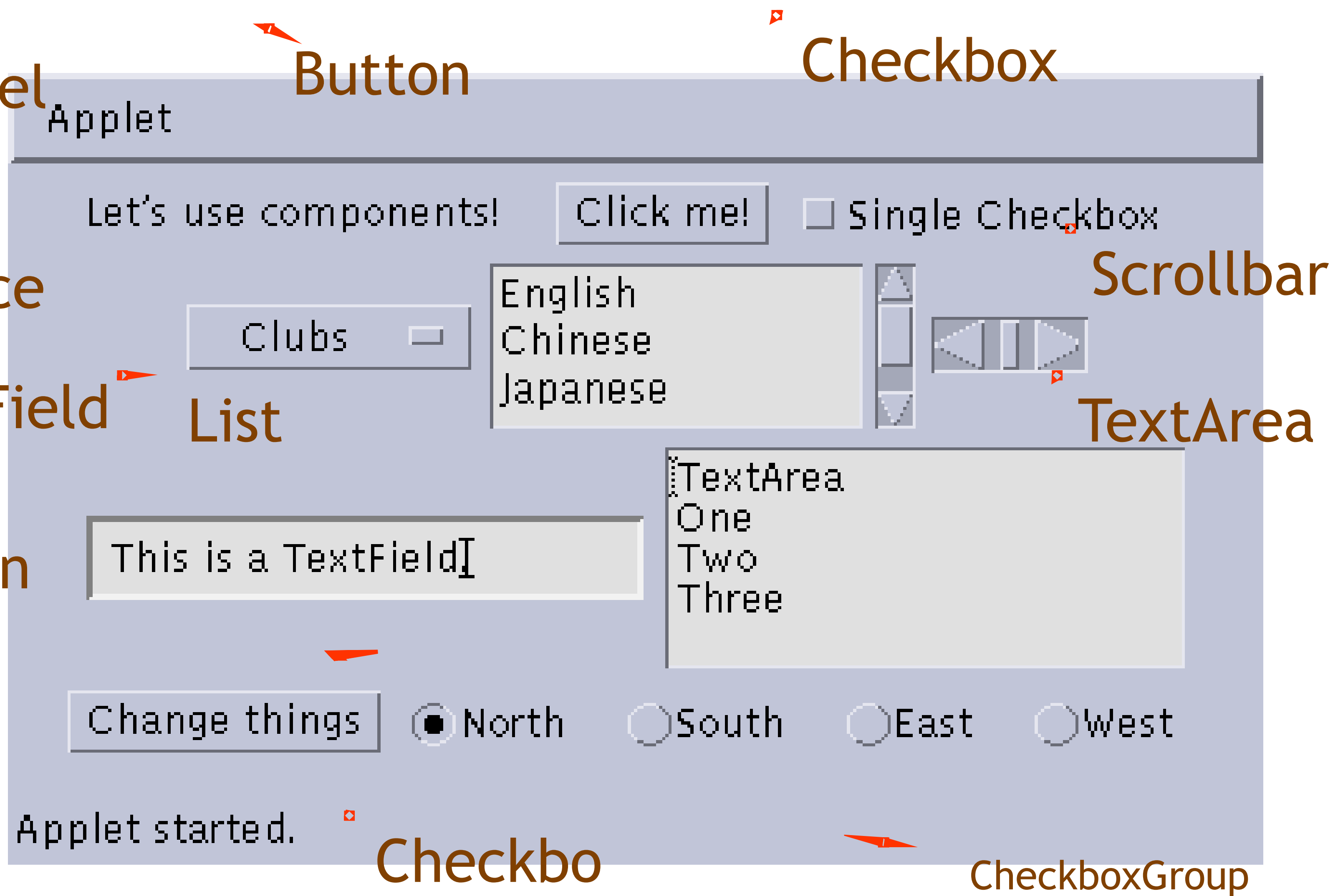
```
TextField emailTextField = new TextField();
TextField passwordTextField = new TextField();
passwordTextField.setEchoChar("*");
[…]

String userEmail = emailTextField.getText();
String userpassword = passwordTextField.getText();
```

TextArea

- This class displays multiple lines of optionally editable text.
- This class inherits several methods from TextComponent.
- TextArea also provides the methods: appendText(), insertText() and replaceText()

```
// 5 rows, 80 columns
TextArea fullAddressTextArea = new TextArea(5, 80);
[…]

String userFullAddress= fullAddressTextArea.getText();
```

Some types of components



Layout Managers

Since the Component class defines setSize() and setLocation() -methods, all Components can be sized and positioned with those methods.

- **There are several different** LayoutManagers, each of which sizes and positions its Components based on an algorithm:
  - FlowLayout
  - BorderLayout
  - GridLayout
- For Windows and Frames, the default LayoutManager is BorderLayout. For Panels, the default LayoutManager is FlowLayout.

Flow Layout

- The algorithm used by the FlowLayout is to lay out Components like words on a page: Left to right, top to bottom.
- It fits as many Components into a given row before moving to the next row.

```
Panel aPanel = new Panel();
aPanel.add(new Button("Ok"));
aPanel.add(new Button("Add"));
aPanel.add(new Button("Delete"));
aPanel.add(new Button("Cancel"));
```

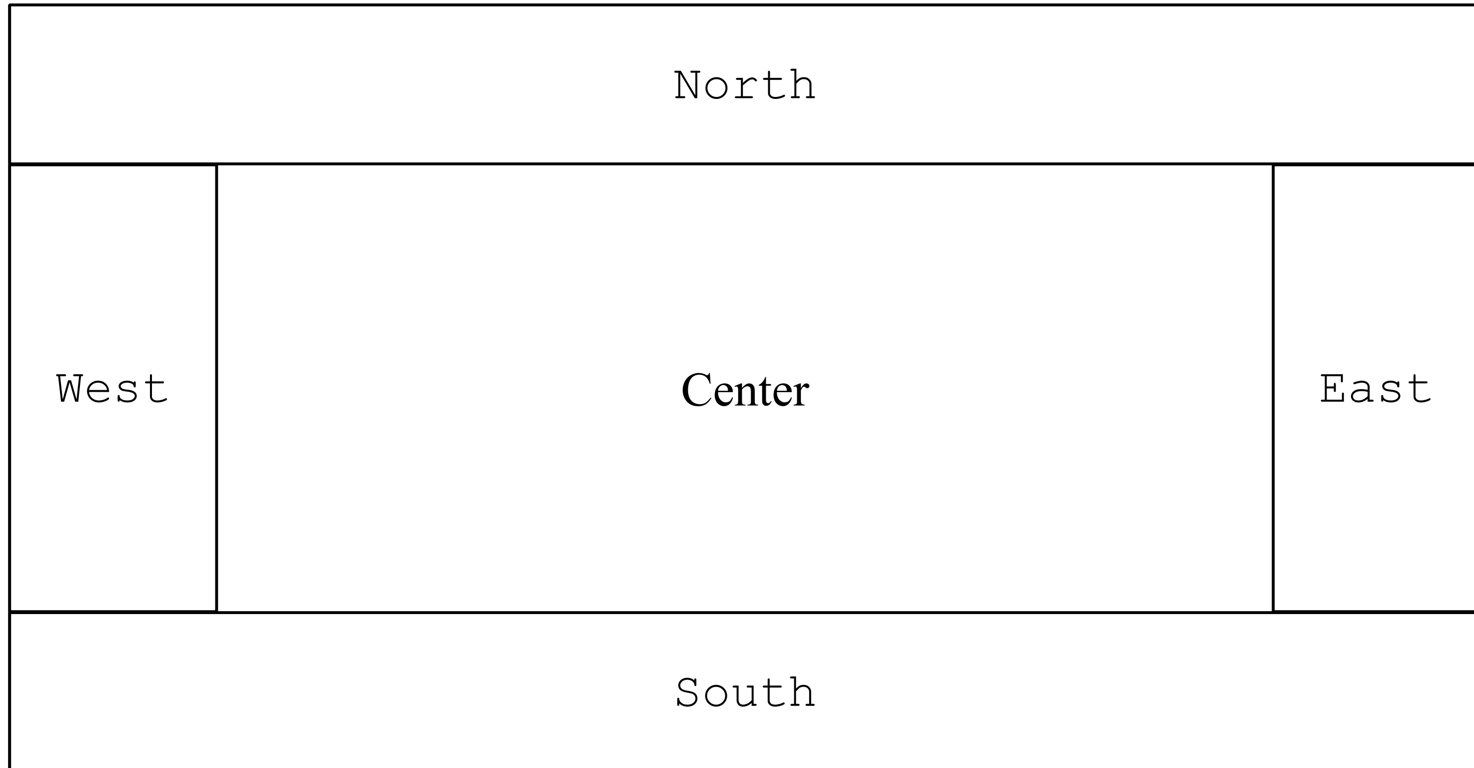
Border Layout

- The BorderLayout Manager breaks the Container up into 5 regions (North, South, East, West, and Center).
- When Components are added, their region is also specified:

```
Frame aFrame = new Frame();
aFrame.add("North", new Button("Ok"));
aFrame.add("South", new Button("Add"));
aFrame.add("East", new Button("Delete"));
aFrame.add("West", new Button("Cancel"));
aFrame.add("Center", new Button("Recalculate"));
```

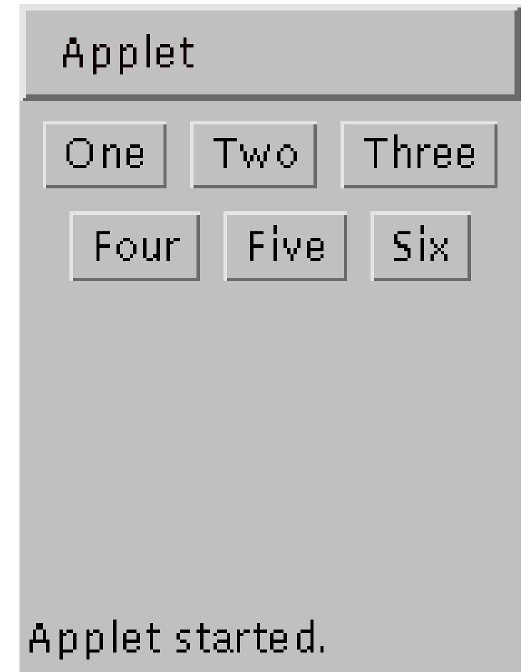
Border Layout (cont)

- The regions of the BorderLayout are defined as follows:



Complete example: Flow Layout

```
import java.awt.*;
import java.applet.*;
public class FlowLayoutExample extends Applet {
  public void init () {
    setLayout (new FlowLayout ()); // default
    add (new Button ("One"));
    add (new Button ("Two"));
    add (new Button ("Three"));
    add (new Button ("Four"));
    add (new Button ("Five"));
    add (new Button ("Six"));
  }
}
```



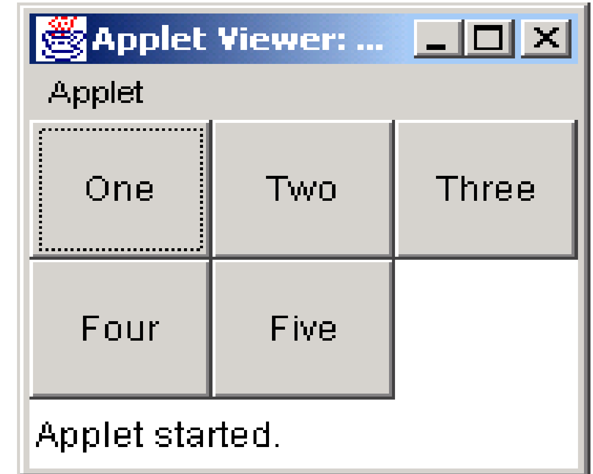
Grid Layout

- The GridLayout class divides the region into a grid of equally sized rows and columns.
- Components are added left-to-right, top-to-bottom.
- The number of rows and columns is specified in the constructor for the LayoutManager.

```
Panel aPanel = new Panel();
GridLayout theLayout = new GridLayout(2,2);
aPanel.setLayout(theLayout);
aPanel.add(new Button("Ok"));
aPanel.add(new Button("Add"));
aPanel.add(new Button("Delete"));
aPanel.add(new Button("Cancel"));
```

GridLayout

The GridLayout manager divides the container up into
a given number of rows and columns:




```
new GridLayout(rows, columns)
```

Graphics

- It is possible to draw lines and various shapes within a Panel under the AWT.
- Each Component contains a Graphics object which defines a Graphics Context which can be obtained by a call to getGraphics().
- Common methods used in Graphics include:

drawLine

drawOval

drawPolygon

drawPolyLine

drawRect

drawRoundRect

drawString

draw3DRect

fill3DRect

fillArc

- fillOval
- fillPolygon
- fillRect
- fillRoundRect
- setColor
- setFont
- setPaintMode
- drawImage

GUI BASED APPLICATION

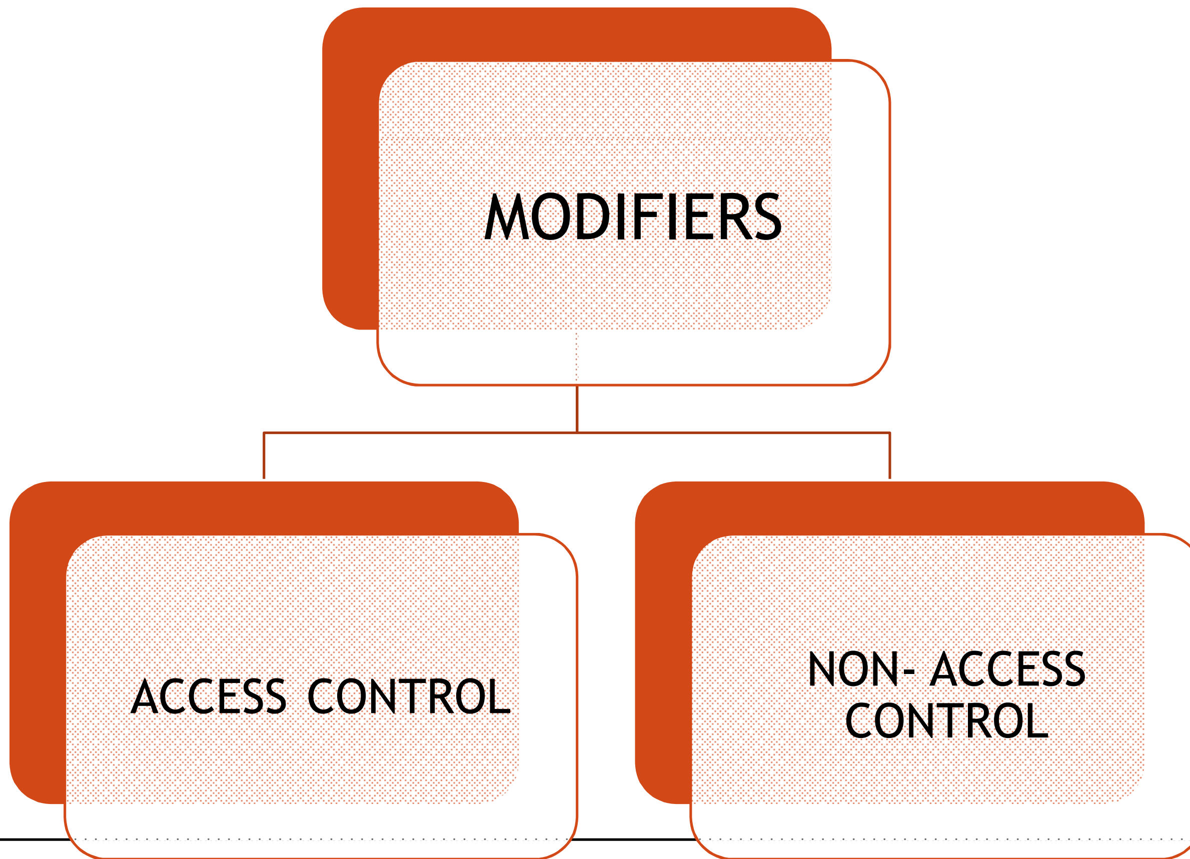
- AUTOMATED TALLER MACHINE (ATM)
- AIRLINE TICKETING SYSTEM
- INFORMATION ABOUT RAILWAY STATION
- MOBILE APPLICATION
- NAVIGATION SYSTEMS

MODIFIER

MODIFIERS :

Modifiers are KEYWORDS that are placed in a class, method or variable declaration that changes how it operates

java Modifiers are categorized in to two types



ACCESS CONTROL MODIFIER:

Java language has FOUR access modifier to control access levels for CLASSES,VARIABLE METHODS AND CONTRUCTOR

DEFAULT:

Defalt has scope only inside the same package

PUBLIC :

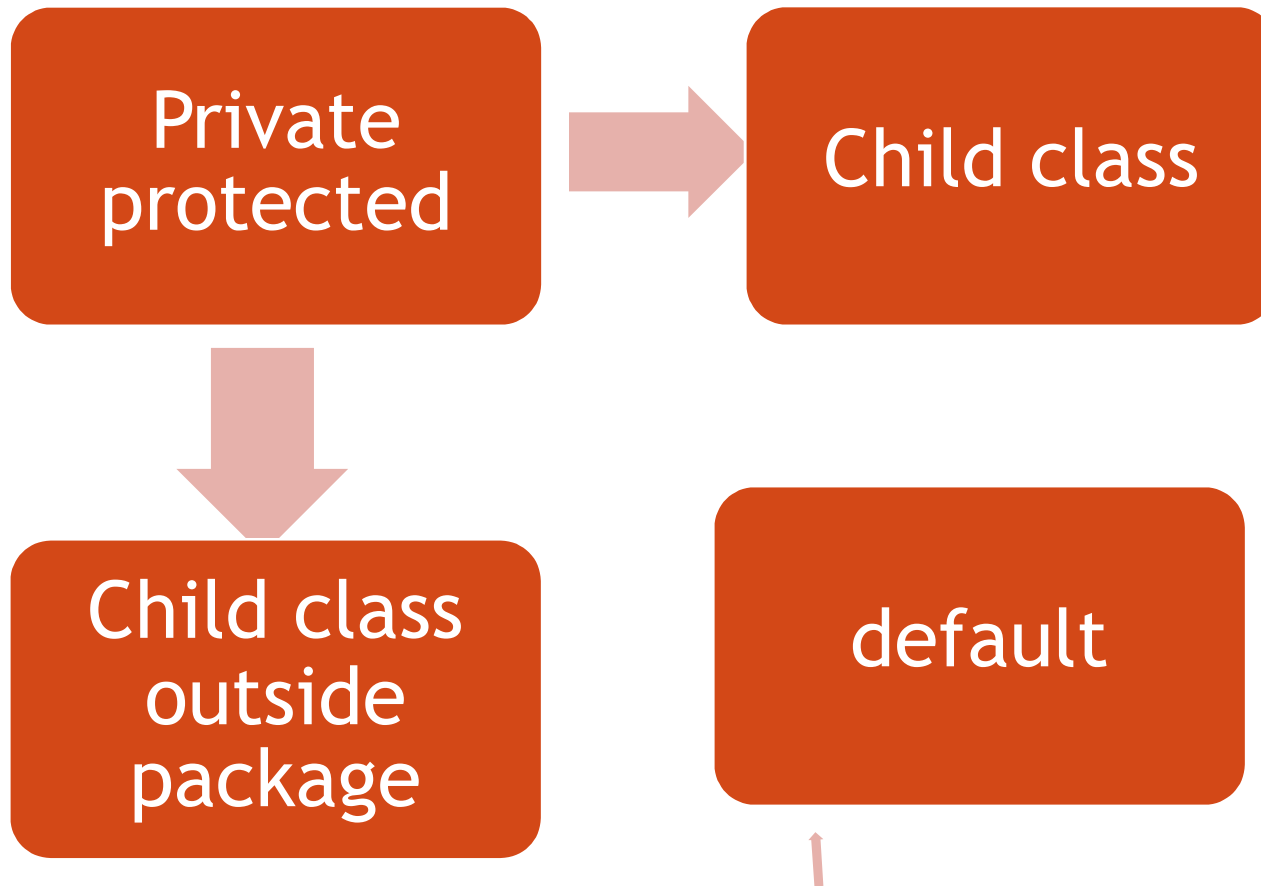
Public scope is visible everywhere

PROTECTED:

Protected has scope within the package and all sub classes

PRIVATE

private has scope only with in the classes



Public Access Modifier

- Methods, Variables and Constructor that are declared private can only be accessed within the declared class itself.
- Private access modifier is the most restrictive access level class and interfaces cannot be private.
- Variables that are declared private can be accessed outside the class if public getter methods are present in the class.
- Using the private modifiers is the main way that an object encapsulates itself and hide data from the outside world.

NON – ACCESS MODIFIER:

Non-access modifers do not change the accessibility of variables and methods,but they do provide them special properties

Non – access modifiers are five type

- FINAL
- STATIC
- TRANSIENT
- SYNCHRONIZED
- VOLATILE

FINAL MODIFIER:

Final modifier is used to declare a field as final

It prevents its content from being modified

Final field must be initialized when it is declared

STATIC MODIFIER:

Static modifier are used to create class variable and class methods which can be accessed of a class

Static with variable:

static variables are defined as a class member that can be accessed without any object of that class

Static variable has only one single storage

All the object of the class having static variable will have the same instance of static variable

Static variables are initialized only once

Static variable are used to represent common property of a class

Eg:

```
Class Employee
{
Int e-id;
String name;
Static string company –name =“study to night”;
}
```

All employee have its unique name and employee id but company name will be same all 100 employee

here company name is the common property. so if you create a class to store employee detail,company – name field will be mark as static

Static variable
vs
instance variable

Static variable	Instance variable
1.Represent common property	1.Represent unique property
2.Accessed using class name	2.Accessed using object
3.Get memory class once	3.Get new memory each time a new object is created

TRANSIENT MODIFIER :

When an instance variable is declared as transient then its value doesn't persist when an object is serialized

SYDCHRONIZED MODIFIER:

When a method is synchronized it can be accessed by only one thread at a time.

VOLATILE MODIFIER:

Volatile modifier tells the compiler that the volatile variable can be changed unexpectedly by other parts of your protgram

EXCEPTION HANDLING

WHAT IS 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.

REASONS FOR EXCEPTION

An exception can occur for many different reasons. Following are some scenarios where an exception occurs.

- A user has entered an invalid data.
- A file that needs to be opened cannot be found.
- A network connection has been lost in the middle of communications or the JVM has run out of memory.

TYPES OF EXCEPTION

Checked Exception

Unchecked Exception

CHECKED EXCEPTION

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

```
import java.io.File;
import java.io.FileReader;
public class FilenotFound_Demo
 {
  public static void main(String args[])
{
     File file = new File("E://file.txt");
     FileReader fr = new FileReader(file);
   }
}
```

OUTPUT

```
C:\>javac FilenotFound_Demo.java
FilenotFound_Demo.java:8: error:
unreported exception
FileNotFoundException; must be caught or
declared to be thrown
      FileReader fr = new FileReader(file);
                          ^
1 error
```

UNCHECKED EXCEPTION

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 or improper use of an API. Runtime exceptions are ignored at the time of compilation.

EXAMPLE

```
public class Unchecked_Demo {

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

OUTPUT

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException:5

Exceptions.Unchecked_Demo.main(Unchecked_Demo.java:8)

ERROR

These are not exceptions at all, but problems that arise beyond the control of the user or the programmer. Errors are typically ignored in your code because you can rarely do anything about an error. For example, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation.

Java Exception Handling Keywords

There are 5 keywords used in java exception handling.

- try
- catch
- finally
- throw
- throws

JAVA TRY BLOCK

Java try block is used to enclose the code that might throw an exception. It must be used within the method.

Java try block must be followed by either catch or finally block.

SYNTAX OF TRY-CATCH BLOCK

```
try{
//code that may throw exception
}catch(Exception_class_Name ref){}
```

SYNTAX OF TRY-FINALLY BLOCK

```
try{
//code that may throw exception
}finally{}
```

EXAMPLE

```
// File Name : ExcepTest.java
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");
   }
```

OUTPUT

Output:Exception thrown

:java.lang.ArrayIndexOutOfBoundsException: 3

Out of the block

JAVA CATCH BLOCK

Java catch block is used to handle the Exception. It must be used after the try block only.

You can use multiple catch block with a single try.

Problem without exception handling

TRY-CATCH BLOCK

```
public class Testtrycatch1{
  public static void main(String args[]){
      int data=50/0;//may throw exception
      System.out.println("rest of the code...");
}
}
```

OUTPUT

Output:Exception in thread main

java.lang.ArithmeticException:/ by zero

Solution by exception handling

- USE TRY-CATCH BLOCK

```
public class Testtrycatch2{
  public static void main(String args[]){
   try{
      int data=50/0;
   }catch(ArithmeticException e){System.out.println(e);}
   System.out.println("rest of the code...");
}
}
```

OUTPUT

Output:Exception in thread main

java.lang.ArithmeticException:/ by zero

rest of the code...

MULTIPLE CATCH BLOCK

If the user have to perform different tasks at the occurrence of different Exceptions, use java multi catch block.

MULTIPLE CATCH BLOCK

```
public class TestMultipleCatchBlock{
 public static void main(String args[]){
  try{
  int a[]=new int[5];
    a[5]=30/0;
   }
  catch(ArithmeticException e){System.out.println("task1 is completed");}
  catch(ArrayIndexOutOfBoundsException e){System.out.println("task 2
completed");}
   catch(Exception e){System.out.println("common task completed");}

  System.out.println("rest of the code...");
 }
}
```

OUTPUT

- Output:task1 completed
- task2 completed
- common task completed rest of the code...

JAVA FINALLY BLOCK

Java finally block is a block that is used to execute important code such as closing connection, stream etc.

Java finally block is always executed whether exception is handled or not.

Java finally block follows try or catch block.

CASE 1

EXCEPTION DOESN'T OCCUR

```
class TestFinallyBlock{
  public static void main(String args[]){
  try{
   int data=25/5;
   System.out.println(data);
  }
  catch(NullPointerException e){System.out.println(e);}
  finally{System.out.println("finally block is always executed");}
  System.out.println("rest of the code...");
  }
}
```

OUTPUT

finally block is always executed

rest of the code...

CASE 2

EXCEPTION OCCURS AND NOT HANDLED

```
class TestFinallyBlock1{
  public static void main(String args[]){
  try{
   int data=25/0;
   System.out.println(data);    }
  catch(NullPointerException e){System.out.println(e);}
  finally{System.out.println("finally block is always executed");}
  System.out.println("rest of the code...");
  }
}
```

OUTPUT

Output:finally block is always executed

Exception in thread main

java.lang.ArithmeticException:/ by zero

CASE 3

EXCEPTION OCCURS AND HANDLED

```
public class TestFinallyBlock2{
 public static void main(String args[]){
 try{
  int data=25/0;
  System.out.println(data);
 }
 catch(ArithmeticException e){System.out.println(e);}
 finally{System.out.println("finally block is always executed");}
 System.out.println("rest of the code...");
 }
}
```

OUTPUT

- Output:Exception in thread main java.lang.ArithmeticException:/ by zero
- finally block is always executed
- rest of the code...

JAVA THROW EXCEPTION

The Java throw keyword is used to explicitly throw an exception. We can throw either checked or unchecked exception in java by throw keyword. The throw keyword is mainly used to throw custom exception.

EXAMPLE

```
public class TestThrow1{
   static void validate(int age){
     if(age<18)
      throw new ArithmeticException("not valid");
     else
      System.out.println("welcome to vote");
   }
   public static void main(String args[]){
      validate(13);
      System.out.println("rest of the code...");
  }
}
```

OUTPUT

Exception in thread main

java.lang.ArithmeticException:not valid

JAVA THROWS KEYWORD

The Java throws keyword is used to declare an exception. It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.

DECLARE THE EXCEPTION

PROGRAM IF EXCEPTION DOES NOT OCCUR

```
import java.io.*;
class M{
 void method()throws IOException{
  System.out.println("device operation performed");
 }
}
class Testthrows3{
   public static void main(String args[])throws IOException{//declare
exception
     M m=new M();
     m.method();

    System.out.println("normal flow...");
  }
}
```

OUTPUT

Output:device operation performed

normal flow...

PROGRAM IF EXCEPTION OCCURS

```
import java.io.*;
class M{
 void method()throws IOException{
  throw new IOException("device error");
 }
}
class Testthrows4{
   public static void main(String args[])throws IOException{//declare exception
     M m=new M();
     m.method();

    System.out.println("normal flow...");
  }
}
```

OUTPUT

Output:Runtime Exception

Interfaces & Packages

Interfaces

- An interface is a reference type in Java, it is similar to class,it is a collection of abstract methods.
- A class implements an interface, thereby inheriting the abstract methods of the interface.
- Along with abstract methods an interface may also contain constants, default methods, static methods, and nested types.
- Method bodies exist only for default methods and static methods.
- Writing an interface is similar to writing a class. But a class describes the attributes and behaviours of an object. And an interface contains behaviours that a class implements.
- Unless the class that implements the interface is abstract, all the methods of the interface need to be defined in the class.

- An interface is similar to a class in the following ways:

- An interface can contain any number of methods.

- An interface is written in a file with a **.java** extension, with the name of the interface matching the name of the file.

- The byte code of an interface appears in a **.class** file.

- Interfaces appear in packages, and their corresponding byte code file must be in a directory structure that matches the package name.

- **Declaring Interfaces:**

The **interface** keyword is used to declare an interface. Here is a simple example to declare an interface:

```
/* File name : NameOfInterface.java */
import java.lang.*;
//Any number of import statements

public interface NameOfInterface
{
	//Any number of final, static fields
	//Any number of abstract method
declarations\
}
```

Example

```
/* File name : Animal.java */
interface Animal {

    public void eat();
    public void travel();
}
```

- **Implementing Interfaces:** A class uses the **implements** keyword to implement an interface. The implements keyword appears in the class declaration following the extends portion of the declaration.

```java
/* File name : MammalInt.java */
public class MammalInt implements Animal{

	public void eat(){
		System.out.println("Mammal eats");
	}

	public void travel(){
		System.out.println("Mammal travels");
	}

	public int noOfLegs(){
		return 0;
	}

	public static void main(String args[]){
		MammalInt m = new MammalInt();
		m.eat();
		m.travel();
	}
}
```

The program would produce the following result:

Mammal eats

Mammal travels

Packages

- Packages are used in Java in order to prevent naming conflicts, to control access, to make searching/locating and usage of classes, interfaces, numerations and annotations easier, etc.

- A Package can be defined as a grouping of related types (classes, interfaces, enumerations and annotations) providing access protection and name space management.

- Some of the existing packages in Java are::

  - java.lang - bundles the fundamental classes

  -java.io - classes for input , output functions are bundled in this package

- Programmers can define their own packages to bundle group of classes/interfaces, etc. It is a good practice to group related classes implemented by you so that a programmer can easily determine that the classes, interfaces, enumerations, annotations are related.

- Since the package creates a new namespace there won't be any name conflicts with names in other packages. Using packages, it is easier to provide access control and it is also easier to locate the related classes.

- **Creating a package:**

- While creating a package, you should choose a name for the package and include a **package** statement along with that name at the top of every source file that contains the classes, interfaces, enumerations, and annotation types that you want to include in the package.

- The **package** statement should be the first line in the source file. There can be only one package statement in each source file, and it applies to all types in the file.

- If a **package** statement is not used then the class, interfaces, enumerations, and annotation types will be placed in the current default package.

- To compile the Java programs with package statements you have to do use -d option as shown below:

```
javac -d Destination_folder file_name.java
```

- Then a folder with the given package name is created in the specified destination, and the compiled class files will be placed in that folder

- Let us look at an example that creates a package called animals. It is a good practice to use names of packages with lower case letters to avoid any conflicts with the names of classes, interfaces.

- Below given package example contains interface named animals:

```
/* File name : Animal.java */
package animals;
interface Animal {
        public void eat();
        public void travel();
}
```

- Now, let us implement the above interface in the same package animals:

```java
package animals;

/* File name : MammalInt.java */
public class MammalInt implements Animal{

    public void eat(){
        System.out.println("Mammal eats");
    }

    public void travel(){
        System.out.println("Mammal travels");
    }

    public int noOfLegs(){
        return 0;
    }

    public static void main(String args[]){
        MammalInt m = new MammalInt();
        m.eat();
        m.travel();
    }
}
```

- Now compile the java files as shown below:

```
$ javac -d . Animal.java
$ javac -d . MammalInt.java
```

- Now a package/folder with the name animals will be created in the current directory and these class files will be placed in it.

- You can execute the class file with in the package and get the result as shown below.

```
$ java animals.MammalInt
ammal eats
ammal travels
```

UNIT-IV-**Multithreading**

Threads

A thread is a lightweight sub process, a smallest unit of processing. It is a separate path of execution.

Threads are independent, if there occurs exception in one thread, it doesn't affect other threads. It shares a common memory area.

Thread Methods

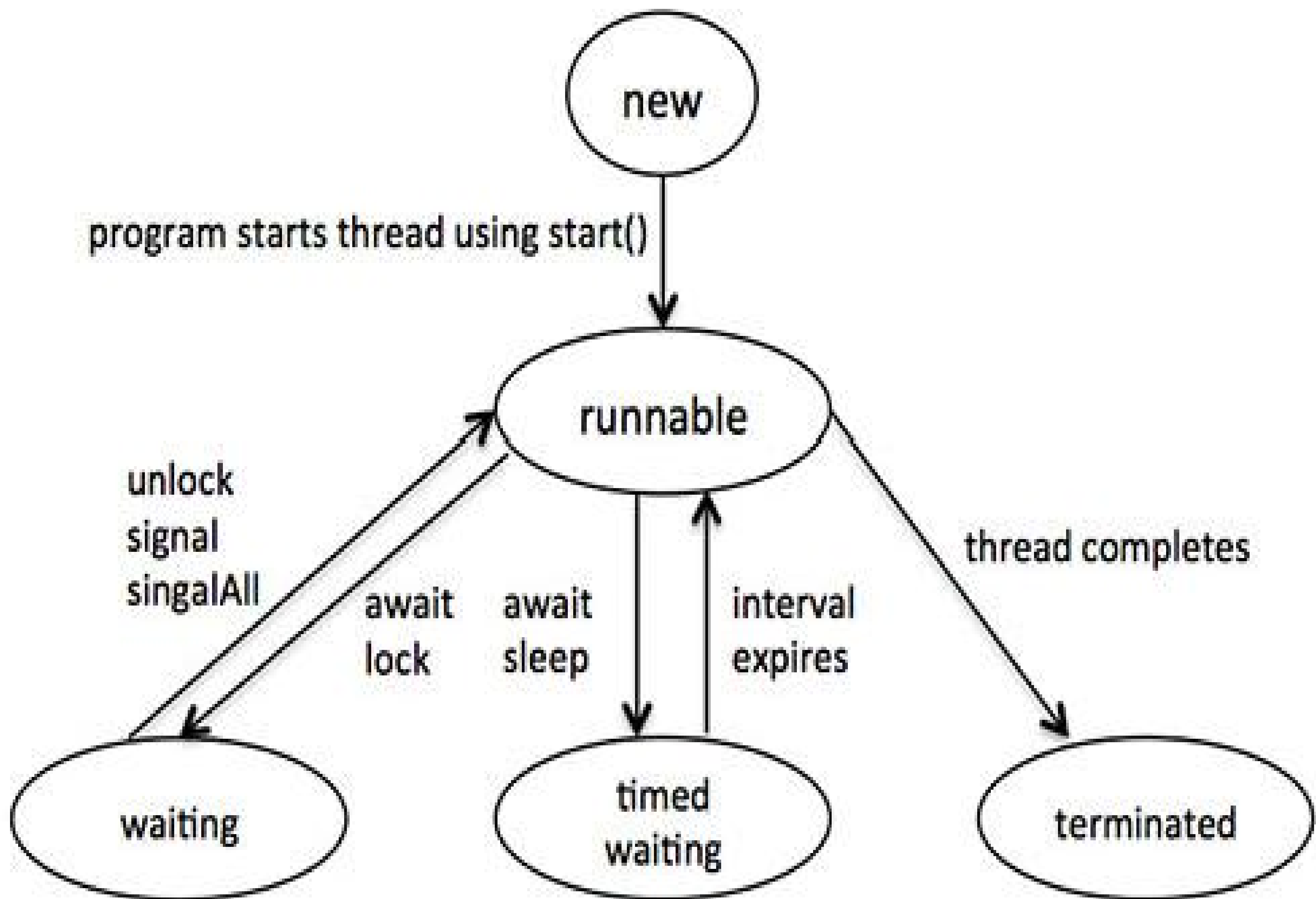
public void start()

public void run()

public static void sleep(long millisec)

public static boolean holdsLock(Object x)

public final void join(long millisec)ect...


new
program starts thread using start()
runnable
unlock
signal
singalAll
await
lock
await
sleep
interval
expires
thread completes
waiting
timed
waiting
terminated

Java Multithreading

Multitasking is a process of executing multiple tasks simultaneously.

We use multitasking to utilize the CPU. Multitasking can be achieved by two ways:

- Process-based Multitasking(Multiprocessing)
- Thread-based Multitasking(Multithreading)

Process-based Multitasking (Multiprocessing)

Each process have its own address in memory i.e. each process allocates separate memory area.

Process is heavyweight.

Thread-based Multitasking (Multithreading)

Threads share the same address space.

Thread is lightweight.

Major Java Multithreading Concepts

- While doing Multithreading programming in Java, you would need to have the following concepts very handy:

- Handling threads inter communication
- Handling thread deadlock
- Major thread operations

Thread synchronization

Synchronized block can be used to perform synchronization on any specific resource of the method.

Suppose you have 50 lines of code in your method, but you want to synchronize only 5 lines, you can use synchronized block.

If you put all the codes of the method in the synchronized block, it will work same as the synchronized method.

Syntax to use synchronized block

```
synchronized (object reference expression)
{

  //code block

}
```

Handling threads inter communication

public void wait()

Causes the current thread to wait until another thread invokes the notify().

public void notify()

Wakes up a single thread that is waiting on this object's monitor.

public void notifyAll()

Wakes up all the threads that called wait( ) on the same object.

Handling threads deadlock

Deadlock in java is a part of multithreading.

Deadlock can occur in a situation when a thread is waiting for an object lock, that is acquired by another thread and second thread is waiting for an object lock that is acquired by first thread.

Since, both threads are waiting for each other to release the lock, the condition is called deadlock.

Major thread operations

public void wait()

Causes the current thread to wait until another thread invokes the notify().

public void notify()

Wakes up a single thread that is waiting on this object's monitor.

public void stop()

This method stops a thread completely.

A Simple Example

```
class MultithreadingDemo implements Runnable{
public void run(){
System.out.println("My thread is in running state.");
}
public static void main(String args[]){
MultithreadingDemo obj=new MultithreadingDemo();
Thread tobj =new Thread(obj);
tobj.start();
}
}
```

Output:

My thread is in running state.

Advantages:

- general: better *use of system resources*
- parallelize tasks enhanced performance on multiprocessor machines
- multithreaded servers and interactive GUIs: better availability

Disadvantages

- general: ***increased complexity***
- synchronization of shared resources (objects, data)
- difficult to debug, result is sometimes unpredictable
- potential deadlocks "starvation": some threads may not be served

with a bad design

constructing and synchronizing threads is CPU/memory intensive

STREAMS and INPUT / OUTPUT

Overview of I/O Streams

To bring in information, a program opens a *stream* on an information source (a file, memory, a socket) and reads the information sequentially, as shown in the following figure.

Streams

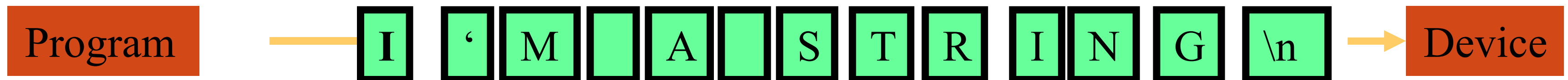
- ***Stream***: an object that either delivers data to its destination (screen, file, etc.) or that takes data from a source (keyboard, file, etc.)
  - it acts as a buffer between the data source and destination
- ***Input stream***: a stream that provides input to a program
  - System.in is an input stream
- ***Output stream***: a stream that accepts output from a program
  - System.out is an output stream
- A stream connects a program to an I/O object
  - System.out connects a program to the screen
  - System.in connects a program to the keyboard

- The I/O System in Java is based on Streams
  - Input Streams are data sources
    - Programmers read data from input streams
  - Output Streams are data sinks
    - Programmers write data to output streams

- **Java has two main types of Streams**
  - Byte Oriented
    - Each datum is a byte
    - uses InputStream class hierarchy & OutputStream class hierarchy
  - Character-based I/O streams
    - each datum is a Unicode character
    - uses Reader class hierarchy & Writer class hierarchy

Streams

- JAVA distinguishes between 2 types of streams:



- Text – streams, containing ‘characters‘

•Binary Streams, containing 8 – bit information

Streams

- Streams in JAVA are Objects, of course !

- Having

- 2 types of streams (text / binary) and
- 2 directions (input / output)

- results in 4 base-classes dealing with I/O:

1. Reader: text-input
2. Writer: text-output
3. InputStream: byte-input
4. OutputStream: byte-output

Streams

- InputStream, OutputStream, Reader, Writer are abstract classes

- Subclasses can be classified by 2 different characteristics of sources / destinations:
  - For final device (data sink stream)

    purpose: serve as the source/destination of the stream (these streams 'really' write or read !)
  - for intermediate process (processing stream)

    Purpose: alters or manages information in the stream (these streams are 'luxury' additions, offering methods for convenient or more efficient stream-handling)

I/O: General Scheme

- In General:
- Reading (writing):
  - open an input (output) stream
  - while there is more information

    read(write) next data from the stream
  - close the stream.

- In JAVA:
  - Create a stream object and associate it with a disk-file
    - Give the stream object the desired functionality
  - while there is more information

    read(write) next data from(to) the stream
  - close the stream.

Binary Files

- Stores binary images of information identical to the binary images stored in main memory
- Binary files are more efficient in terms of processing time and space utilization
- drawback: not ‘human readable‘, i.e. you can‘t use a texteditor (or any standard-tool) to read and understand binary files

Binary Files

- Example: writing of the integer ’42‘
- TextFile: ‘4‘ ‘2‘ (internally translated to 2 16-bit representations of the characters ‘4‘ and ‘2‘)
- Binary-File: 00101010, one byte
- (= 42 decimal)

Writing Binary Files

- Class: FileOutputStream
- ... see FileWriter
- The difference:
- No difference in usage, only in output format

Reading Binary Files

- Class: FileInputStream
- ... see FileReader
- The difference:
- No difference in usage, only in output format

Binary vs. TextFiles

	pro	con
Binary	Efficient in terms of time and space	Preinformation about data needed to understand content
Text	Human readable, contains redundant information	Not efficient

Image filter

Image

- In the world of computers, there are many types of images, each of which is associated with a specific file format.

These image types are usually identified by
their file extensions, which include PCX, BMP,GIF, JPEG (or JPG), TIFF (or TIF), TGA, and more.

- Each of these file types was created by third-party software companies for use with their products, but many became popular enough to grow into standards.

Image filter

- Image filtering is sometimes referred to as image processing.

- Most popular graphical paint programs contain image processing features, such as sharpening or softening an image.

- Image filters can be plugged in to enhance images during album generation, adding watermarking, logos etc.

The Filters

- Color Adjustment Filters
- Distortion and Warping Filters
- Effects Filters
- Texturing Filters
- Blurring and Sharpening Filters

Color Adjustment Filters

ChannelMixFilter Mixes the RGB channels

ContrastFilter Adjusts brightness and contrast

Distortion and Warping Filters

BicubicScaleFilter Scaling with bucolic interpolation

CircleFilter Wrap an image around a circle

Effects Filters

BlockFilter Mosaic or pixellate an image

CrystallizeFilter Make an image look like stained glass

Texturing Filters

CheckFilter Draw a checkerboard pattern

FBMFilter Fractal Brownian motion texturing

Blurring and Sharpening Filters

BlurFilter Simple blur

VariableBlurFilter Blurring with a variable radius taken from a mask

The Image Filter Classes

- ImageProducer
- FilteredImageSource
- MemoryImageSource
- ImageConsumer
- PixelGrabber
- ImageFilter
- RGBImageFilter
- CropImageFilter

The ColorFilter class.

```
class ColorFilter extends RGBImageFilter {

boolean red, green, blue;
public ColorFilter(boolean r, boolean g, boolean b)

{
red = r; green = g; blue = b; canFilterIndexColorModel = true;

}
public int filterRGB(int x, int y, int rgb) {

// Filter the colors
int r = red ? 0: ((rgb >> 16) & 0xff);
 int g = green ? 0: ((rgb >> 8) & 0xff);

 int b = blue ? 0: ((rgb >> 0) & 0xff);

 // Return the result
 return (rgb & 0xff000000) | (r << 16) | (g << 8) | (b << 0);

}
}
```

An Alpha Image Filter

The AlphaFilter class filters the alpha components of an image according to the alpha level you supply in its constructor.

Listing contains the source code for the AlphaFilter class AlphaFilter.java .

The AlphaFilter class.

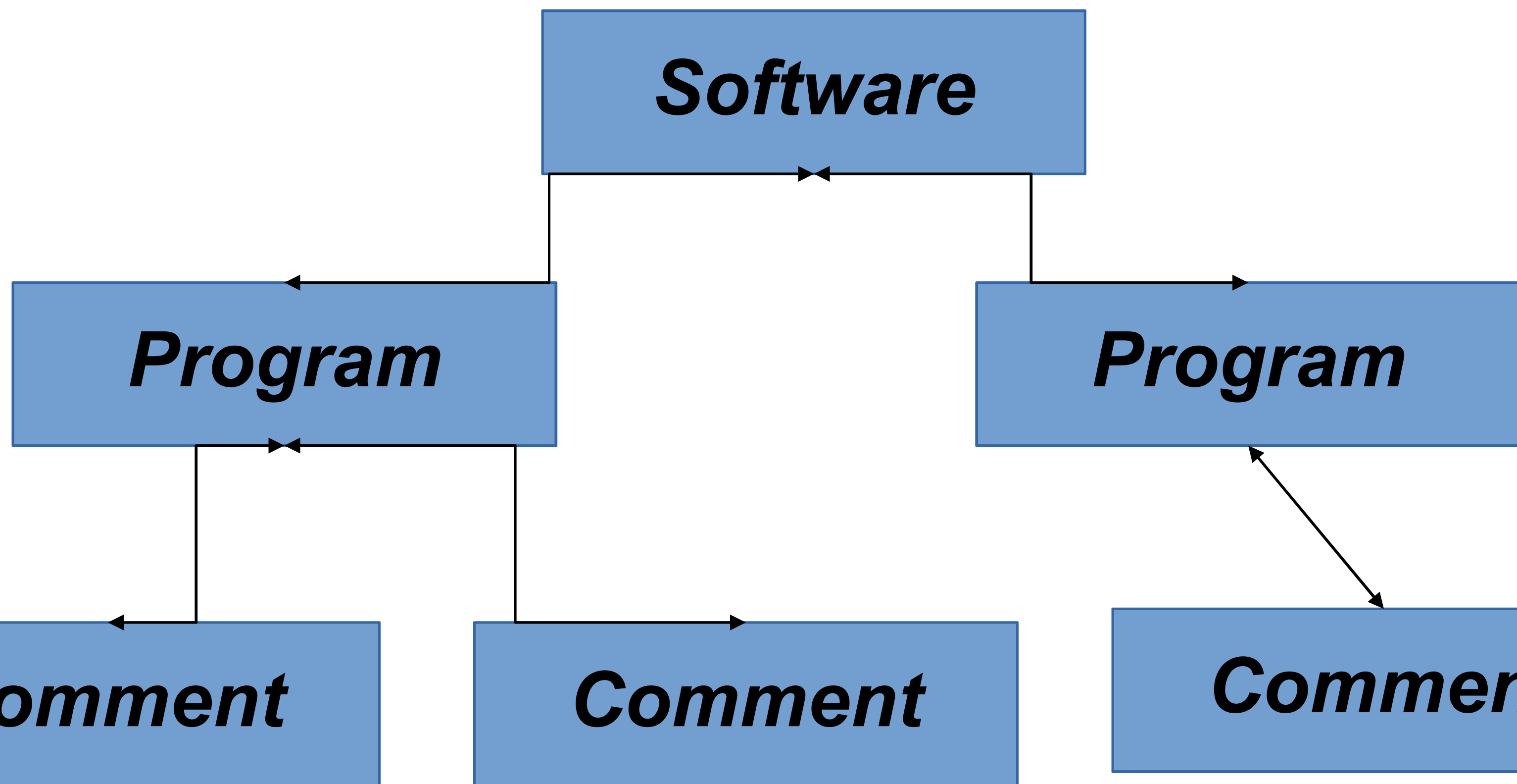
```
class AlphaFilter extends RGBImageFilter {

int alphaLevel;
public AlphaFilter(int alpha) {
alphaLevel = alpha;
canFilterIndexColorModel = true;
}
public int filterRGB(int x, int y, int rgb) {
// Adjust the alpha value
 int alpha = (rgb >> 24) & 0xff;
 alpha = (alpha * alphaLevel) / 255;

// Return the result
 return ((rgb & 0x00ffffff) | (alpha << 24));

}
 }
```

Java programming tools

Introduction

Java programming language was originally developed by Sun Microsystems which was initiated by James Gosling and released in 1995 as core component of Sun Microsystems' Java platform (Java 1.0).

s of December 2008, the latest release of the Java Standard Edition is 6.


Software
Program
Program
Comment
Comment
Comment

Few Essential Tools

A good text editor to write and edit the program.

A nice debugger to debug the program.

A memory detector in case you are using dynamic memory allocation.

Putty to connect to a remote machine.

WinSCP or FileZilla to ftp files on a remote machine.

Java-based Tools and Packages

- BioWidget
- JaMBW
- Jmol

BioWidget

BioWidget(http://www.cbil.upenn.edu/bioWidgets/) There is an effort to create and share Graphical User Interface(GUI) components for the display and visualization of genomic data, potentially over the World Wide Web.

JaMBW

Java based Molecular Biologist's Workbench

3D Molecular Modelling
DNA Sequence Analysis
RNA Analysis
Protein Analysis
Biochemical Engineering
PCR
Sequence Format Conversion Utilities
Phylogenetic Analysis
Plasmid Map
Image Analysis
Sequence Alignment & Analysis
Bio-Chip

Jmol

Jmol: an open-source Java viewer for chemical structures in 3D.

The JmolApplet is a web browser applet that can be integrated into web pages.

The Jmol application is a standalone Java application that runs on the desktop.

The JmolViewer is a development tool kit that can be integrated into other Java applications.

UNIT-V-VISUAL BASIC

Introduction to Visual Basic

Hands-On Exercise 1

- Start Microsoft Visual Basic 6.0
- Open the Welcome Project
- Open the Welcome Form
- Run the Welcome Project
- End the Welcome Project
- Exit Visual Basic

VBA - Excel

- VBA is Visual Basic for Applications
- The goal is to demonstrate how VBA can be used to leverage the power of Excel
  - VBA syntax and usage
  - the Excel VB programming environment
  - the Excel object model
  - an application

VBA - Excel

- What advantage is there in using VBA
  - extend Excel – new functions
  - makes it possible to use the Execl GUI environment
  - makes it possible to respond to events (mouse, …)
  - makes Windows programming API accessible
  - Excel can be used to control Automation servers (other software components that expose an API through COM)
  - by understanding how to use the Excel object model with VBA it is a small step to using Excel as an Automation server (controlled by other program)

- In order to run VBA code your security settings must be properly set
  - Tools | Macro | Security…
  - At least Medium security must be set – each macro will require user verification to run
- Signed code can be run in all cases

VBA – The Basics

- Data types
  - Integer 2 byte integer
  - Long 4 byte integer
  - Single 4 byte floating point
  - Double 8 byte floating point
  - Currency 8 byte real
  - String upto 64K characters
  - Byte 1 byte
  - Boolean 2 byte true or false
  - Date 8 bytes
  - Object 4 bytes – an object reference
  - Variant 16 bytes + 1 byte / character

VBA – The Basics

- The variant data type is special – a variant can hold any type of data
- A variable declared as variant (the default) can hold anything
- The actual type of the data is kept in the data
- It adds flexibility but at a cost – it requires more processing at compute time to determine what it is and how to handle it

VBA – The Basics

- Variables
  - must start with a letter
  - can contain _ and numbers
  - cannot exceed 255 characters in length
- Within a procedure declare a variable using

```
Dim variable
Dim variable As type
```

- If a variable is not declared it will be created when used, the type will be Variant
- Use Option Explicit in the declarations section to require declaration of variables
- VBA variables have scope restrictions
  - variables declared in a procedure are local to that procedure
  - variables declared in a module can be public or private

VBA – The Basics

- String variables

```
Dim variable As String
Dim variable As String * 50
```

- The first form is variable length
- The second form is limited to 50 characters
  - the variable will be space filled if string is < 50 characters
  - the string will be truncated if the contents are > 50 characters
  - the Trim and RTrim functions are useful for working with fixed length strings
- Boolean variables contain either True or False

VBA – The Basics

- The Object type is used to store the address (a reference) of an object

```
Dim variable As Object
```

  - this form d[illegible]
  - this is referred to as *late-binding*, the object types are checked at runtime (slower)
- The declaration of a specific object is

```
Dim variable As Worksheet
```

  - this form will only store Excel Worksheet objects, an attempt to put anything else into it will result in an error
  - this is referred to as *early-binding*, the object types are checked at compile time (faster)

VBA – The Basics

- Arrays are declared using

```
Dim A (1 To 10) As Double
Dim B (1 To 10, 1 To 10) As Double
Dim C (4,4,4) As Integer
Dim D () As Double
```

- Arrays can be multidimensional
- The lower bound starts at zero
  - can explicitly specify lower bound
  - can use Option Base command to reset to something other than 0

```
Option Base 1
```

- The last form above is a dynamic array – it must be dimensioned using ReDim before it can be used
- Use ReDim Preserve to retain any existing entries in array - only the upper bound of array can be changed

VBA – The Basics

- Constants are declared using

```
Const pi = 3.14159
Const pi As Double = 3.14159
```

- Constants have the same scope limitations as variables

VBA – The Basics

- User defined data types
  - can only be defined in the declarations section of a Module
  - can be Public or Private in scope

```
Public Type SystemInfo
    CPU As Variant
    Memory As Long
    ColorBits As Integer
    Cost As Currency
    PurchaseDate As Date
End Type
```

- Declare variable with this type

```
Dim MySystem As SystemInfo
```

- Referencing fields

```
MySystem.CPU = "Pentium"
If MySystem.PurchaseDate > #1/1/2006#
Then
      …
End If
```

VBA – The Basics

```
Dim a, b, c As Integer
```

- it is equivalent to

```
Dim a As Variant
Dim b As Variant
Dim c As Integer
```

VBA – The Basics

- Objects
- VBA can use pre-defined objects – such as intrinsic Excel objects
- VBA can create user-defined objects – Class Modules

- Declaring a variable to contain an object

```
Dim variable As class
Dim variable As New class
```

  - the first form declares that the variable will contain a reference to an object of the named class
  - the second form declares the variable then creates an instance of the class
- To instantiate a class

Set variable = New class

VBA – The Basics

- Objects
- To declare a variable that will refer to an instance of the Excel Worksheet class

```
Dim ws1 As Worksheet
```

- To put a reference into it

```
Set ws1 = Worksheets(“Sheet1”)
```

- This fragment will print the name of the worksheet “Sheet1”

```
Dim ws1 As Worksheet
Set ws1 = Worksheets("sheet1")
Debug.Print ws1.Name
```

VBA – The Basics

- Objects - Collections
- There is a special form of objects known as Collections
- They contain references to other objects and collections
- It is the mechanism by which the object hierarchy is defined
- By convention, collection names are usually plural
  - Workbooks – list of Workbook objects
  - Worksheets – list of Worksheet objects
  - Range – list of objects that represent cells, columns, rows
- The following example iterates through Workbooks collection

```
For Each ws In Worksheets
    Debug.Print ws.Name
Next
```

VBA – The Basics

- Statements
- VBA implements common programming statements
  - logical statements
  - looping statements
  - expressions

VBA – The Basics

- Statements
- VBA implements common programming statements
  - logical statements
  - looping statements
  - expressions

VBA – The Basics

```
If a>10 Then
   …
End If
```

- Logical statements
- The If Then Else statement is the basic logic test

```
If a>10 Then
   …
Else
   …
End If
```

```
If a>10 Then
   …
ElseIf a<0 Then
   …
Else
   …
End If
```

VBA – The Basics

- Logical statements
- The Select statement can be used to replace a multi-way if statement

```
Select Case expression
    Case expr1
        …
    Case expr2
        …
    Case Else
        …
End Select
```

VBA – The Basics

- Loop statements
- Various Do loop forms

```
Do While expr
     …
Loop
```

```
Do Until expr
     …
Loop
```

```
Do
     …
Loop While expr
```

```
Do
     …
Loop Until expr
```

VBA – The Basics

- Loop statements
- A common For loop

```
For i=1 To 10
   Debug.print i
Next i
```

```
For i=1 To 10 Step 2
   Debug.print i
Next i
```

VBA – The Basics

- Loop
- Anot

```
For Each element In group
    …
Next element
```

```
For Each ws In Worksheets
    Debug.Print ws.Name
Next
```

- Com

VBA – The Basics

- Procedures
- Procedures in VBA are either Macros or Functions
  - a macro does not return a value

```
Sub Name()
  …
End Sub
```

  - a function will return a value

```
Function Name() As Double
  …
End Sub
```

  - Property functions (Get and Let) are used in Class Modules to provide access to private properties

VBA – The Basics

- Dealing with runtime errors
- The On Error statement will trap errors

```
…
On Error GoTo label
```

- The *error* name is a label in the code

```
…
On Error GoTo check
…
check:
…
```

- In the error code a Resume statement will cause the statement that caused the error to be executed again
- In the error code a Resume Next statement will restart execution on the statement after the one that caused the error

Assigning a Variable

- To assign a value is simple…
  - Ex:

ResidenceStatus = 0

- You can do the same with a string
  - Ex:

lblDateString.Text = txtDayOfWeek.Text & ", " _

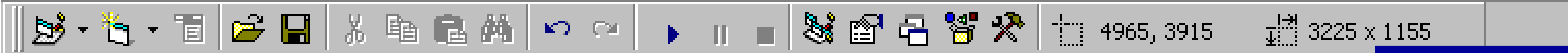
*What this does is that lblDateString.Text will have the same text as txtDayofWeek.Text

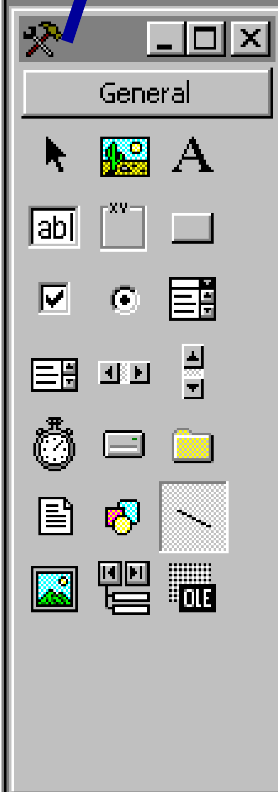
Elements of the Integrated Development Environment

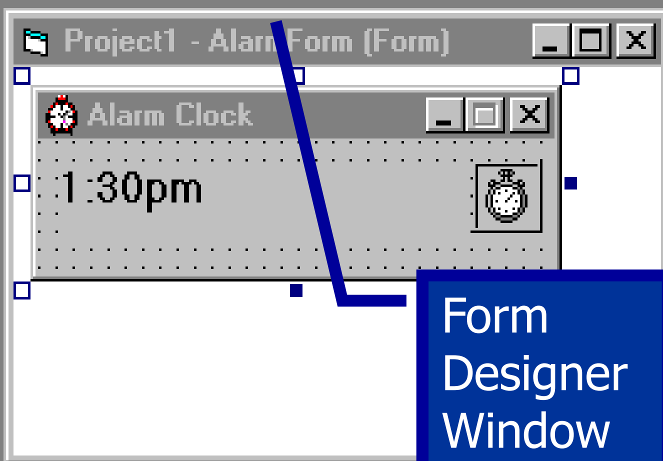
- Application icon
- Application name
- Context menus
- Controls
- Current project
- Design view mode
- Title bar

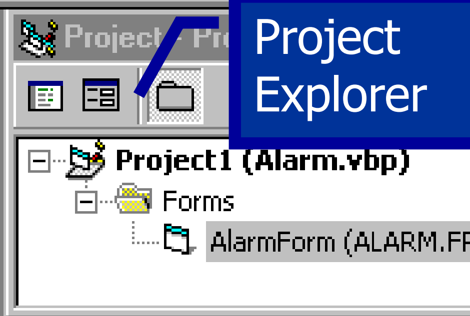
- Major windows
  - Form Designer
  - Form Layout
  - Toolbox
  - Project Explorer
  - Object Browser
  - Properties
  - Code Editor
  - Immediate, Locals, Watch

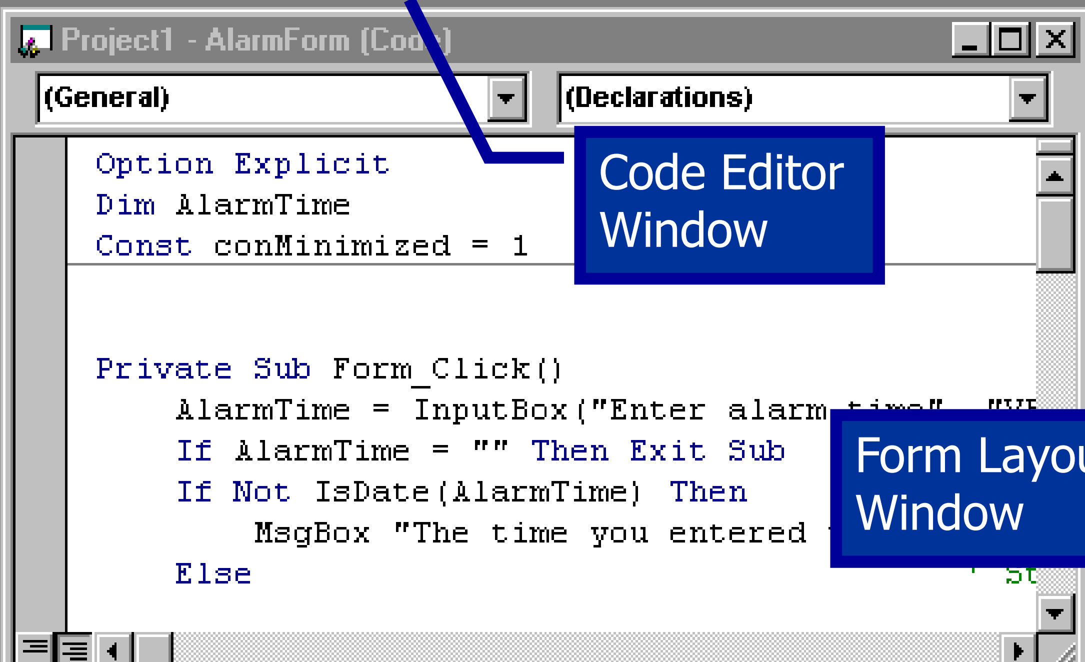
Visual Basic 6 Interactive Development Environment

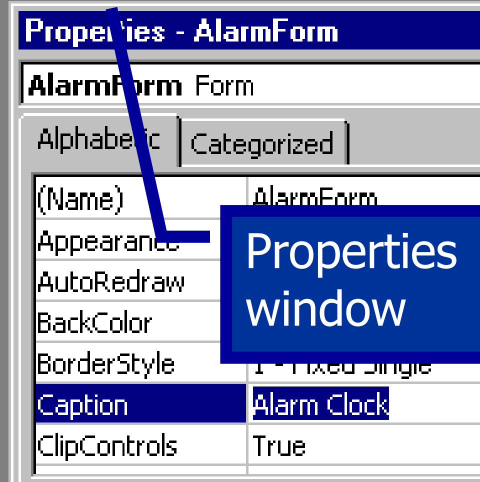


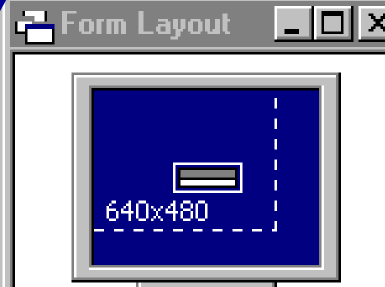












Performing calculations

- Operators:
  - + is addition
  - - is subtraction
  - **MOD* gets remainder
  - / is **floating point* division
  - \ is **Integer division*
  - **^ is exponentiation*
- Examples

  Integer = 5 + 5;

  Integer = 5 MOD 5;

  (remainder shows)

  Integer = 5 / 5;

Converting between variable types

Conversion

- Convert between items
- Use a Ctype function

Example

*Dim anInteger As Integer = 54

MsgBox (CStr(anInteger))

This will convert anInteger = 54 to output as a string “54”

If Then and If…Else…Then Statements

Both statements are conditional statements

- Visual Basic is unique on this
- If… Then is one condition is met
- If… Else… Then is if one condition is not met then another event will trigger

Example

```
Private Sub chkRush_Click(ByVal sender
    As Object, ByVal e As
    System.EventArgs) Handles
    chkRush.Click
      If chkRush.Checked Then
          rbtUPS.Enabled = True
          rbtSpecialCarrier.Enabled = True
      Else
          rbtUPS.Enabled = False
          rbtSpecialCarrier.Enabled = False
      End If
   End Sub
```

UNIT-VI-XML

Definition

- A cross-platform, software- and hardware-independent tool for storing and transmitting information.
- XML is a software and hardware independent tool for carrying information.

Introduction

- The **Extensible Markup Language** (**XML**) is a general-purpose markup language. It is classified as an extensible language because it allows its users to define their own elements.
- Its primary purpose is to facilitate the sharing of structured data across different information systems, particularly via the Internet.
- It is used both to encode documents and serialize data.

XML

- XML stands for Extensible **M**arkup **L**anguage
- XML is a **markup language** much like HTML
- XML was designed to **carry data**, not to display data
- XML tags are not predefined. Users must **define their own tags**
- XML is designed to be **self-descriptive**
- XML is a **W3C Recommendation**
- XML is fee-free open standard
- A tag-based meta language
- Designed for structured data representation
- Represents data hierarchically (in a tree)

XML Rules

- Tags are enclosed in angle brackets.
- Tags come in pairs with start-tags and end-tags.
- Tags must be properly nested.
  - **<name><email>…</name></email> is not allowed.**
  - **<name><email>…</email><name> is.**
- Tags that do not have end-tags must be terminated by a '/'.
  -
 is an html example.

More XML Rules

- Tags are case sensitive.
  - **<address> is not the same as <Address>**
- XML in any combination of cases is not allowed as part of a tag.
- Tags may not contain ‘<‘ or ‘&’.
- Tags follow Java naming conventions, except that a single colon and other characters are allowed. They must begin with a letter and may not contain white space.
- Documents must have a single *root* tag that begins the document.

The Difference Between XML and HTML

- XML is **not a replacement** for HTML.

  XML and HTML were designed with **different goals**:

- **XML** was designed to **transport** and **store data**, with focus on **what data is**.

  HTML was designed to **display data**, with focus on **how data looks**.

- HTML is *about displaying information*, while XML is *about carrying information*.

XML Important

Plain Text

- Easy to edit
- Platform independent

Data Identification

- Tell you what kind of data you have
- Can be used in different ways by different applications

Easily Processed

- Vendor-neutral standard

XML Important

- ***Stylability***
  - Inherently style-free
  - XSL---Extensible Stylesheet Language
  - Different XSL formats can then be used to display the same data in different ways
- ***Inline Reusability***
  - Can be composed from separate entities
  - Modularize your documents

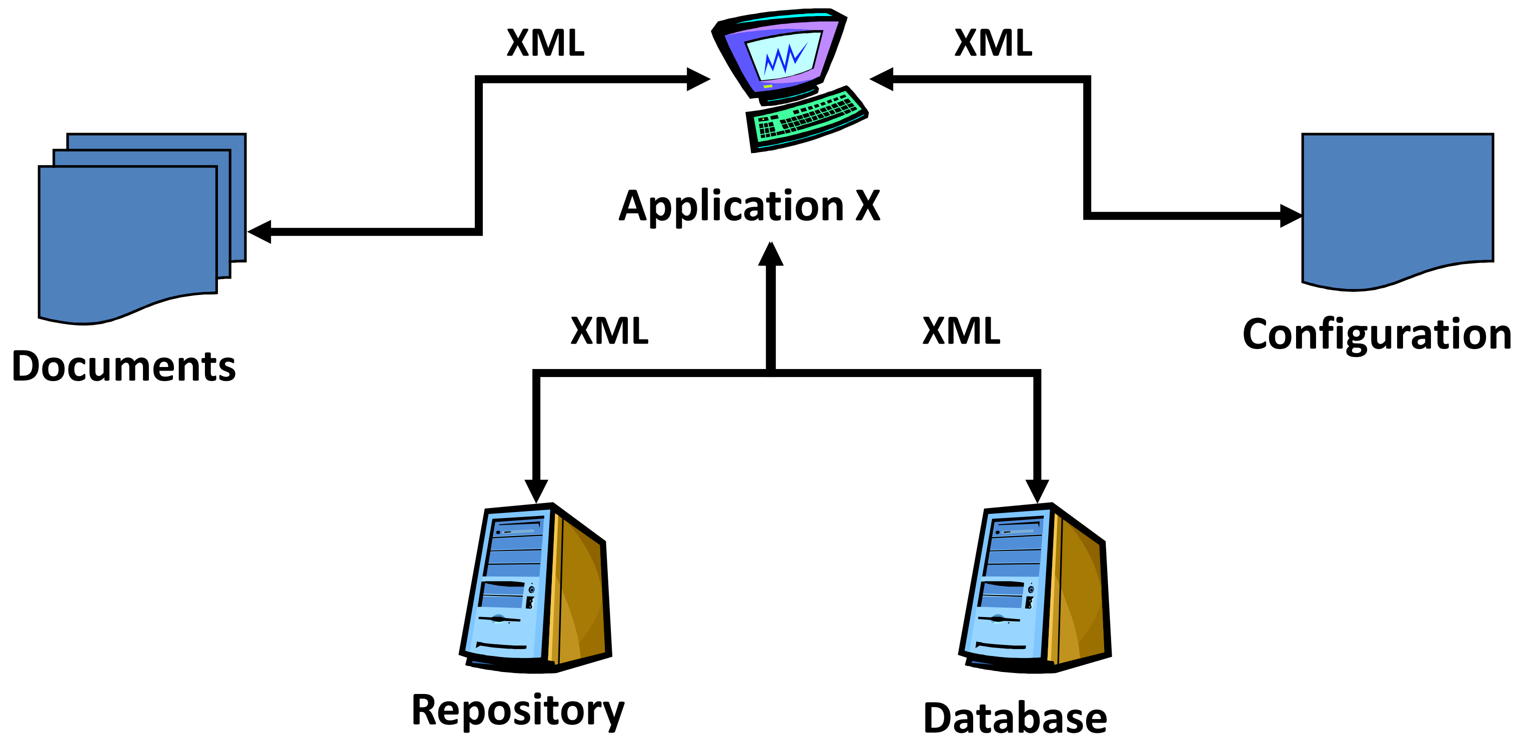
XML important

- ***Linkability -- XLink and XPointer***
  - Simple unidirectional hyperlinks
  - Two-way links
  - Multiple-target links
  - “Expanding” links

- ***Hierarchical***
  - Faster to access
  - Easier to rearrange

XML

- XML is a “use everywhere” data specification



XML and Structured Data

```
<PURCHASE_ORDER>
<PO_NUM> PO-1234 </PO_NUM>
<CUST_ID> CUST001 </CUST_ID>
<ITEM_NUM> X9876 </ITEM_NUM>
<QUANTITY> 5 </QUANTITY>
<PRICE> 14.98 </PRICE>
</PURCHASE_ORDER>
```

Benefits of XML

- Open W3C standard
- Representation of data across heterogeneous environments
  - Cross platform
  - Allows for high degree of interoperability
- Strict rules
  - Syntax
  - Structure
  - Case sensitive

XML Building blocks

- PI (Processing Instruction)
- Tags
- Elements
- Content
- Attributes
- Entities
- Comments

Tags

- *Tags* are used to specify a name for a given piece of information.
- A tag consists of opening and closing angular brackets (<>) that enclose the name of the tag.
- Example

<EMP_NAME>Nick Shaw</EMP_NAME>

Elements

- Elements are represented using tags.
- An XML document must always have a *root element*.
- General format:

<element> … </element>

- Empty element:

<empty-Element />

- Example

<Authorname>Aruna Lakshmanan</Authorname>

Elements

- XML Elements are Extensible

  XML documents can be extended to carry more information

- XML Elements have Relationships

  Elements are related as parents and children

- Elements have Content

  Elements can have different content types: element content, mixed content, simple content, or empty content and attributes

- XML elements must follow the naming rules

Content

- Content refers to the information represented by the elements of an XML document.
  - Character or data content
  - Element content
  - Combination or mixed content
- Example

<BOOKNAME>Aruna History</BOOKNAME>

Syntax

<element-name attribute1 attribute2> ....content </element-name>

- **eleent-nae** is the name of the element. The *name*its case in the start and end tags must match.
- **attribute1, attribute2** are attributes of the element separated by white spaces. An attribute defines a property of the element. It associates a name with a value, which is a string of characters. An attribute is written as:
- **name = "value“**
- ***name*** is followed by an = sign and a string *value* inside double(" ") or single(' ') quotes.

Empty Element

- An empty element (element with no content) has following syntax:
- <name attribute1 attribute2.../>
- Example of an XML document using various XML element:
- **<?xml version="1.0"?>**
- **<contact-info>**
- **<address category="residence">**
- **<name>Tanmay Patil</name>**
- **<company>TutorialsPoint</company>**
- **<phone>(011) 123-4567</phone>**
- **<address/>**
- **</contact-info>**

XML Elements Rules

- Following rules are required to be followed for XML elements:
- An element *name* can contain any alphanumeric characters. The only punctuation mark allowed in names are the hyphen (-), under-score (_) and period (.).
- Names are case sensitive. For example, Address, address, and ADDRESS are different names.
- Start and end tags of an element must be identical.
- An element, which is a container, can contain text or elements as seen in the above example.

XML - Attributes

- Attributes are part of the XML elements. An element can have multiple unique attributes. Attribute gives more information about XML elements. To be more precise, they define properties of elements. An XML attribute is always a *name-value* pair.

- **Syntax**

An XML attribute has following syntax:

```
<element-name attribute1 attribute2 >
 ....content..
< /element-name>
```

Attribute Types

- StringType
- TokenizedType
- EnumeratedType

Element Attribute Rules

- Following are the rules that need to be followed for attributes:
- An attribute name must not appear more than once in the same start-tag or empty-element tag.
- An attribute must be declared in the Document Type Definition (DTD) using an Attribute-List Declaration.
- Attribute values must not contain direct or indirect entity references to external entities.
- The replacement text of any entity referred to directly or indirectly in an attribute value must not contain either less than sign <

XML comment

Syntax:

<!-------Your comment----->

A comment starts with <!-- and ends with -->. You can add textual notes as comments between the characters. You must not nest one comment inside the other.

XML Comments Rules

- Comments cannot appear before XML declaration.
- Comments may appear anywhere in a document.
- Comments must not appear within attribute values.
- Comments cannot be nested inside the other comments.

Entities

Entities are the placeholders in XML. These can be declared in the document prolog or in a DTD. There are different types of entities.

Character Entities can be used to display those symbols/special characters also.

Types of Character Entities

- Predefined Character Entities
- Numbered Character Entities
- Named Character Entities

Predefined Character Entities

- Ampersand: &
- Single quote: '
- Greater than: >
- Less than: <
- Double quote: "

Numeric Character Entities

The numeric reference is used to refer to a character entity. Numeric reference can either be in decimal or hexadecimal format.

As there are thousands of numeric references available, these are a bit hard to remember. Numeric reference refers to the character by its number in the Unicode character set.

Named Character Entity

As its hard to remember the numeric characters, the most preferred type of character entity is the named character entity.

For example:

'Aacute' represents capital character with acute accent.

Displaying XML

- XML documents do not carry information about how to display the data

- We can add display information to XML with
  - CSS (Cascading Style Sheets)
  - XSL (eXtensible Stylesheet Language) --- preferred

Components of an XML Document

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="template.xsl"?>
<ROOT>
   <ELEMENT1><SUBELEMENT1 /><SUBELEMENT2 /></ELEMENT1>
   <ELEMENT2> </ELEMENT2>
   <ELEMENT3 type='string'> </ELEMENT3>
   <ELEMENT4 type='integer' value='9.3'> </ELEMENT4>
</ROOT>
```

Elements with Attributes

Elements

Prologue (processing instructions)

Element vs. Tag vs. Attribute

- **Element** consists of *start tag, optional content* and an *end tag:*
  - <name>Introduction to XML</name>
- **Start tag**
  - <name>
- **Content**
  - Introduction to XML
- **End tag**
  - </name>
- **Start tag** may have **attribute**
  - <slide **number="1"**>

Advantages of XML

- It is **text-based**.
- It **supports Unicode**, allowing almost any information in any written human language to be communicated.
- It can represent common computer science data structures: records, lists and trees
- XML is heavily used as a format for **document storage** and processing, both online and offline.
- It is based on **international standards**.
- The **hierarchical** structure is suitable for most (but not all) types of documents.
- It manifests as **plain text** files, which are less restrictive than other spesific/proprietary document formats.
- It is **platform-independent**, thus relatively immune to changes in technology.

XML Application1—Separate data

XML can Separate Data from HTML

- Store data in separate XML files
- Using HTML for layout and display
- Using Data Islands
- Data Islands can be bound to HTML elements

Benefits:

Changes in the underlying data will not require any changes to your HTML

XML Application2—Exchange data

XML is used to Exchange Data

- Text format
- Software-independent, hardware-independent
- Exchange data between incompatible systems, given that they agree on the same tag definition.
- Can be read by many different types of applications

Benefits:

- Reduce the complexity of interpreting data
- Easier to expand and upgrade a system

XML Application3—Store Data

XML can be used to Store Data

- Plain text file
- Store data in files or databases
- Application can be written to store and retrieve information from the store
- Other clients and applications can access your XML files as data sources

Benefits:

Accessible to more applications

XML Application4—Create new language

XML can be used to Create new Languages

- WML (Wireless Markup Language) used to markup Internet applications for handheld devices like mobile phones (WAP)
- MusicXML used to publishing musical scores
- RSS.
- MathML.

XML support in IE 5.0+

Internet Explorer 5.0 has the following XML support:

- Viewing of XML documents
- Full support for W3C DTD standards
- XML embedded in HTML as Data Islands
- Binding XML data to HTML elements
- Transforming and displaying XML with XSL
- Displaying XML with CSS
- Access to the XML DOM (Document Object Model)

*Netscape 6.0 also have full XML support

Java APIs for XML

- JAXP: Java API for XML Processing
- JAXB: Java Architecture for XML Binding
- JDOM: Java DOM
- DOM4J: an alternative to JDOM
- JAXM: Java API for XML Messaging (asynchronous)
- JAX-RPC: Java API for XML-based Remote Process Communications (synchronous)
- JAXR: Java API for XML Registries

* * * * * * * * * * * *

Thank you