II YEAR – IV SEMESTER
COURSE CODE: 4BIT4C1
CORE COURSE VII – JAVA PROGRAMMING
UNIT 1,2,3,4

JAVA HISTORY

Java is an object-oriented programming language developed by James Gosling and colleagues at Sun Microsystems in the early 1990s.(James Gosling, Mike Sheridan, and Patrick Naughton)

AK - DEPT.OF.INFORMATION TECHNOLOGY- APSAC

Java was started as a project called "Oak" by James Gosling in June 1991. Gosling's goals were to implement a virtual machine and a language that had a familiar C-like notation but with greater uniformity and simplicity than C/C++. The first public implementation was Java 1.0 in 1995. It made the promise of "**Write Once, Run Anywhere**", with free runtimes on popular platforms

There were five primary goals in the creation of the Java language:

1. It should use the object-oriented programming methodology.

2. It should allow the same program to be executed on multiple operating systems.

3. It should contain built-in support for using computer networks.

4. It should be designed to execute code from remote sources securely.

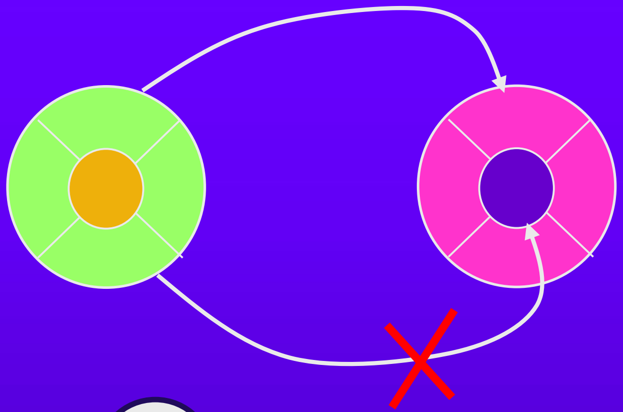
5. It should be easy to use by selecting what was considered the good parts of other object-oriented languages.

Characteristics of Java

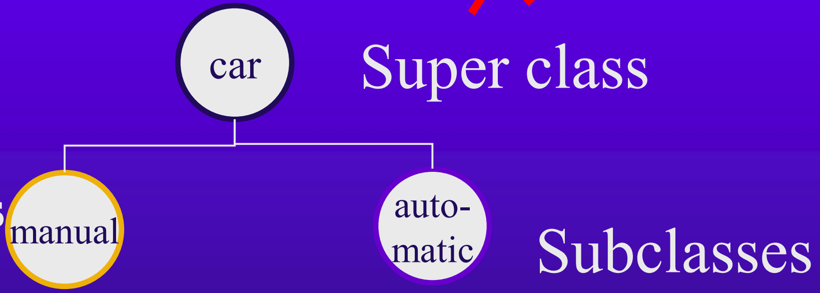
- **Java is simple**
- **Java is object-oriented**
- **Java is distributed**
- **Java is interpreted**
- **Java is robust [**memory management (opaque references, automatic garbage collection)]
- **Java is secure**
- **Java is architecture-neutral**
- **Java is portable [** WORA - Write Once, Run Anywhere].
- **Java's performance**
- **Java is multithreaded** (multiple simultaneous tasks).
- **Java is dynamic**

The three principles of OOP

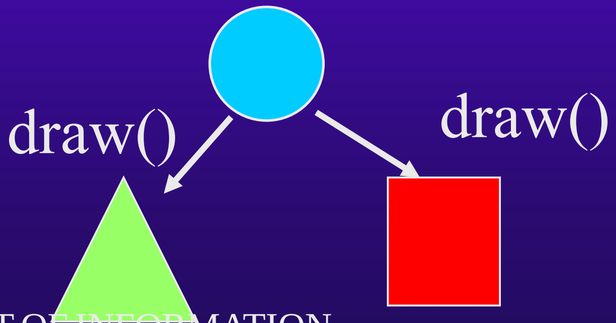
- Encapsulation
  - Objects hide their functions (**methods**) and data (**instance variables**)



- Inheritance
  - Each **subclass** inherits all variables of its **superclass**



- Polymorphism
  - Interface same despite different data types



JDK Versions

- JDK 1.02 (1995)
- JDK 1.1 (1996)
- Java 2 SDK v 1.2 (a.k.a JDK 1.2, 1998)
- Java 2 SDK v 1.3 (a.k.a JDK 1.3, 2000)
- Java 2 SDK v 1.4 (a.k.a JDK 1.4, 2002)

Java Development Kit

- **javac - The Java Compiler**
- **java - The Java Interpreter**
- **jdb - The Java Debugger**
- **appletviewer -Tool to run the applets**
- javap - to print the Java bytecodes
- javaprof - Java profiler
- javadoc - documentation generator
- javah - creates C header files

JDK Editions

- Java Standard Edition (J2SE)
  - J2SE can be used to develop client-side standalone applications or applets.
- Java Enterprise Edition (J2EE)
  - J2EE can be used to develop server-side applications such as Java servlets and Java ServerPages.
- Java Micro Edition (J2ME).
  - J2ME can be used to develop applications for mobile devices such as cell phones.

Java IDE Tools

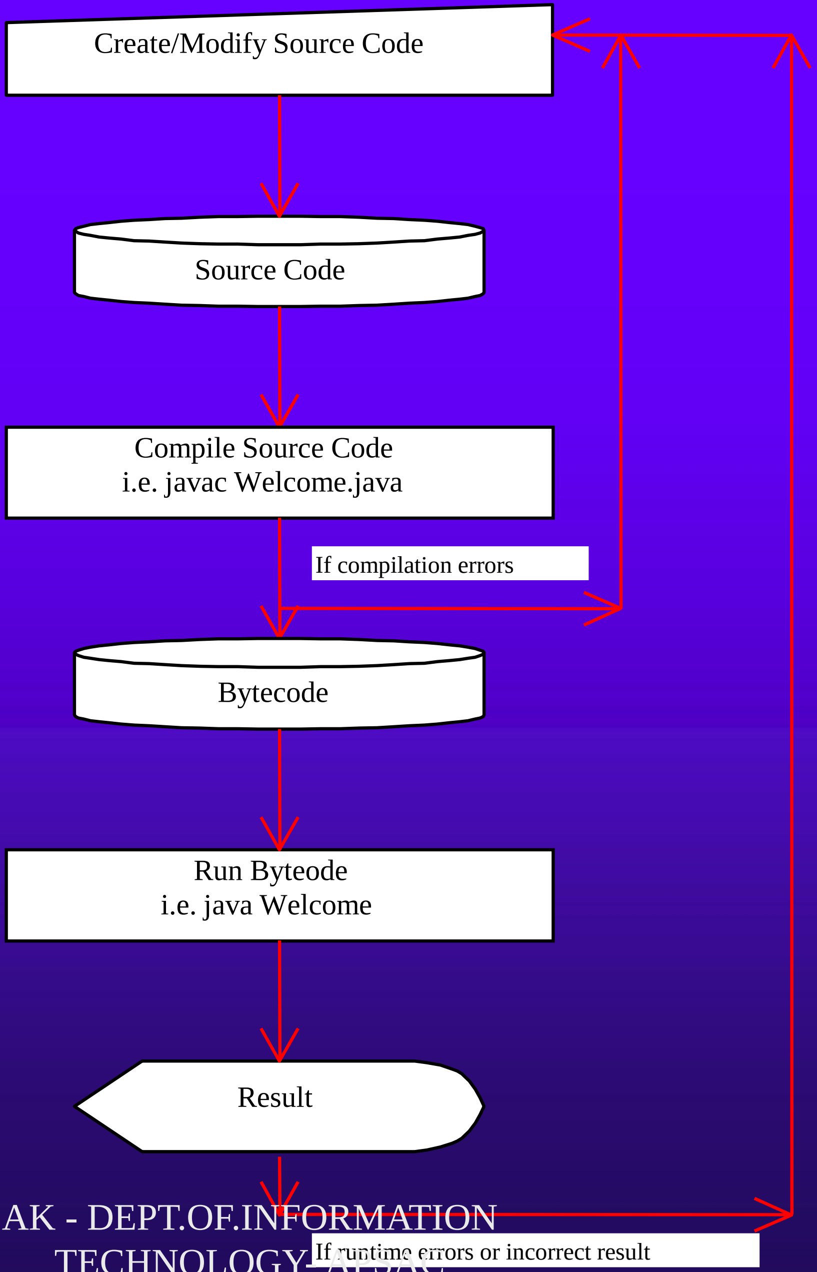
- Forte by Sun MicroSystems
- Borland JBuilder
- Microsoft Visual J++
- WebGain Café
- IBM Visual Age for Java

```
class Welcome
{
  public static void main(String args[])
{
 System.out.println("Welcome to Java!");
  }
}
```

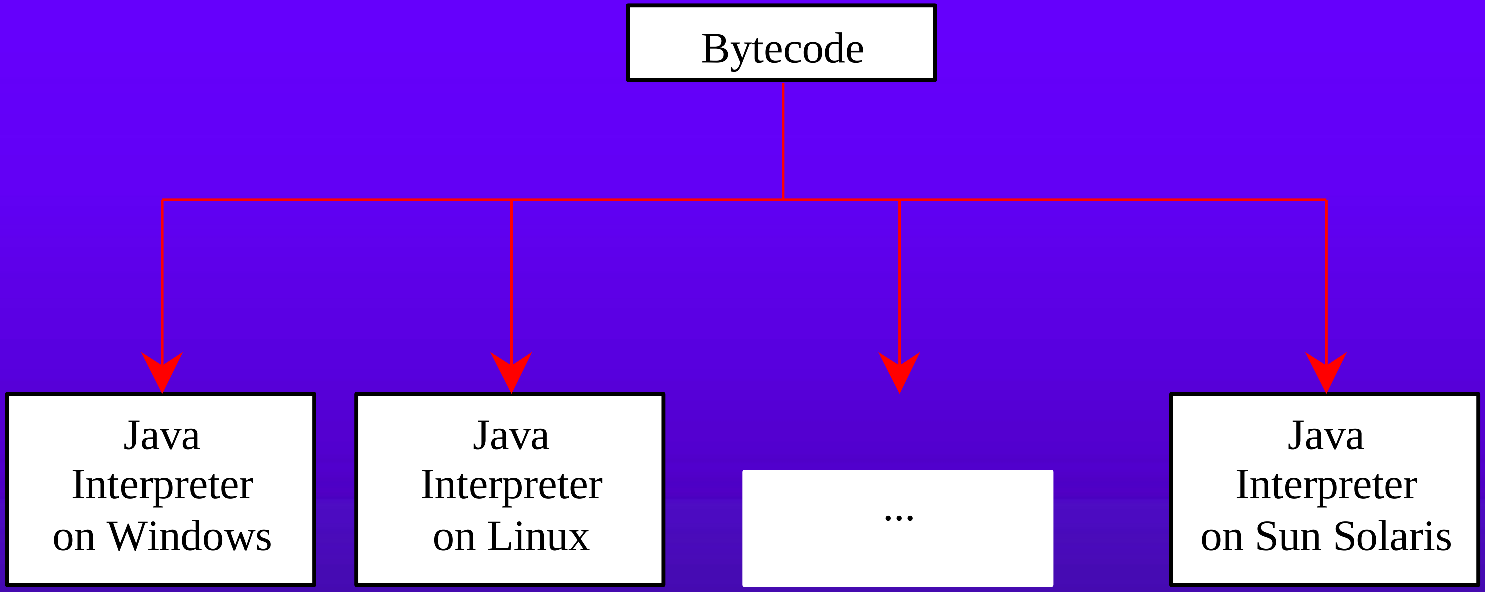
Compiling Programs

- On command line
  ```
  -javac file.java
  ```
- On command line
  ```
  -java classname
  ```

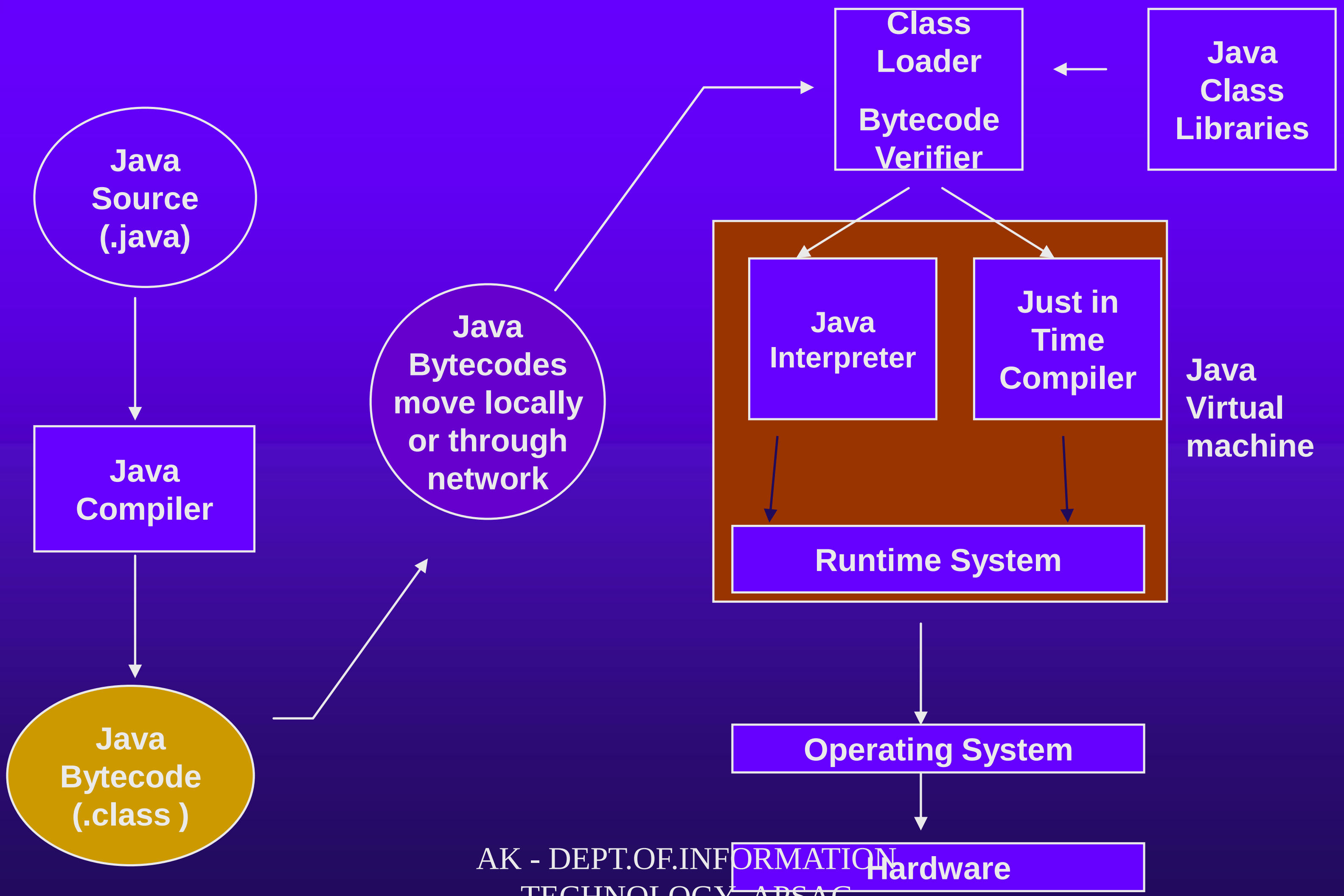
Compiling Programs



Executing Applications


Bytecode
Java Interpreter on Windows
Java Interpreter on Linux
...
Java Interpreter on Sun Solaris

How it works…!


Java Source (.java)
Java Compiler
Java Bytecode (.class)
Java Bytecodes move locally or through network
Class Loader
Bytecode Verifier
Java Class Libraries
Java Interpreter
Just in Time Compiler
Runtime System
Java Virtual machine
Operating System
Hardware

Anatomy of a Java Program

- Comments
- Package
- Reserved words
- Modifiers
- Statements
- Blocks
- Classes
- Methods
- The main method

Java Comments

The Java programming language supports three kinds of comments:

/* text */

The compiler ignores everything from /* to */.

/** documentation */

This indicates a documentation comment (doc comment, for short). The compiler ignores this kind of comment, just like it ignores comments that use /* and */. The JDK javadoc tool uses doc comments when preparing automatically generated documentation.

// text

The compiler ignores everything from // to the end of the line.

Reserved words or keywords

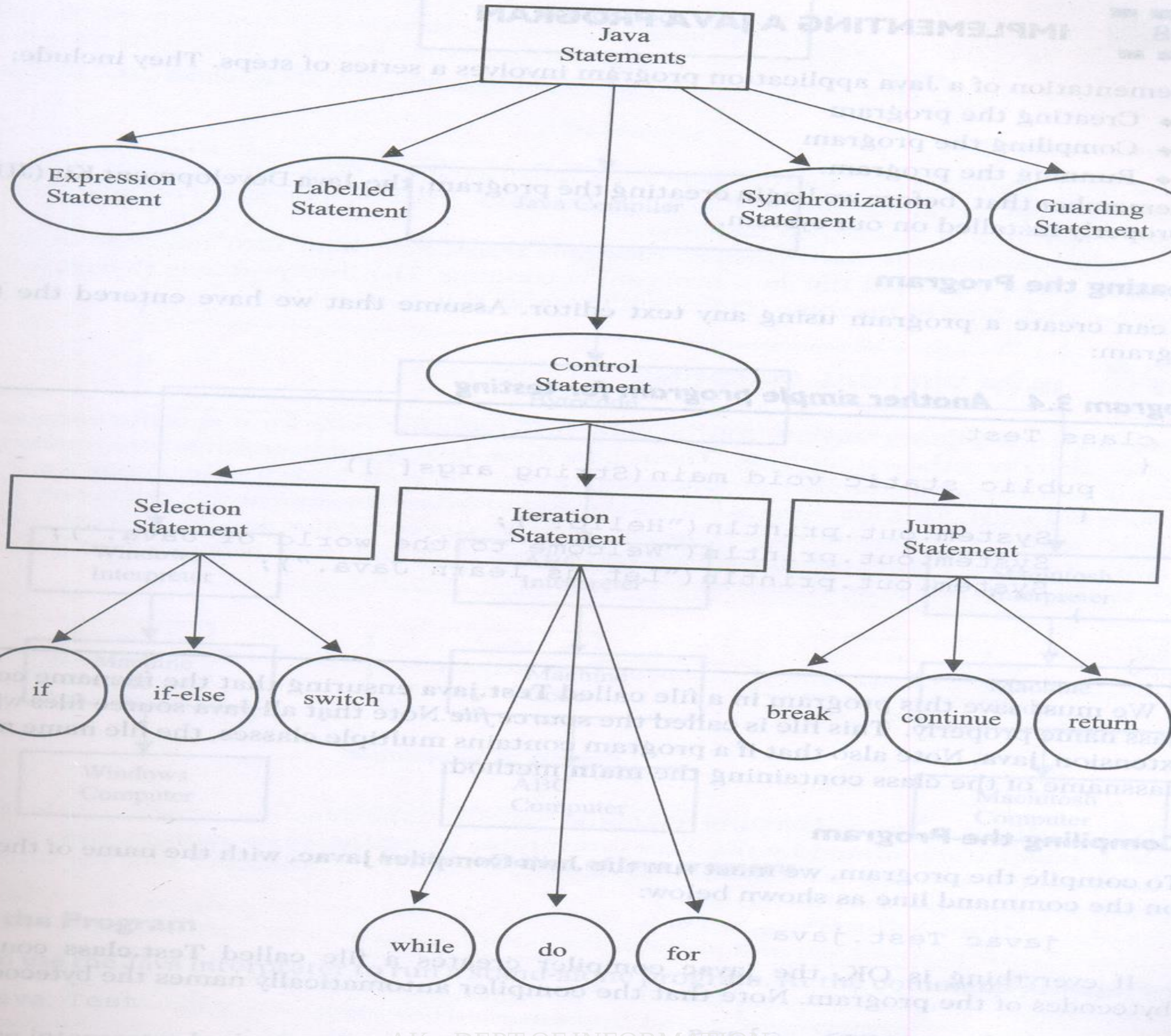
Reserved words or keywords are words that have a specific meaning to the compiler and cannot be used for other purposes in the program. For example, when the compiler sees the word class, it understands that the word after class is the name for the class. Other reserved words are if, for, int, float, public, static, and void.

Modifiers

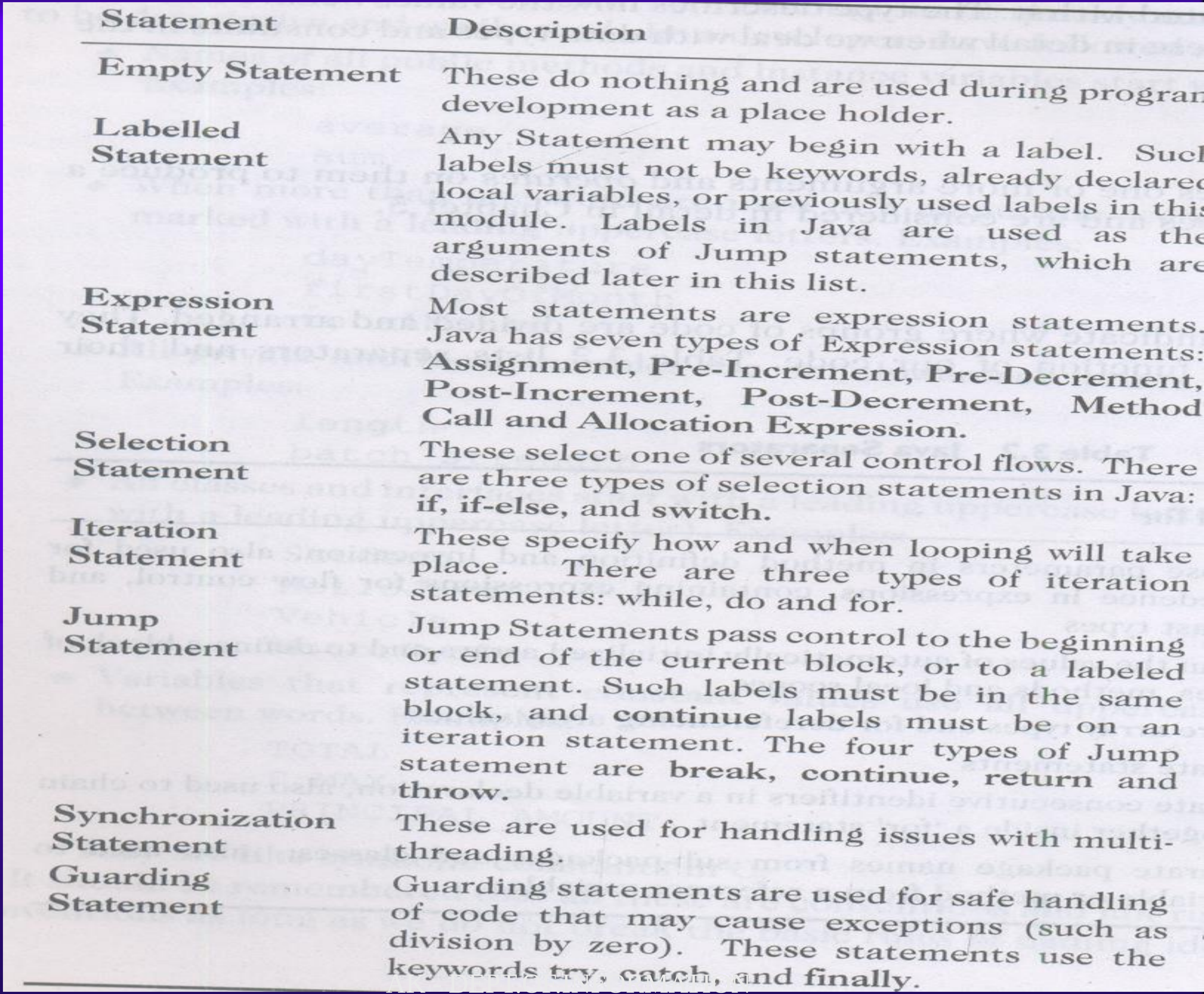
Java uses certain reserved words called *modifiers* that specify the properties of the data, methods, and classes and how they can be used. Examples of modifiers are <u>public</u> and <u>static</u>. Other modifiers are <u>private</u>, <u>final</u>, <u>abstract</u>, and <u>protected</u>. A <u>public</u> datum, method, or class can be accessed by other programs. A <u>private</u> datum or method cannot be accessed by other programs.

Statements

A *statement* represents an action or a sequence of actions. The statement System.out.println**("Welcome to Java")** is a statement to display the greeting "Welcome to Java!‟. Every statement in Java ends with a semicolon (;).



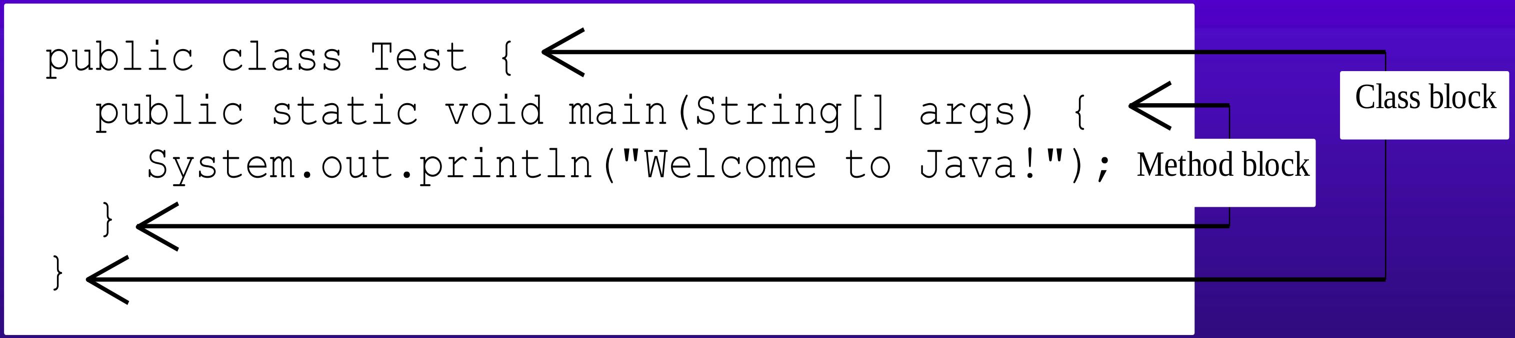
Classification of Java statements



Statement	Description
Empty Statement	These do nothing and are used during progran development as a place holder.
Labelled Statement	Any Statement may begin with a label. Suc labels must not be keywords, already declare local variables, or previously used labels in thi module. Labels in Java are used as the arguments of Jump statements, which are described later in this list.
Expression Statement	Most statements are expression statements. Java has seven types of Expression statements: **Assignment, Pre-Increment, Pre-Decrement, Post-Increment, Post-Decrement, Method Call and Allocation Expression.**
Selection Statement	These select one of several control flows. There are three types of selection statements in Java: **if, if-else, and switch.**
Iteration Statement	These specify how and when looping will take place. There are three types of iteration statements: while, do and for.
Jump Statement	Jump Statements pass control to the beginning or end of the current block, or to a labeled statement. Such labels must be in the same block, and **continue** labels must be on an iteration statement. The four types of Jump statement are **break, continue, return** and **throw.**
Synchronization Statement	These are used for handling issues with multi-threading.
Guarding Statement	Guarding statements are used for safe handling of code that may cause exceptions (such as division by zero). These statements use the keywords **try, catch, and finally.**

Blocks

A pair of braces in a program forms a block that groups components of a program.



```
public class Test {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

Classes

The *class* is the essential Java construct. A class is a template or blueprint for objects. The program is defined by using one or more classes.

Methods

- A method is a named sequence of code that can be invoked by other Java code.
- A method takes some parameters, performs some computations and then optionally returns a value (or object).
- your program may call the same method many times
- methods can return a value

```
public static int addNums(int num1, int num2)
{
        int answer = num1 + num2;
        return answer;
}
```

main Method

The main method provides the control of program flow. The Java interpreter executes the application by invoking the main method.

The main method looks like this:

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

Java Data and Variables

Integer Primitive Data Types

Type	**Size**	**Range**
byte	8 bits	-128 to +127
short	16 bits	-32,768 to +32,767
int	32 bits	(about)-2 billion to +2 billion
long	64 bits	(about)-10E18 to +10E18

Floating Point Primitive Data Types

Type	Size	Range
float	32 bits	-3.4E+38 to +3.4E+38
double	64 bits	-1.7E+308 to 1.7E+308

Char , String

Boolean

For each primitive type, there is a corresponding *wrapper class*. A wrapper class can be used to convert a primitive data value into an object, and some type of objects into primitive data. The table shows primitive types and their wrapper classes:

primitive type	**Wrapper type**
byte	Byte
short	Short
int	Int
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

Variables only exist within the structure in which they are defined. For example, if a variable is created within a method, it cannot be accessed outside the method. In addition, a different method can create a variable of the same name which will not conflict with the other variable. A java variable can be thought of as a little box made up of one or more bytes that can hold a value of a particular data type:

```
class example
                                                  {
  public static void main ( String[] args )
 {
 long x = 123;    //a declaration of a variable named x with a datatype of long
System.out.println("The variable x has: " + x );

                                                  }
}
```

TYPE CONVERSIONS

General Format:

```
(type-name)expression;
```

Examples	Action
x=(int)7.5	7.5 is converted to integer by truncation
a=(int)21.3/(int)4.5	Evaluated as 21/4 & result would be 5
b=(double)sum/n	Division is done in floating point mode
y=(int)a+b	a is converted to integer & added to b
z=(int)(a+b)	the result of a+b is converted to int.
p=cost((double)x)	converts x to double before using it as parameter.

Note :

1.float to int causes truncation of the fractional part.

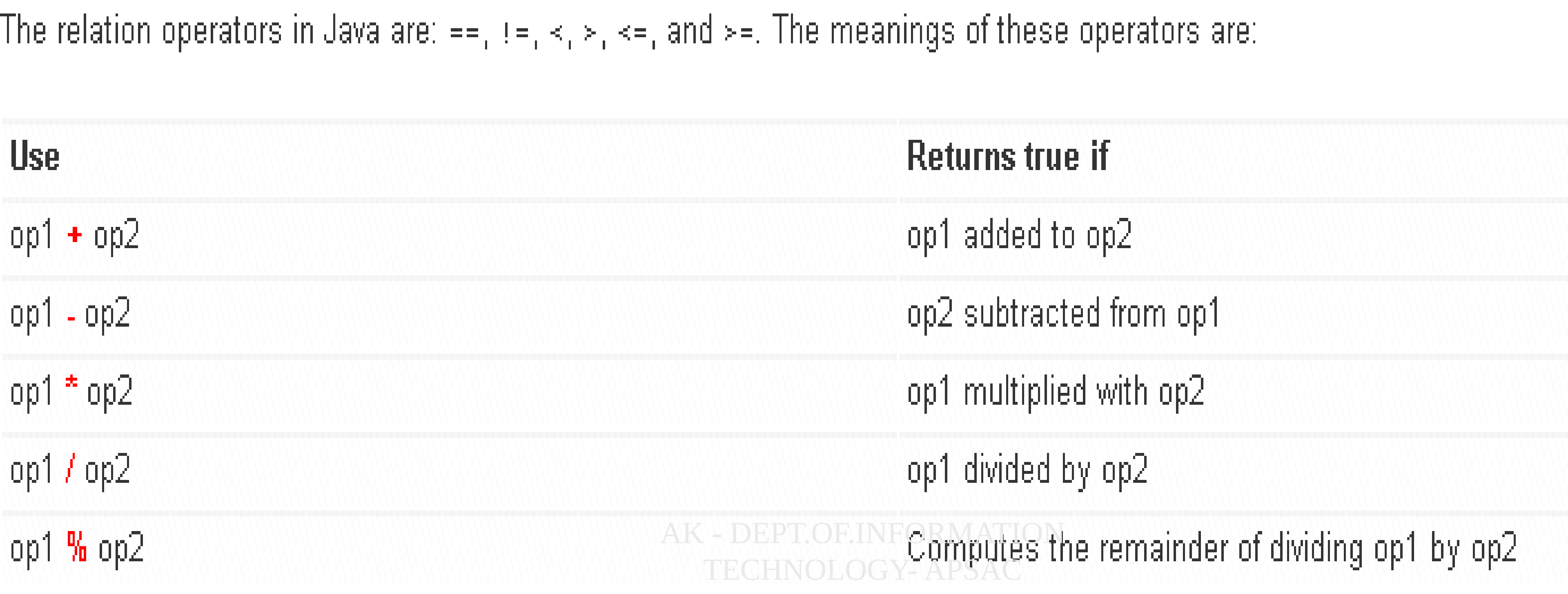
2.double to float causes rounding of digits.

3.long to int causes dropping of the excess higher order bits.

Java Arithmetic Operators

The Java programming language has includes five simple arithmetic operators like are + **(addition), - (subtraction),** *** (multiplication), / (division)**, and **% (modulo).**

The following table summarizes the binary arithmetic operators in the Java programming language.



The relation operators in Java are: ==, !=, <, >, <=, and >=. The meanings of these operators are:

Use	Returns true if
op1 + op2	op1 added to op2
op1 - op2	op2 subtracted from op1
op1 * op2	op1 multiplied with op2
op1 / op2	op1 divided by op2
op1 % op2	Computes the remainder of dividing op1 by op2

Java Assignment Operators

The assignment operator is evaluated from **right to left**, so a = b = c = 0; would assign 0 to c, then c to b then b to a.

i = i + 2;

Here we say that we are assigning **i's** value to the new value which is i+2.

A shortcut way to write assignments like this is to use the += operator. It's one operator symbol so don't put blanks between the + and =.
i += 2; // Same as "i = i + 2“

```
//assign the literal
//"Hello" to str
String str = new String("Hello");

//assign b to a, then assign a
//to d; results in d, a, and b being equal
int d = a = b;
```

Java Increment and Decrement Operators

There are 2 Increment or decrement operators -> ++ and --. These two operators are unique in that they can be written both before the operand they are applied to, called prefix increment /decrement, or after, called postfix increment /decrement. The meaning is different in each case.

Example

```
x = 1;
y = ++x;
System.out.println(y);
```

prints 2, but

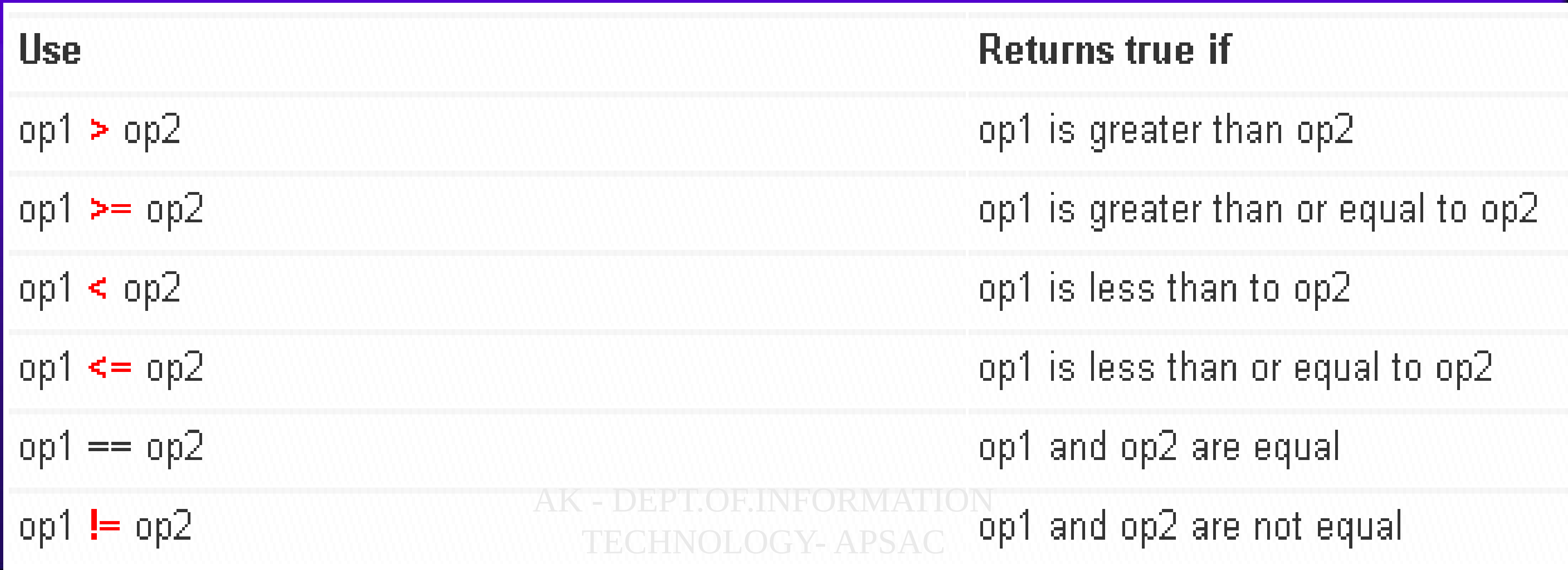
```
x = 1;
y = x++;
System.out.println(y);
```

prints 1

Java Relational Operators

A relational operator compares two values and determines the relationship between them. For example, != returns true if its two operands are unequal. Relational operators are used to test whether two values are equal, whether one value is greater than another, and so forth.

The relation operators in Java are: ==, !=, <, >, <=, and>=. The meanings of these operators are:



Use	Returns true if
op1 > op2	op1 is greater than op2
op1 >= op2	op1 is greater than or equal to op2
op1 < op2	op1 is less than to op2
op1 <= op2	op1 is less than or equal to op2
op1 == op2	op1 and op2 are equal
op1 != op2	op1 and op2 are not equal

Java Conditional Operators

The JVM tests the value of **Boolean-expression**. If the value is true, it evaluates **expression-1**; otherwise, it evaluates **expression-2**. For

Example

```
if (a > b)
{
    max = a;
}
else
{
    max = b;
}
```

Logical Operator

&& --→AND

|| --→ OR

! --→ NOT

Setting a single variable to one of two states based on a single condition is such a common use of if-else that a shortcut has been devised for it, the conditional operator, ?:. Using the conditional operator you can rewrite the above example in a single line like this:

```
max = (a > b) ? a : b;
```

Java If-Else Statement

The if-else class of statements should have the following form:

Simple IF

```
if (condition)
{
statements;
}
```

IF….ELSE

```
if (condition)
 {
statements;
} else
{
statements;
}
```

ELSE IF LADDER

```
if (condition)
{
statements;
}
else if (condition)
{
statements;
} else
 {
statements;
}
```

Operator Precedence in Java

```
( )
*,/
+,-
System.out.println("1 + 2 = " + 1 + 2);
System.out.println("1 + 2 = " + (1 + 2));
------------------------------------------
System.out.println(1 + 2 + "abc");
System.out.println("abc" + 1 + 2);
```

Example

```
class demo
{
 public static void main(String[] args)
{
 int count = 6+2*5-8/2;
System.out.println("Count is: " + count);
}
}
```

Operator	Description	Level	Associativity
[] . () ++ --	access array element access object member invoke a method post-increment post-decrement	1	left to right
++ -- + - ! ~	pre-increment pre-decrement unary plus unary minus logical NOT bitwise NOT	2	right to left
() new	cast object creation	3	right to left
* / %	multiplicative	4	left to right
+ - +	additive string concatenation	5	left to right
<< >> >>>	shift	6	left to right
< <= > >= instanceof	relational type comparison	7	left to right

== !=	equality	8	left to right
&	bitwise AND	9	left to right
^	bitwise XOR	10	left to right
|	bitwise OR	11	left to right
&&	conditional AND	12	left to right
||	conditional OR	13	left to right
?:	conditional	14	right to left
= += -= *= /= %= &= ^= |= <<= >>= >>>=	assignment	15	right to left

Looping Statement

The for Statement

The for statement provides a compact way to iterate over a range of values. Programmers often refer to it as the "for loop" because of the way in which it repeatedly loops until a particular condition is satisfied. The general form of the for statement can be expressed as follows:

```
for (initialization; termination; increment)
{
   statement(s)
}
```

When using this version of the for statement, keep in mind that:

•The **initialization expression** initializes the loop; it's executed once, as the loop begins.

•When the **termination expression** evaluates to false, the loop terminates.

•The **increment expression** is invoked after each iteration through the loop; it is perfectly acceptable for this expression to increment or decrement a value.

```
class ForDemo
 {
public static void main(String args [])
{
for(int i=1; i<11; i++)
{
System.out.println("Count is: " + i);
 }
}
}
```

The output of this program is:

```
Count is: 1
Count is: 2
Count is: 3
Count is: 4
Count is: 5
Count is: 6
Count is: 7
Count is: 8
Count is: 9
 Count is: 10
```

While Loop

The while statement continually executes a block of statements while a particular condition is true.
Its syntax can be expressed as:

```
while (expression)
 {
    statement(s)
}
```

The while statement evaluates expression, which must return a boolean value. If the expression evaluates to true, the while statement executes the statement(s) in the while block.

The while statement continues testing the expression and executing its block until the expression evaluates to false.

Using the while statement to print the values from 1 through 10 can be accomplished as in the following WhileDemo program:

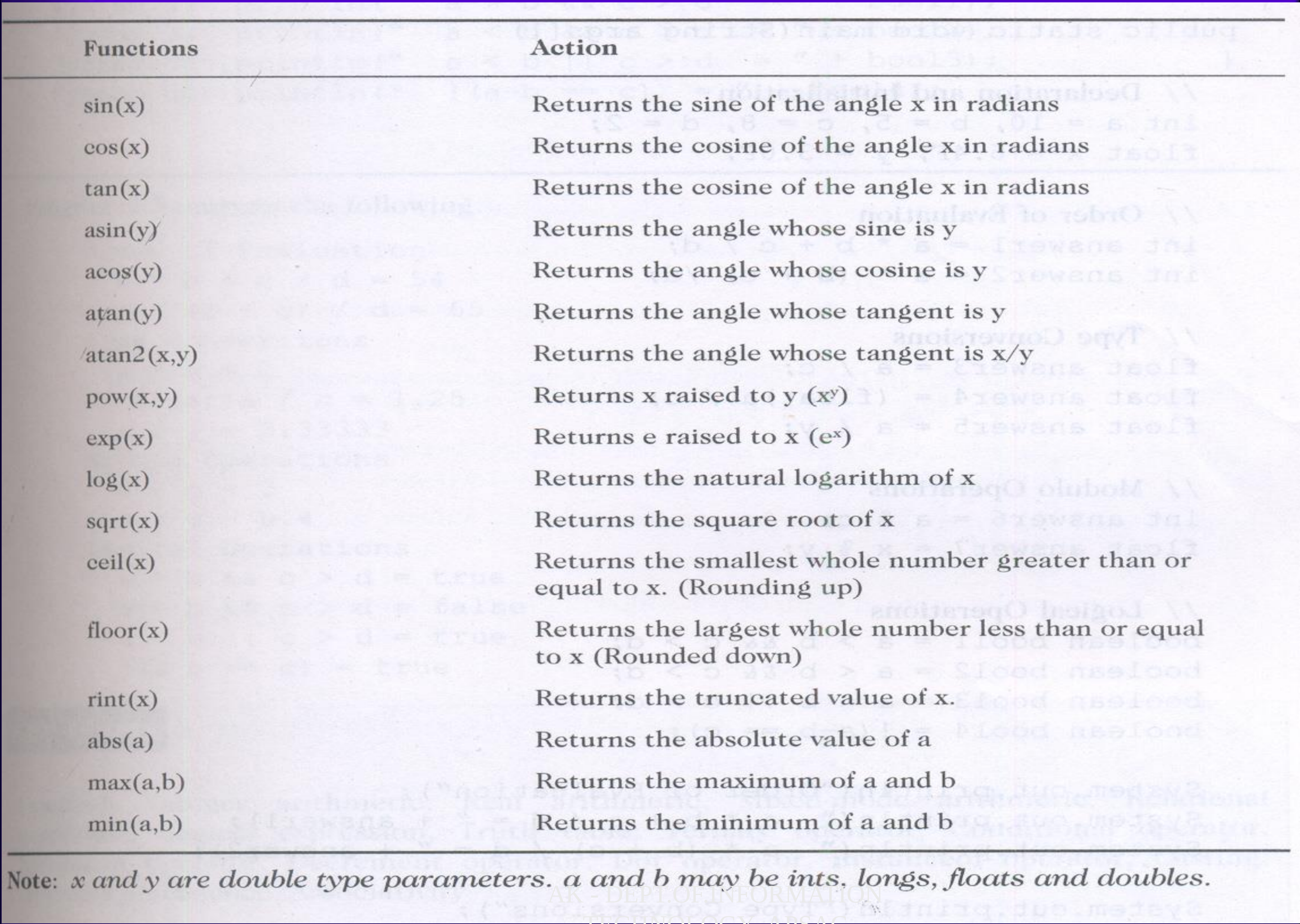
```
class whiledemo
{

public static void main(String[] args)

{
          int count = 1;
          while (count < 11)
{

System.out.println("Count is: "+ count);
 count++;

}
  }
}
```

The Java programming language also provides a **do-while statement**, which can be expressed as follows:

```
do {
    statement(s)
} while (expression);
```

The difference between do-while and while is that do-while evaluates its expression at the bottom of the loop instead of the top. Therefore, the statements within the do block are always executed at least once, as shown in the following DoWhileDemo program:

```
class DoWhileDemo {
    public static void main(String[] args)
{
        int count = 1;
        do {
            System.out.println("Count is: " + count);
            count++;
        } while (count <= 11);
    }
}
```



Functions	Action
sin(x)	Returns the sine of the angle x in radians
cos(x)	Returns the cosine of the angle x in radians
tan(x)	Returns the cosine of the angle x in radians
asin(y)	Returns the angle whose sine is y
acos(y)	Returns the angle whose cosine is y
atan(y)	Returns the angle whose tangent is y
atan2(x,y)	Returns the angle whose tangent is x/y
pow(x,y)	Returns x raised to y (x^y)
exp(x)	Returns e raised to x (e^x)
log(x)	Returns the natural logarithm of x
sqrt(x)	Returns the square root of x
ceil(x)	Returns the smallest whole number greater than or equal to x. (Rounding up)
floor(x)	Returns the largest whole number less than or equal to x (Rounded down)
rint(x)	Returns the truncated value of x.
abs(a)	Returns the absolute value of a
max(a,b)	Returns the maximum of a and b
min(a,b)	Returns the minimum of a and b

Note: *x and y are double type parameters. a and b may be ints, longs, floats and doubles.*

ARRAYS

An Array is a group of contiguous or related data items that share a common name.

Types of Array
1.One Dimensional Array
2.Two Dimensional Array

One Dimensional Array
A list of items can be given one variable name using only one subscript and such a variable is called a single subscripted variable or one-dimensional array.

For example
```
int number[ ]= new int[5];
```

The computer reserves five storage location. Like that table.

Number[0]
Number[1]
Number[2]
Number[3]
Number[4]

The values of the array elements can be assigned as follows
number[0]=35
number[1]=40
number[2]=30
number[3]=25
number[4]=50

This would cause the array number to store the values like that table

number[0]	35
number[1]	40
number[2]	30
number[3]	25
number[4]	50

Creating an Array

Like any other variables,arrays must be declared and created in the computer memory before they are used .

1.Declare the Array
2.Create memory locations
3.Put values into the memory location.

Declaration of Array.
Arrays in java may be declared in two forms:
Form1
```
            datatype arrayname[ ];
```
Form2
```
            datatype [ ] arrayname;
```
Examples:
```
int number[ ];
float average[ ];
int [ ] num;
```

Initialization of Arrays

arrayname[subscript]=value;

Example

Number[0]=35;

Type arrayname[ ]={list of values};

int number [ ] = { 35 , 40 ,20 ,57 ,19};

Its possible to assign an array object to another

int a[ ] = { 1 , 2 , 3}

int b[ ];

b=a;

Array Length:

int k=array.length;

TWO – DIMENSIONAL ARRAYS

The table shows the value of sales of three items by four salesgirls.

	Item1	Item2	Item3
Salesgirl#1	310	275	365
Salesgirl#2	210	190	325
Salesgirl#3	405	235	240
Salesgirl#4	260	300	380

The table contains a total of 12 values, three in each line.We can think of this table as a matrix consistng of four rows and three columns. Each row represents the values of sales by a particular salesgirl and each column represents the values of sales of a particular item.

In mathematics , we represent a particular value in a matrix by using two subscripts such as Vij. V denotes Entire Matrix.

Vij refers to the value in the ith row and jth column

For example V23 refers to the value 325.

In java its can be represented as V[4][3].

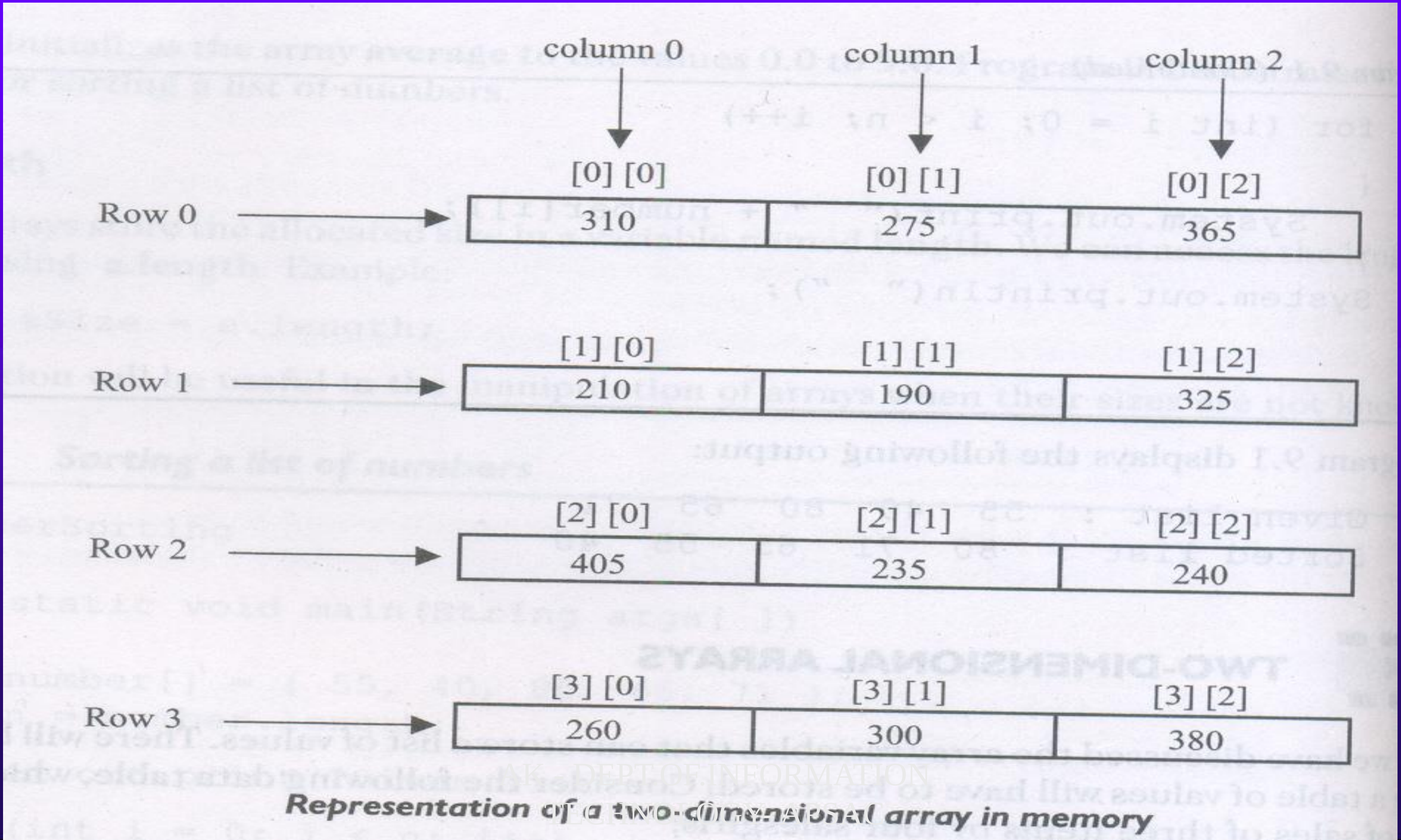
In two dimensional arrays the first index selects the row and the second index selects the column within that row.

Declaration

```
int  myArray[ ] [ ];
myArray=new int[3][4];
```

Or

```
int  myArray[ ][ ]= new int[3][4];
```



Representation of a two-dimensional array in memory

Two dimensional arrays may be initialized by following their declaration with a list of initial values enclosed in braces.

```
int  table[2][3]={0,0,0,1,1,1}
```

Initializes the elements of the first row to zero and the second row to one. The initialization is done row by row.The above statement can be equivalently written as

```
int table[ ] [ ]={{0,0,0},{1,1,1}};
```

By surrounding the elements of each row by braces.

We can also initializes a two-dimensional array in the form of a matrix….

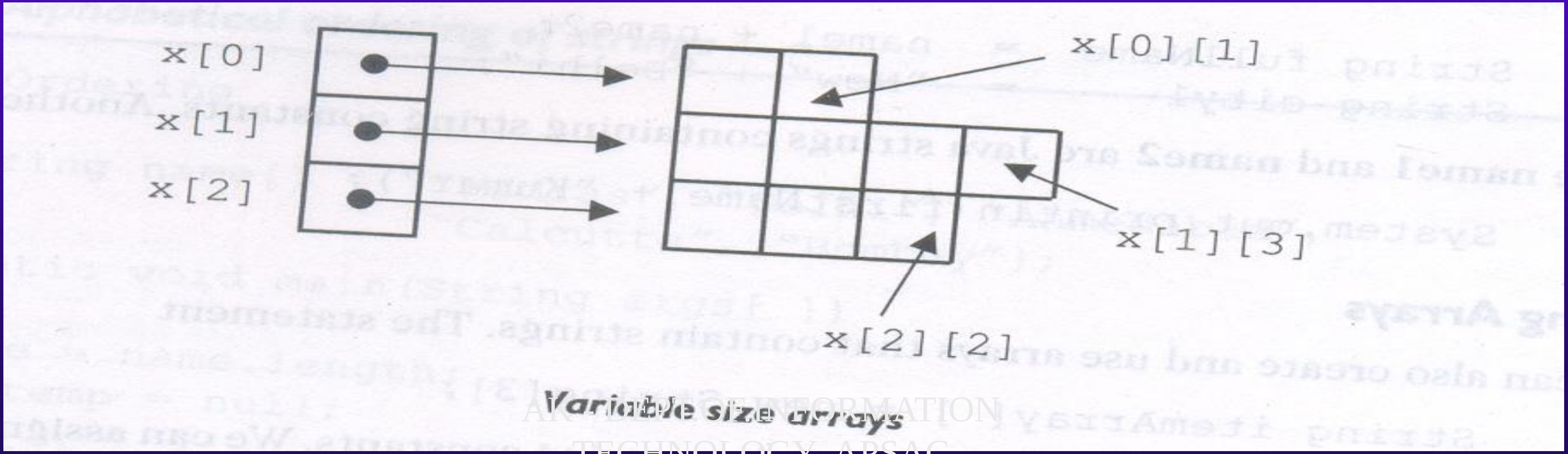
```
int table[ ] [ ]={
                     {0,0,0},
                     {1,1,1}
                    };
```

Variable size of Arrays

Java treats multidimensional arrays as “arrays of arrays ”.It is possible to declare a two-dimensional array as follows:

```
int x[ ][ ]=new int[3][ ];
x[0]=new int[2];
x[1]=new int[4];
x[2]=new int[3];
```

These statements create a two-dimensional array as having different lengths for each row .



Variable size arrays

STRINGS

String represent a sequence of characters.

```
char temp=new char[4];
temp[0]='j';
temp[1]='a';
temp[2]='v';
temp[3]='a';
```

In Java , Strings are class objects and implemented using two classes namely **String** and **StringBuffer**.

Java Strings as compared to C strings are more reliable and predictable. A Java string is not a character array and is not NULL terminated.

```
       String name;
      name=new String("Apsac");
          or
    String name=new String("Apsac");
```

To get the length of String using the **length** method

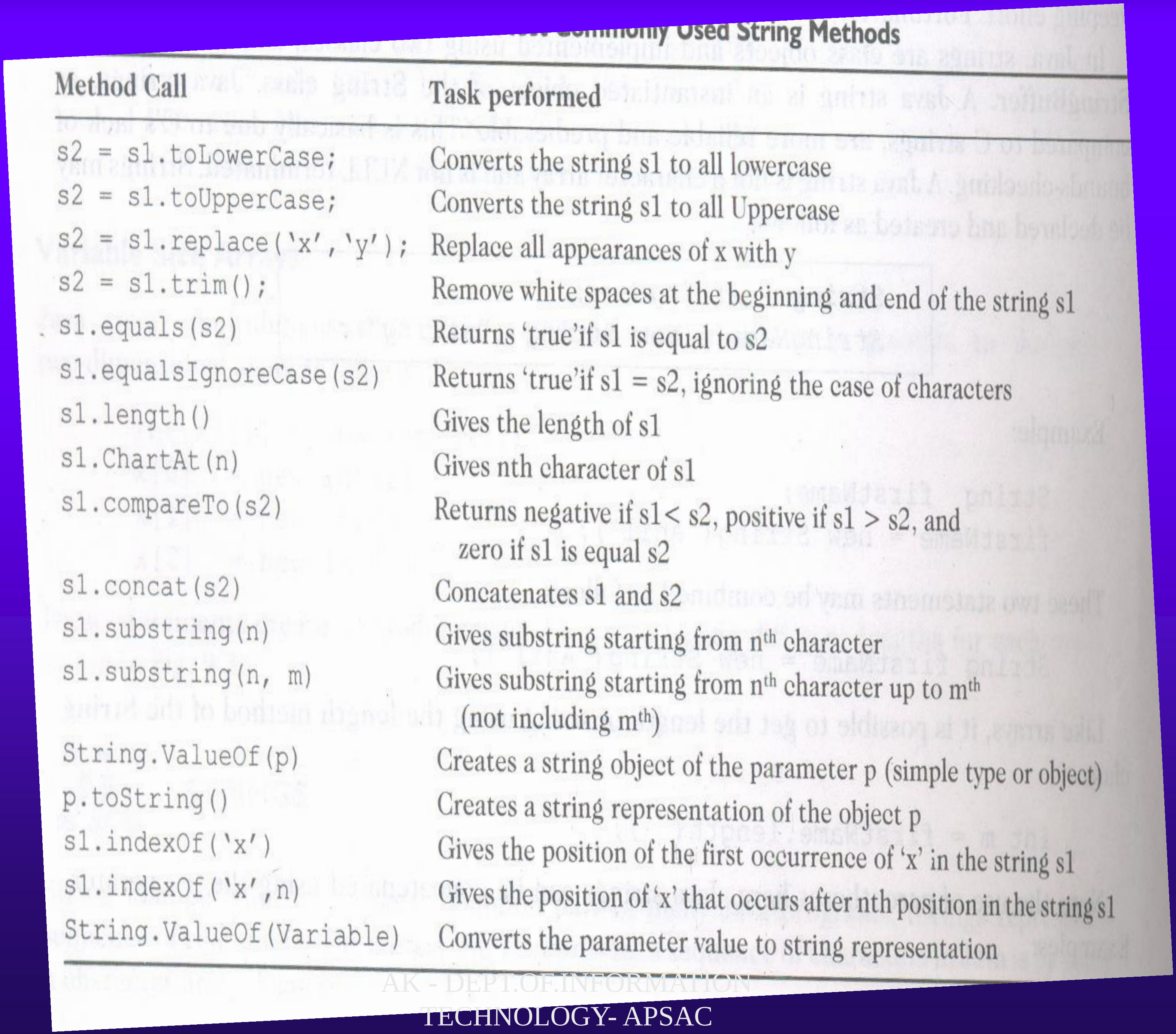
```
int m=name.length( );
```

```
String fullname=firstname+lastname;
String Fullname="apsa"+"college";
```

String Arrays

String item=new String[3];

Its create item array of size of 3 to hold three string statements.



ommonly Used String Methods

Method Call	Task performed
`s2 = s1.toLowerCase;`	Converts the string s1 to all lowercase
`s2 = s1.toUpperCase;`	Converts the string s1 to all Uppercase
`s2 = s1.replace('x', 'y');`	Replace all appearances of x with y
`s2 = s1.trim();`	Remove white spaces at the beginning and end of the string s1
`s1.equals(s2)`	Returns 'true' if s1 is equal to s2
`s1.equalsIgnoreCase(s2)`	Returns 'true' if s1 = s2, ignoring the case of characters
`s1.length()`	Gives the length of s1
`s1.ChartAt(n)`	Gives nth character of s1
`s1.compareTo(s2)`	Returns negative if s1< s2, positive if s1 > s2, and zero if s1 is equal s2
`s1.concat(s2)`	Concatenates s1 and s2
`s1.substring(n)`	Gives substring starting from nth character
`s1.substring(n, m)`	Gives substring starting from nth character up to mth (not including mth)
`String.ValueOf(p)`	Creates a string object of the parameter p (simple type or object)
`p.toString()`	Creates a string representation of the object p
`s1.indexOf('x')`	Gives the position of the first occurrence of 'x' in the string s1
`s1.indexOf('x',n)`	Gives the position of 'x' that occurs after nth position in the string s1
`String.ValueOf(Variable)`	Converts the parameter value to string representation

String Buffer

StringBuffer s=New StringBuffer();

Method	Task
s1.setCharAt(n,’x’)	Modifies the nth character to x
s1.append(s2)	Appends the string s2 to s1 at the end
S1.insert(n,s2)	insert the string s2 at the position n of the string s1
S1.setLength(n)	Sets the length of the string s1 to n.

Vector

Arrays can be easily implemented as Vectors.that can hold objects of any type and any number.The objects do not have to be homogenous.

Vector v=new Vector(); declaring without size

Vector v=new Vector(5); declaring with size

Methods

v.addElement(item) – Adds the item to the list at the end

v.elementAt(10) -

v.size()

v.removeElement(item)

v.removeElementAt(n)

v.removeElements()

v.copyInto(array)

v.insertElementAt(item,n)

General structure of a Java Program

Documentation Section	Suggested
Package Statement	Optional
Import Statements	Optional
Interface Statement	Optional
Class Definitions	Optional

Main Method Class
{
Main Method Definition
}
	Essential

Chapter 8 : Classes , Objects and Methods

Classes create objects and objects use methods to communicate between them.

In Java , the data items are called fields and the functions are called methods.

Defining a CLASS

```
class classname [extends superclassname]
{
        [variable declaration; ]
        [method declaration; ]
}
```

Everything inside the square brackets is optional.

classname and superclassname are any valid Java Identifiers .The keyword extends indicates that the properties of the superclassname class are extended to the classname class. This concept known as inheritance.

Adding Variables

Data is encapsulated in class by placing data fields inside the body of the class definition. These variables are called instance variables because they are created whenever an object of the class is instantiated.

```
class rectangle
{
int length;
int width;
}
```

The class rectangle contains two integer type instance variables. It is allowed to declare them in one line as

```
int length , width;
```

These variables are only declared and therefore no storage space has been created in the memory. Instance variables are also known as member variables.

Adding Methods

Methods are declared inside the body of the class but immediately after the declaration of instance variables.

```
type methodname  (parameter-list)
{
method-body;
}
```

Method declaration have four parts

- The name of the method (method name)
- The type of value the method returns(type)
- A list of parameters(parameter-list)
- The body of the method

```
class Rectangle
{
int length,width;
void getData(int x,int y)
{
length=x;
width=y;
}
int rectArea( )
{
int area=length*width;
return(area);
}
}
```

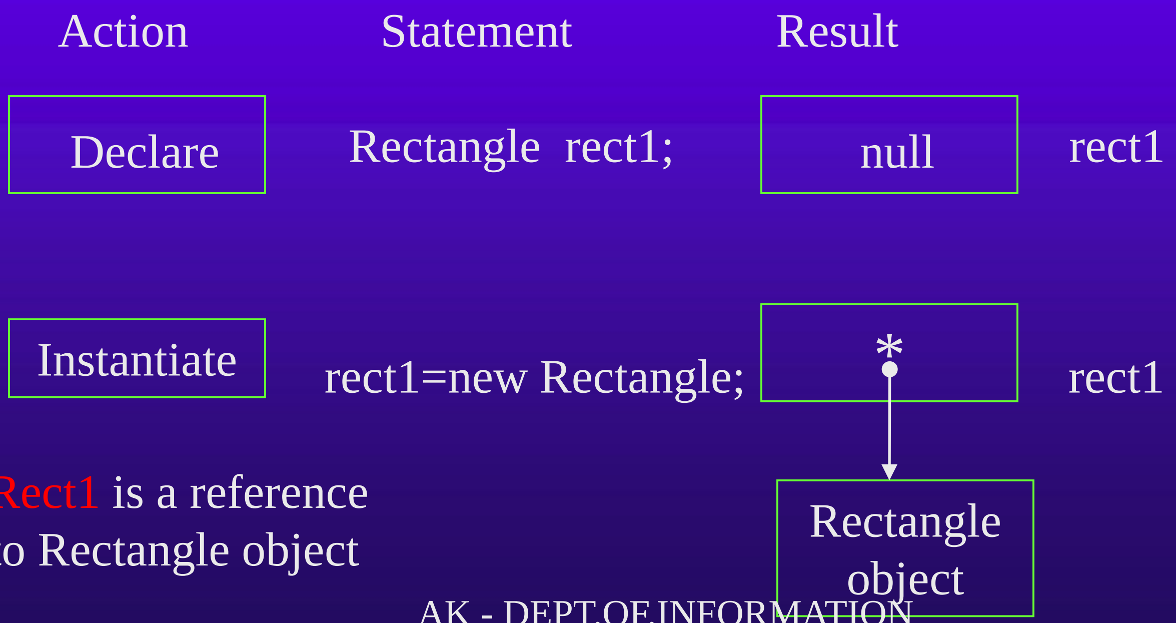
```
class Access
{
int  x;
void method1( )
{
int y;
x=10; // legal
y=x;   // legal
}

void method2( )
{
int z;
x=5;    // legal
z=10; // legal
y=1;   // illegal
}
}
```

Creating Objects

An object in Java is essentially a block of memory that contains space to store all the instance variables . Creating an object is also referred to as instantiating an object.

Objects in Java are created using the new operator. The new operator creates an object of the specified class and returns a reference to that object.

```
Rectangle rect1;              // declare
rect1=new Rectangle( ) ; // instantiate
```



Rect1 is a reference to Rectangle object

Constructors

- Constructors enables an object to initialize itself when it is created.
- Constructors have the same name as the class itself.
- They do not specify a return type , not even void.

```
class Rectangle
{
int length,width;
Rectangle(int x , int y)
{
length=x;
width=y;
}
int rectArea( )
{
int area=length*width;
return(area);
}
}
```

```
class areafind
{
Public static void main(String args[ ] )
{
Rectangle rect1=new Rectangle(15,10); // Calling Constructor
int  a=rect1.rectArea( );
System.out.println(" Area : "+ a);
}
}
```

METHODS OVERLOADING

Create methods that have the same name , but different parameter lists and different definitions is called **Method Overloading.**

When we call a method in an object , Java matches up the method name first and then the number and type of parameters to decide which one of the definitions to execute .
This process is known as **polymorphism**.

STATIC MEMBERS

A Class basically contains two sections . One declares variables and the other declares methods . These variables and methods are called instance variables and instance methods . This is because every time the class is instantiated , a new copy of each of them is created. They are accessed using the objects(with dot operator).

Let us assume that we want to define a member that is common to all the objects and accessed without using a particular object. That is ,the member belongs to the class as a whole rather than the objects created from the class.

static int count ;
static int max(int x,int y);

The members that are declared static as shown above are called static members.

Restrictions:
1.They can only call other static methods.
2.They can only access static data.
They cannot refer to this or super in any way.

NESTING OF METHODS

A method can be called by using only its name by another method of the same class is known as nesting of methods.

INHERITANCE : EXTENDING A CLASS

Reusability is yet another aspect of OOP paradigm. It is always nice if we could reuse something that already exists rather than creating the same all over again.Java classes can be reused in several ways.

This is basically done by creating new classes,reusing the properties of existing ones.The mechanism of deriving a new class from an old one is called Inheritance.

The old class is known as the **base class** or **super class** or **parent class.**

The new one is called the **sub class** or **derived class** or **child class**.

The inheritance allows subclasses to inherit all the variables and methods of their parent classes.

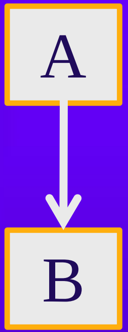
Defining a Subclass

```
class subclassname extends superclassname
{
        variable declaration;
        methods declaration;
}
```

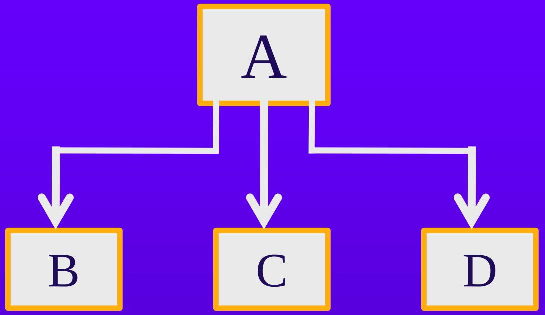
The keyword extends signifies that the properties of the **superclassname** are extended to the **subclassname**.The subclass will now contain its own variables and methods as well those of the superclass.This kind of situation occurs when we want to add some more properties to an existing class without actually modifying it.

Inheritance may take different forms:

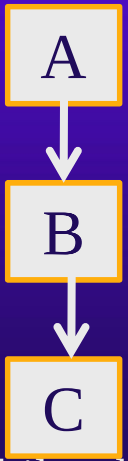
Single Inheritance (only one super class)
Multiple inheritance (several super classes) // java does not directly implement Multiple Inheritance
Hierarchical inheritance (one super class, many subclasses)
Multilevel Inheritance (Derived from a derived class)



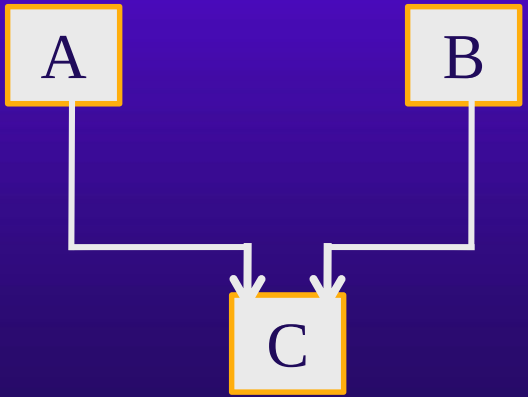
(a) Single inheritance



(b) Hierarchical inheritance



(c) Multilevel inheritance



(d) Multiple inheritance

Subclass Constructor

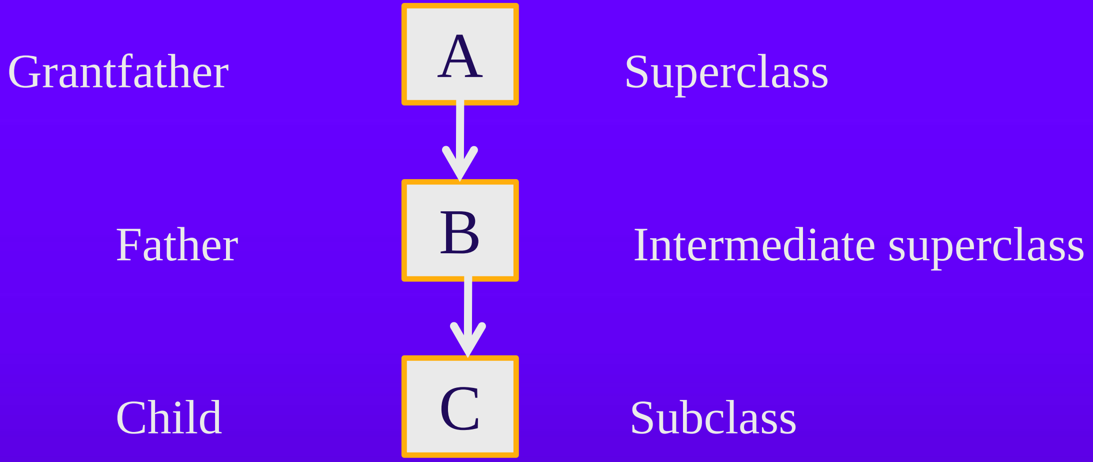
A subclass constructor is used to construct the instance variables of both the subclass and the superclass.The subclass constructor uses the keyword super to invoke the constructor method of the superclass.

Conditions :

- ❖Super may only be used within a subclass constructor method.
- ❖The call to superclass constructor must appear as the first statement within the subclass constructor
- ❖The parameters in the super call must match the order and type of the instance variable declared in the superclass.

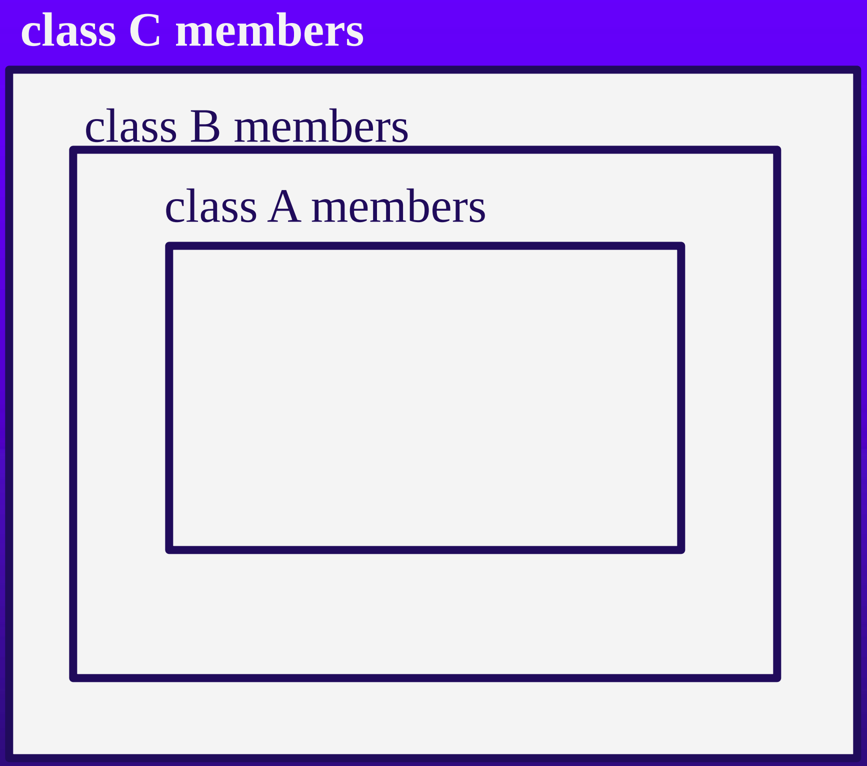
Multilevel Inheritance

A common requirement in object-oriented programming is the use of a derived class as a super class.



```
class A
{
……
……..
}
class  B extends A          // First Level
{
……
……..
}
class  B extends A         // Second Level
{
……
……..
}
```

The class A serves as a base class for the derived class B which in turn serves as a base class for the derived class C.The chain ABC is known as **inheritance path**.

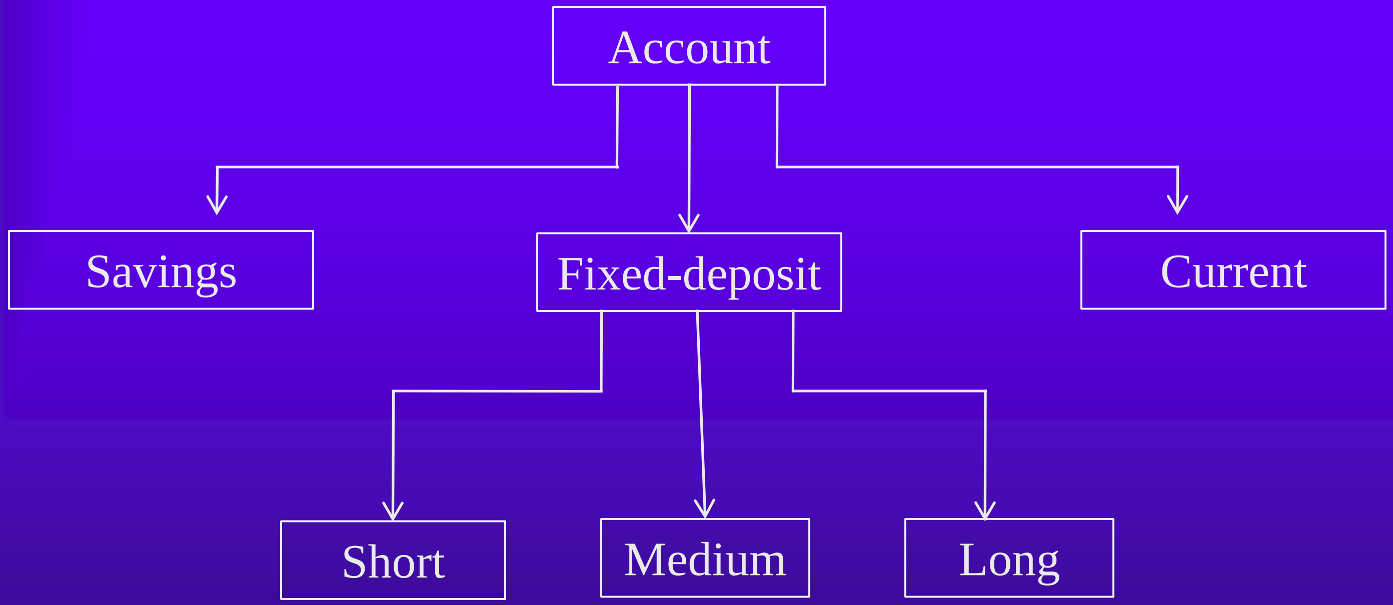


C contains B with contains A

Hierarchical Inheritance

Many programming problems can be cast into a hierarchy where certain features of one level are share by many others below the level.

The example shows a hierarchical classification of accounts in a commercial bank. This is possible because all the accounts posses certain common features.



Hierarchical classification of bank accounts

Overriding Methods

Its possible by defining a method in the subclass that has the same name , same arguments and same return type as a method in the super class.When the method is called ,the method definition in the subclass is invoked and executed instead of the one in the superclass .This is known as overriding.

Final Variables and Methods

All methods and variables can be overridden by default in subclasses. If we wish to prevent the subclasses from overriding the members of the superclass , we declare them as final using the keyword **final** as a modifier.

Example

```
final int size=100;
final void show( )
{ ….
…….}
```

Making a method final ensures that the functionality defined in this method will never be altered in any way. The value of a final variable can never be changed.

Final Classes

Sometimes we may like to prevent a class being further subclassed for security reasons.A class that cannot be subclassed is called a final class.

```
final class abc{    ……………… }
final class bcd extends mark { ……….. }
```

Any attempt to inherit these classes will cause an error and the compiler will not allow it.

Finalizer Methods

A constructor method is used to initialize an object when its declared . This process is known as initialization . Similarly , Java supports a concept called finalization , which is just opposite to initialization.We know that Java run-time is an automatic garbage collecting system. It automatically frees up the memory resources used by the objects. But objects may hold other non-object resources such as file descriptors or window system fonts. The garbage collector cannot free these resources.In order to free these resources we must use a finalizer method. This is similar to destructors in c++.

The finalizer method is simply finalize( ) and can be added to any classes.Java calls that method whenever it is about to reclaim the space for that object.

ABSTRACT METHODS AND CLASSES

Java allows us to do something that is exactly opposite to final.That is , we can indicate that a method must always be redefined in a subclass,thus making overriding compulsory.

Example

```
abstract class shape
{
………
……..
abstract void draw( );
…………
}
```

Conditions:

We cannot use abstract classes to instantiate objects directly.

Example

```
        shape s = new  shape( ); // its illegal because shape is an abstract class.
```

The abstract methods of an abstract class must be defined in its subclass.

We cannot declare abstract constructors or abstract static methods.

Visibility Control

In some situations to restrict the access to certain variables and methods from outside the calss. we can achieve this in Java by applying visibility modifiers to the instance variables and methods.The visibility modifiers are also known as access modifiers.

Java provides three type of visibility modifiers: public , private and protected.

Public Access
Private Access
Protected Access
Friendly Access
Private protected Access

Managing Errors and Exceptions

Types of Errors

Errors may broadly be classified into two categories:

- Compile-time errors
- Run time errors

Compile –time errors

All syntax errors will be detected and displayed by the java compiler and therefore these errors are known as compile-time errors.Whenever the compiler displays an error, it will not create the .class file.

The most common problems are:

- Missing semicolons
- Missing (or miss match of) brackets in classes and methods
- Misspelling of identifiers and keywords
- Missing double quotes in strings
- Use of undeclared variables
- Incompatible types in assignments/initialization
- Bad references to objects
- Use of = in place of == operator

Run –Time Errors

Some times, a program may compile successfully creating the .class file but may not run properly.Such program produce wrong results due to wrong logic or may terminate due to errors such as stack overflow.

Most common run time errors

•Dividing an integer by zero.

•Accessing an element that is out of the bounds of an array

•Passing a parameter that is not in a valid range or value for a method

Exceptions

An exception is a condition that is caused by a run-time error in the program. When the java interpreter encounters an error such as dividing an integer by zero, it creates an exception object and throws it.

The mechanism suggests incorporation of a separate error handling code that performs the following tasks;

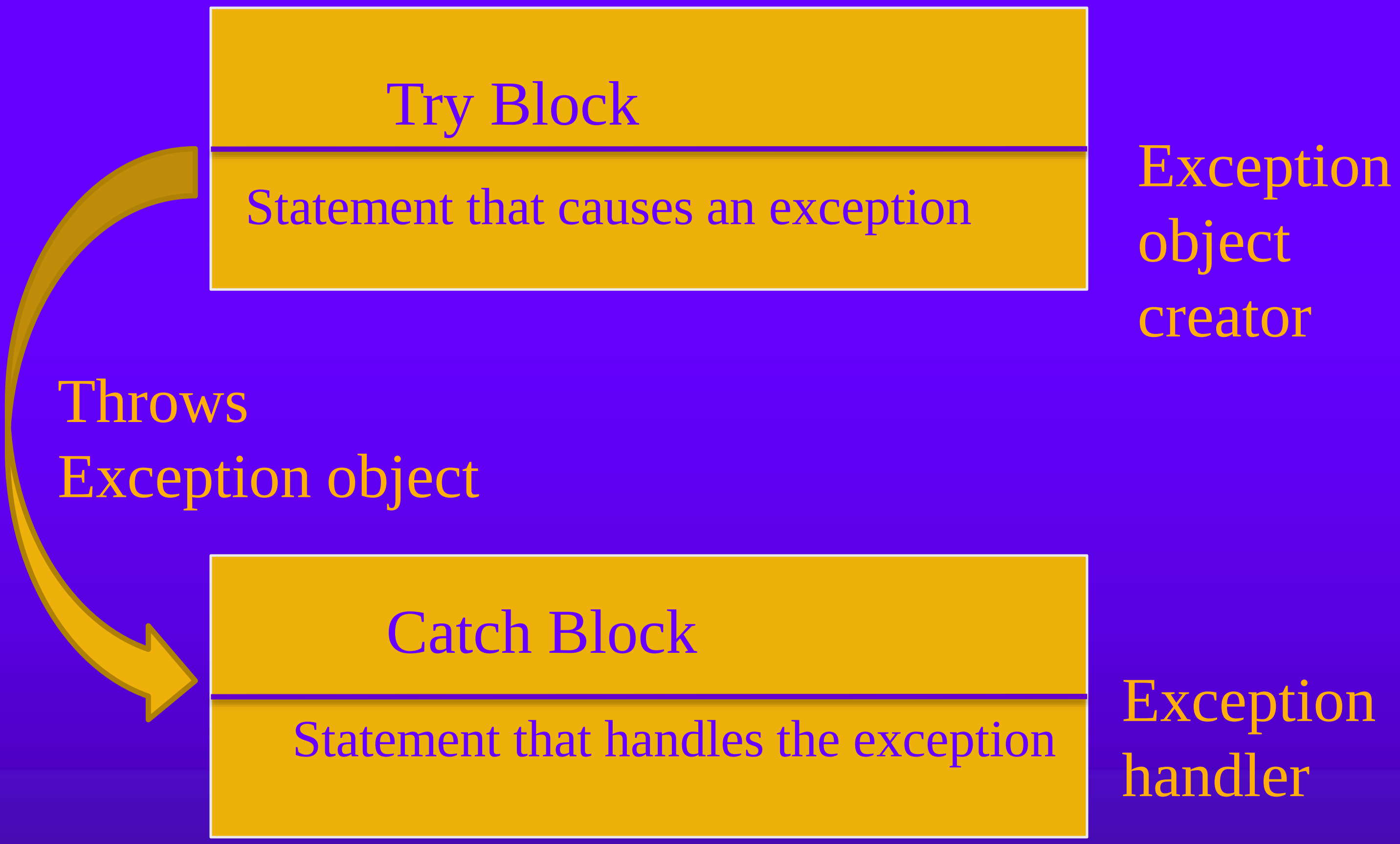
1.Find the problem(Hit the Exception)
2.Inform that an error has occurred(Throw the exception)
3.Receive the error information(Catch the exception)
4.Take corrective actions(Handle the exception)

Common java exceptions

Exception Type	Cause of exception
ArithmeticException	: Caused by math errors such as division by zero
ArrayIndexOutOfBoundsException	:Caused by bad array indexes
IOException	: Caused by general I/O failures.

Syntax of Exception Handling Code

```
try
{
Statement; // generates an exception
}
Catch (Exception-type  e)
{
Statement; // processes the exception
}
```


Try Block
Statement that causes an exception
Exception object creator
Throws Exception object
Catch Block
Statement that handles the exception
Exception handler

Interfaces : Multiple Inheritance

```
class A extends B extends C
{
……………..
……………..
}
```

An interface is basically a kind of class. interfaces contain methods and variables but with major diferrence.The difference is that interfaces define only abstract methods and final fields.This means that interfaces do not specify any code to implement these methods and data fields contain only constants.

```
iterface  interfacename
{
variable declaration;
methods declaration;
}
```

Declaration

```
static final type variablename=value;
Return-type methodname(parameter_list);
```

Example

```
interface Item
{
static final int code=1001;
static final string name=“Pencil”;
void display();
}
```

Extending Interface

```
interface name2 extends name1
{
body of name2
}
```

Implementing Interfaces

Interfaces are used as "superclasses" whose properties are inherited by classes.

```
class classname implements interface
{
Body of class name;
}
------------------
class classname extends superclass implements
interface1,interface2,….
{
Body of class name;
}
```