JAVA

Java Evolution

Java History

- Java is a general Purpose , Object oriented programming language.
- Developed by Sun Microsystem of USA in 1991.
- Initially called as Oak by James Gosling.

1990 – Sun Microsystem decided to develop special software for consumer electronic device.
1991 – James Gosling and his team announced a new language named “Oak”
1994 – The team developed a web browser called “Hot Java” to locate and run applet program on Internet.
1995 – Oak was renamed “java” due to some legal snags.
1996 – Sun releases Java Development Kit 1.0
1997 - Sun releases Java Development Kit 1.1
1998 – Java 2 with version 1.2 of the software Development Kit(SDK 1.2)
1999 – Standard Edition (J2SE) and Enterprise Edition (J2EE)
2000 – J2SE with SDK 1.3 released.
2002 – J2SE with SDK 1.4 released.
2004 – JDK 1.5 released

Java Features

- Simple
- Object-Oriented
- Platform independent & Portable
- Secured
- Robust
- Dynamic
- Interpreted
- High Performance
- Multithreaded
- Distributed

Features

- **Simple**

- It is a small and simple language
- Many Complex concepts has been removed from c & C++. E.g.., external pointers, multiple inheritance and operator overloading etc..
- If you know any object oriented programming language learning java is simple, if you know c++ java is even more simple

- **Object Oriented**

- Almost everything in java is Object.
- All program code and data reside within classes and objects.
- The object model in java is simple and easy to extend.
- almost all data types are objects (files, strings, etc.)
- potentially better code organization and reuse

Features

- **Platform Independent and Portable**

- The most important feature of java.
- Java program can be moved from one system to another, anywhere and anytime
- After compilation of java program we will get the .class file which is called byte code where user and machine cant understand it, it is also called intermediate code resides in JVM.
- This class file can be taken and execute it to in any machine and any operating system with out source code, this is why it is called platform independent.
- In simple, java programs can be executed on any processors like Pentium, Celeron, Dual core and so on. Hence it is called **Architecture Neutral.**

Features

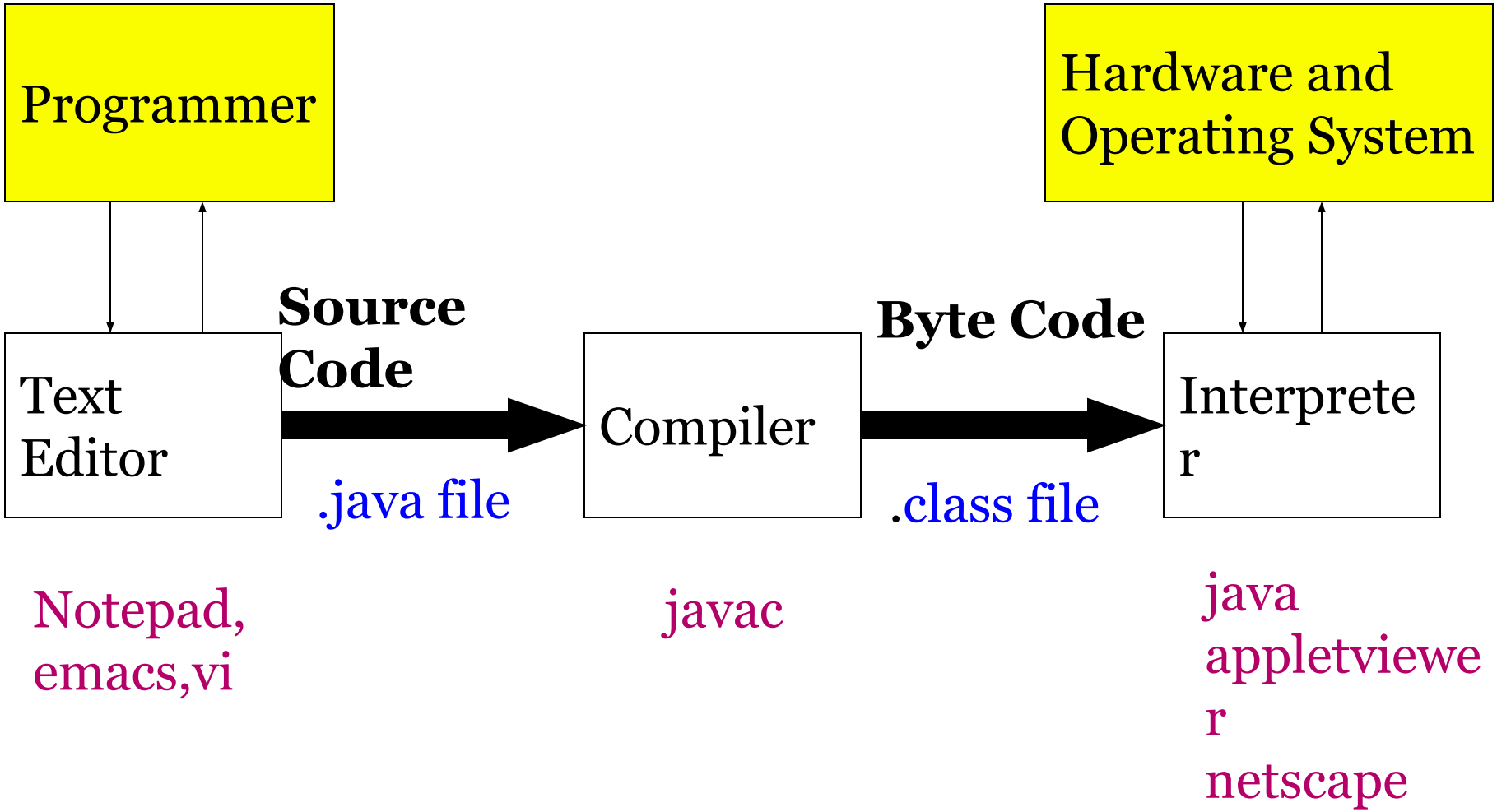
- **Secure**
  - Security is the important issues in all programming language especially Internet Program, Threat of viruses and abuse of resources are everywhere.
  - Java system not only verify all memory access but also ensure that no viruses are communicated with an applet.

- **Robust**
  - Robust means strong. A robust program is one that does not fail.
  - Java programs are strong and they do not crash easily like a c and c++ program.
  - It has strong memory allocation and automatic garbage collection mechanism.
  - It provides the powerful exception handling and type checking mechanism.

Features

- **Dynamic**
  - Java is dynamic language. It is capable of dynamically linking in new class libraries, methods and objects.
  - Java programs support functions written in C and C++, these functions are known as native methods.
- **Interpreted**
  - Java source file is .java file after compilation it produces .class file which is called byte code (intermediate file). Here an Interpreter is needed to run the .class file from the JVM to get an output.
  - Java is the only language which will do both compilation and Interpretion.

Java is Compiled and Interpreted



Features

- **High performances**
  - Java performance is impressive for an interpreted language, mainly due to the use of intermediate byte code.
  - Java architecture is also designed to reduce overheads during runtime.
- **Distributed**
  - Java is designed as a distributed language for creating applications on networks.
  - It has the ability to share both data and program.
  - Java applications can open and access remote objects on Internet as easily as they can do in a local system.

Features

- **Multithreading**
  - It means handling multiple task simultaneously.
  - Java supports multithreaded programs, it means that we need not wait for the application to finish one task before beginning another.

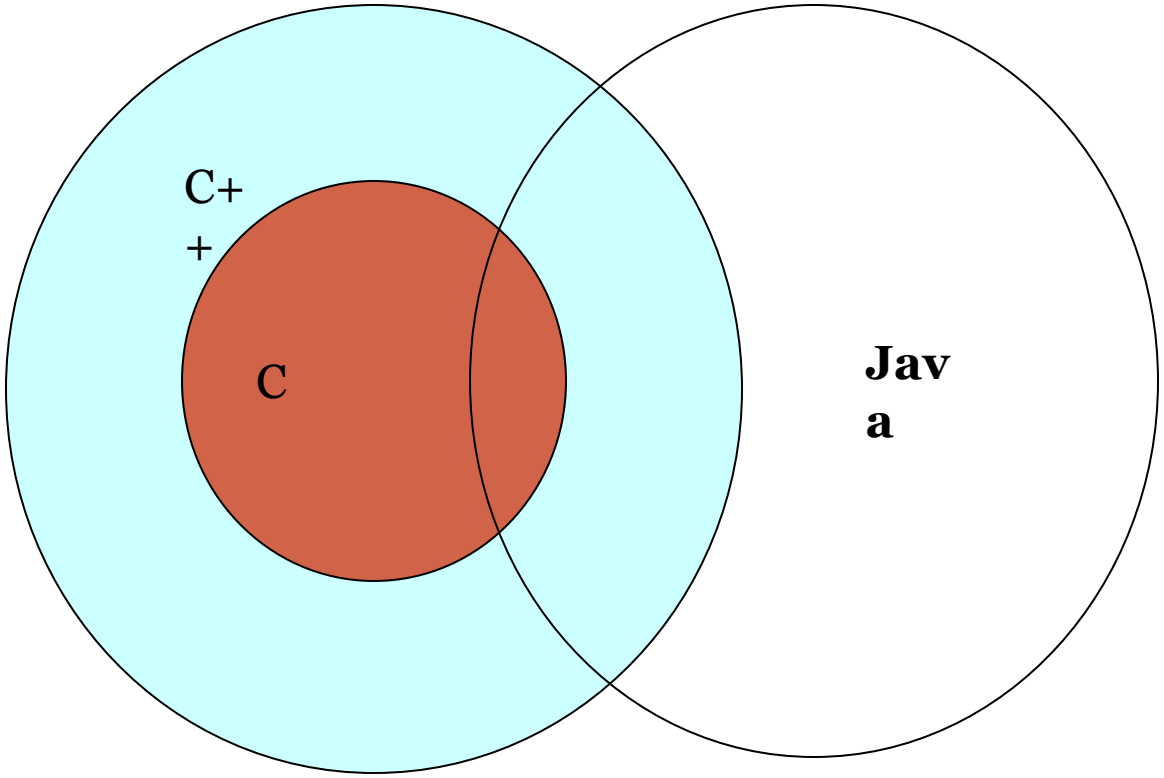
How java differs from c and c++

- **Java and C**
  - Java does not include the C unique statement keywords sizeof, and typedef.
  - Java does not contain the data types struct and union.
  - Java does not define the type modifiers keywords auto, extern, register, signed, and unsigned.
  - Java does not support an explicit pointer type.
  - Java does not have preprocessors like #define, #include statements.
  - Java requires that the functions with no arguments must be declared with empty parenthesis and not with the void keyword as done in c.
  - Java adds labeled break and continue statements.

How java differs from c and c++

- **Java and C++**
- Java does not support operator overloading.
- Java does not have template classes as in c++.
- Java does not support multiple inheritance of classes. This is accomplished using a new feature called “interface”.
- Java does not support global variables.
- Java does not support explicit pointers.
- Java replaced the destructor function with a finalize() function.
- There are no header file in java.

Overlap of C, C++, and Java



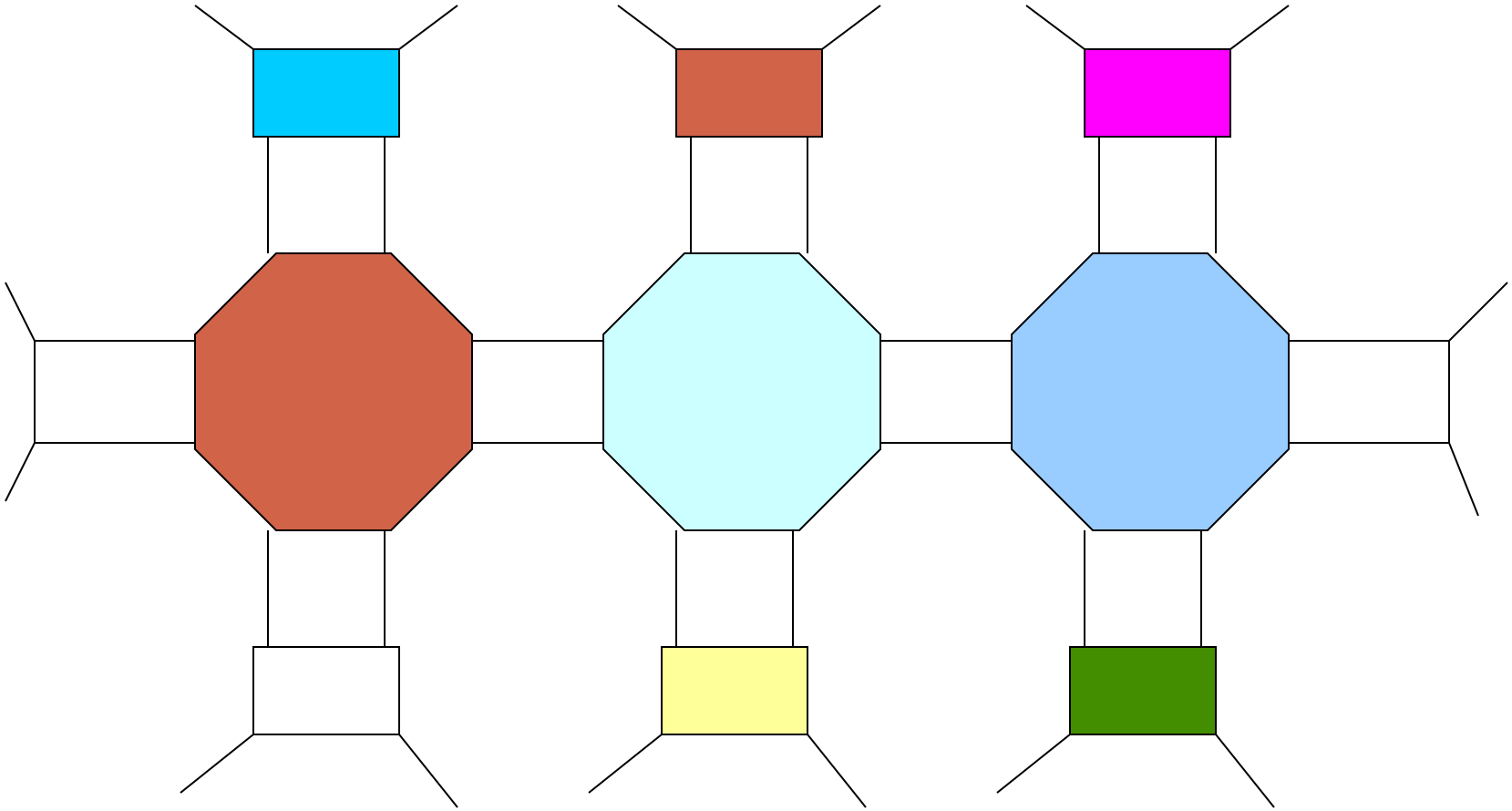
Java and Internet

- Java is strongly associated with the Internet because of the fact that the first application program written in java was HotJava, a Web browser to run applets on Internet.
- Internet users can use java to create applet programs and run them locally using “java enabled browser” such as HotJava.
- Using Java enabled browser the program can be download over Internet and runs it on local computer.
- Using applet program a user can also create web pages and runs on line

Java and World Wide Web

- Web is an open-ended information retrieval system designed to be used in the Internet wide distributed system.
- It contains Web pages (created using HTML) that provide both information and controls.
- Unlike a menu driven system--where we are guided through a particular direction using a decision tree, the web system is open ended and we can navigate to a new document in any direction.
- Java communicates with a web page a special tag called <Applet>
- The user sends request for an HTML document to Web server, the server replied the required document to the user.
- The document contains the APPLET tag which identifies the applet.

Web Structure of Information Search/Navigation



Web Browsers

- A large portion of the Internet is Organized as the “World Wide Wed” which uses hypertext.
- Web browsers are used to navigate through the information found on net.
- They allow as to retrieve the information spread across the Internet and display it using the HTML.
  - HotJava
  - Netscape Navigator
  - Internet Explorer etc..,

Hardware and Software requirements

- The hardware and software requirements of Java are totally depends on the version of Java to be installed and operating system.
- Let us consider of installing J2SE 1.5 on windows XP operating system.
- The minimum physical RAM of 32MB is required to run graphically based applications. More RAM is recommended for applets running with in a browser using Java plug-in.
- Minimum of 120MB of dish space is required.
- 2 GB of Hard Disk Space.
- Internet Explorer 6.0 or higher, Firefox 2.0 of Higher.

http://java.sun.com/j2se/1.5.0/install-windows.html

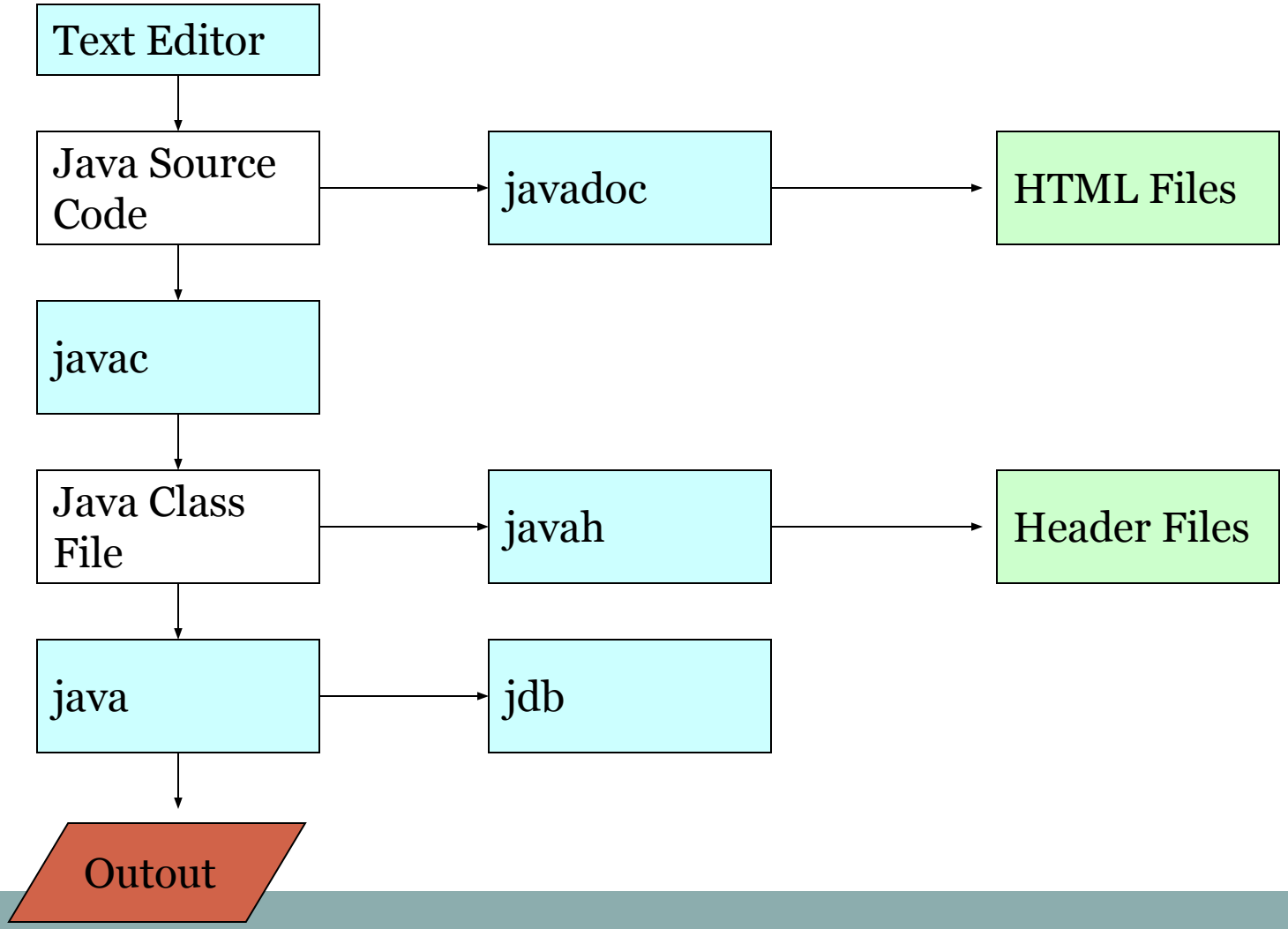
Java Environment

- Java environment includes a large number of development tools and hundreds of classes and methods.
- It development tools are part of the system known as Java Development Kit(JDK).
- The classes and methods are part of the Java Standard Library(JSL) also known as Application Programming Interface (API).

Java Development Kit

- javac - The Java Compiler
- java - The Java Interpreter
- jdb- The Java Debugger
- appletviewer -Tool to run the applets
- javap – java disassembler, which enables us to convert byte code to program description.
- javadoc – creates HTML-format documentation from Java source code files.
- javah – produces header files for use with native methods.

Process of Building and Running Java Programs



Application Programming Interface

- Java Standard Library includes hundreds of classes and methods.

java.lang – Language support Package

java.util – Utilities Package

java.io – Input/Output Packages

java.net – Networking Packages

java.awt – Abstract window toolkit Packages

java.applet – Applet Packages

Java Runtime Environment

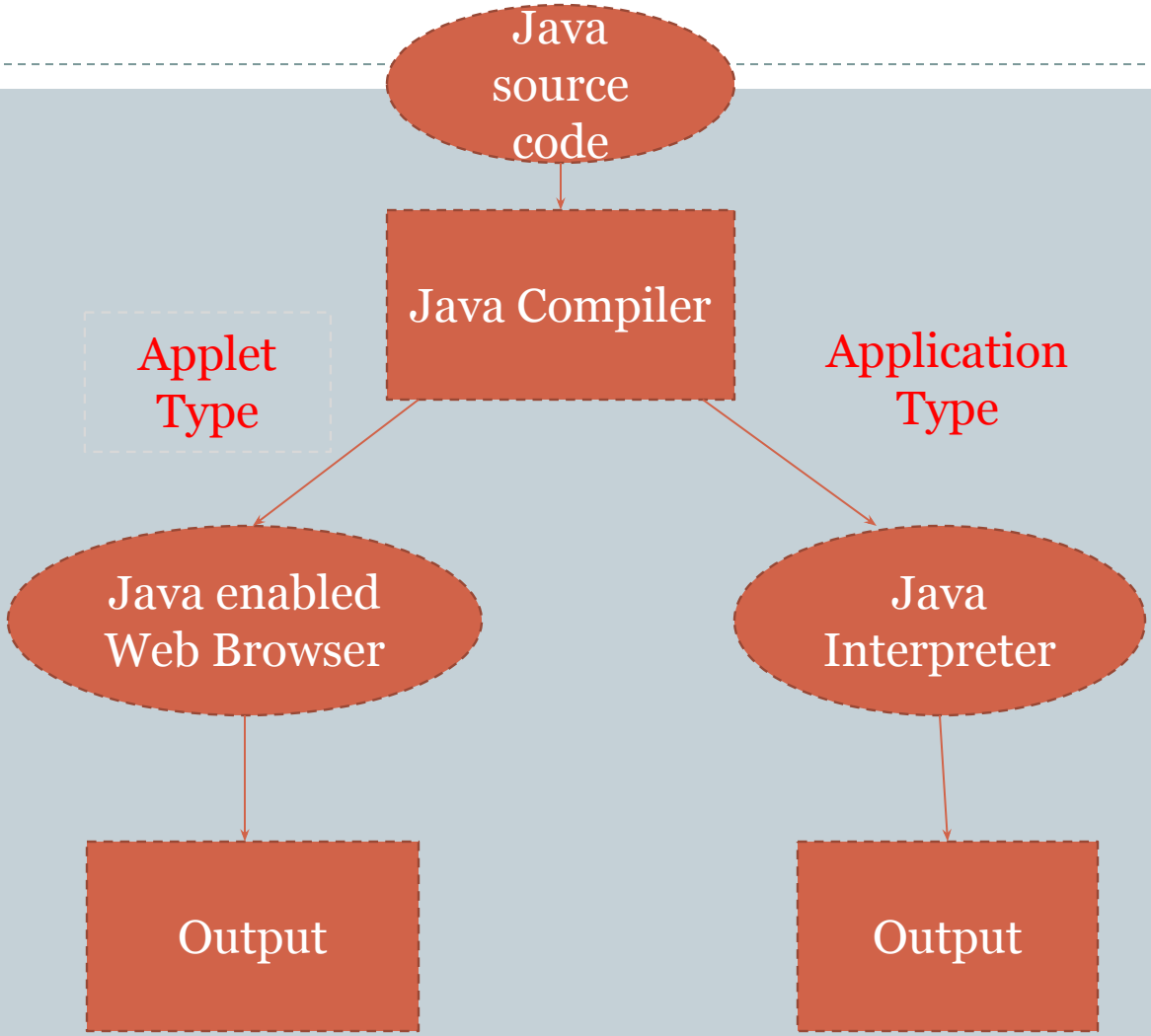
- **Java Virtual Machine (JVM):**
  - It is a program that interprets the intermediate Java byte code and generate the desired output.
  - It is because of byte code and JVM concepts that programs written in Java are highly portable.
- **Run time class libraries:**
  - These are a set of core class libraries that are required for the execution of Java programs.
- **User interface toolkits:**
  - AWT and Swing are examples of toolkits that support varied input methods for the users to interact with the application program.

Java Applications

- We can develop two types of Java programs:
  - Stand-alone applications
  - Web applications (applets)

  Application- A stand-alone program that can be invoked from command line . A program that has a "main" method

  Applet- A program embedded in a web page , to be run when the page is browsed . A program that contains no "main" method
- Application –Executed by the Java interpreter.
- Applet- Java enabled web browser.


Java source code
Java Compiler
Applet Type
Application Type
Java enabled Web Browser
Java Interpreter
Output
Output

Simple Java program

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

- class – keyword , Sample – class name (Identifier/ user defined name)
- public – The key word public is access modifier
- static – This belongs to entire class not a part of particular object.
- void – return type wont return anything.
- main – Main function as like C & C++.
- String args[] – Command Line Argument
- System.out.println() – Display function in java

Compile and Execution

- Type the program in notepad or editor, save the file with class name (main method class) with .java extension.

  Sample.java
- open cmd prompt for the following operations:
- Compilation

  javac filename.java

  eg; javac Sample.java
- Run the program

  java classfilename

  eg; java Sample

Java Program Structure

Documentation Section // Suggested
Package Statement // Optional
Import Statements // Optional
Interface Statement // Optional
Class Definition // Optional
Main method class // Essential **{** **main method definition** **}**

Java Tokens

- Smallest individual unit in a program is token.
- Java language includes 5 types of tokens.

- Reserved Keywords
- Identifiers
- Literals
- Operators
- Separators

Continuation…

- **Keywords**
  - Keywords are reserved word or system defined word or predefined word.
  - We cant change the meaning of the word
    e.g., class, public, int, float, for, import, etc….,
- **Identifiers**
  - Identifiers are user defined word.
  - It can be used for naming variables, class name, object name, method name, interface name, etc..,
  - They can have alphabets, digits, underscore and dollar sign characters.
  - It can be of any length.
    e.g., sample, student, etc..,

Continuation…

- **Literals**
  - Literals in java are a sequence of characters (digits, letters, and other characters) that represent constant values to be stored in variables.
  - Integer, Floating point, Character, String, Boolean Literals are there.
- **Operators**
  - An operator is a symbol that takes one or more arguments and operates on them to produce a result.
- **Separators**
  - Separators are symbols used to indicate where groups of code are divided and arranged.
    - Parentheses ()
    - Braces { }
    - Brackets [ ]
    - Semicolon ;
    - Comma ,
    - Period .

Classes

- Classes provide a convenient method for packing together a group of logically related ***data*** items and ***methods /functions*** that work on them.

Defining a class:

```
class classname
{
        field/data declarations;
        method declarations;
}
```

example

```
class Sample
{
        int a,b,c;
        void input()
        {
                a=10;
                b=20;
        }
}
```

Objects

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

```
Sample obj;              // declare the object, object name is user defined name.
obj=new Sample();   // instantiate the object
```

- It is also written as

```
Sample obj=new Sample();
```

- Here obj is the object of class Sample, by creating obj is to allocate memory for the Sample class properties and also by using obj only we can able to access the Sample class data and methods.

Accessing class Members

- Each object containing its own set of variables.
- We should assign values to these variables in order to use them in our program.
- Since we are creating object outside the class, we cannot access the instance variables and the methods directly.
- To do this, we must use the concerned object and the *dot* operator.

```
Objectname.vaiablename=value;
Objectname.methodname(parameter-list);

Ex.,

Obj.a=10;
Obj.b=50;
Obj.sum();
```

Example for class and object

```
class Sample
{
        int a,b,c;
        void input()
        {
                a=10;
                b=20;
        }
        void cal()
        {
                c=a+b;
        }
        void display()
        {
                System.out.println(c);
        }
}
class Test
{
        public static void main(String arg[])
        {
                Sample s=new Sample();
                s.input();
                s.cal();
                s.display();
        }
}
```

Constructors

- Constructor is a special type of method, that enables an object to initialize itself when it is created.
- Constructor have the same name as that of the class.
- Constructors does not specify a return type, not even void because they return the instance of the class itself.

Example

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

Example

```
 class RectArea
{
public static void main(String args[])
{
 Rectangle rect1 = new Rectangle(15,10);//calling constructor
 int area1=rect1.rectArea();
 System.out.println("Area1=" +area1);
}
}
```

Default Constructor

- **Default Constructor :** Constructor without any parameter is called default constructor.

Example:

```
Rectangle()          //default constructor
{
    length=0;
    width=0;
}
```

Parameterized Constructor

- **Parameterized Constructor:** Constructor with parameter is called as Parameterized Constructor.

Example:

```
Rectangle(int x, int y)
{
    length=x;
    width=y;
}
```

Methods Overloading

- Methods overloading is defined as creating methods that have the same name, but different parameter lists and different definitions. This process is known as polymorphism.
- Here method's return type does not play any role.
- Is used when objects are required to perform similar tasks but using different input parameters.

Example

```
class OverloadDemo
{
    void test()
    {
    System.out.println("No Parameters");
    }
    void test(int a)
    {
    System.out.println("a:"+a);
    }
    void test(int a,float b)
    {
    System.out.println("a and b:"+a+" "+b);
    }
}
```

Example

```
 class Overload
{
 public static void main(String args[])
{
OverloadDemo obj = new OverloadDemo();
 obj.test();
 obj.test(10);
 obj.test(20,45.89);
}
}
```

Static Members

- Is used to define a member that is common to all the objects and accessed without using a particular object.
- Here the member belongs to the class as a whole. To do this the keyword static is used.

Examples:

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

- The members that are declared static are known as static members.
- The static variables are also called as class variables and static methods as class methods.
- Java creates only one copy for a static variable which can be used even if the class is never actually instantiated.
- The static variable and static methods are called without using the objects.

Example

```
class MathOperation
 {
 static float mul(float x, float y)
 {
       return(x * y);
 }
 class MathApplication
 {
 public static void main(String args[])
 {
 float a=MathOperation.mul(8.6,5.2);
 System.out.println("a=" +a);
 }
}
```

Nesting Methods

- A method can be called by using only its name by another method of the same class by implementing nesting of methods.

Example

```
class Nesting
{
int m,n;
Nesting(int x, int y)
{
m=x;
n=y;
}
```

Example

```
int largest()
{
if(m>=n)
return(m);
else
return(n);
}
void display()
{
 int large=largest();
 System.out.println("Largest value is:"+large);
}
}
```

Example

```
class NestingTest
{
public static void main(String args[])
{
Nesting nest=new Nesting(50,40);
nest.display();
}
}
```

Inheritance

- The process of deriving a new class from an already existing class is called Inheritance.
- Old class – Base class/super/parent class.
- New class – Derived/subclass/child class.
- Allows subclasses to inherit all the variables and method of their parent class.

Different forms of Inheritance

- Single Inheritance – only one super class.
- Multiple Inheritance – several super classes: Java does not directly implement multiple inheritance, but it is implemented by using Interface.
- Hierarchical Inheritance – one super class, but many subclasses.
- Multilevel Inheritance – derived from a derived class.

Extending a Class

Syntax:

```
class subclassname extends superclassname
{
variable declarations;
method declarations;
}
```

Example

```
class A
{
int i=10, j=20;
void showij()
{
System.out.println("I and J"+ i+" "+ j);
}
}
class B extends A
{
int k=5;
void showk()
{
System.out.println("k" +k);
}
void sum()
{
System.out.println("i+j+k" +(i+j+k));
}
}
```

Example

```
class Inheritance
{
public static void main(String args[])
{
A obj1 = new A();
B obj2 = new B();
obj1.showij();
obj2.i=4;
obj2.j=3;
obj2.showij();
obj2.showk();
obj2.sum();
}
}
```

Overriding Methods

- It is possible to define a method in the subclass that has the same name, same arguments and same return type as a method in the super class.
- When that method is called, the method defined in the subclass is invoked instead of one in the super class.
- This is known as overriding method or method overriding.

Example

```
class A
{
int x;
A(int x)
{
this.x=x;
}
void display()
{
System.out.println("A------- x=" +x);
}
}
```

Example

```
class B extends A
{
int y;
B(int x, int y)
{
super(x);
this.y=y;
}
void display()
{
System.out.println("A---------x:"+x);
System.out.println("B---------x:" +y);
}
}
```

Example

```
class Override
{
public static void main(String args[])
{
B obj=new B(100,200);
obj.display();
}
}
```

The final variables and methods

- To prevent the sub classes from overriding the members of the super class, we can declare them as final using the keyword final as the modifier.

Example

```
final int SIZE = 100;
final void showstatus()
{
    ------
    ------
}
```

- The value of a final variable and the functionality of a final method cannot be altered in any way.
- The final variables behave like class variables.

The final class

- A class that cannot be sub classed for security reasons is called a final class.
- Example

```
final class sample
{
---------
---------
}
```

- Any attempt to inherit these classes will cause an error.

The finalize method

- Finalization is just opposite to initialization.
- Java run – is an automatic garbage collecting system. ie., it automatically frees up the memory resources used by the objects.
- Garbage collector cannot free other resources such as file descriptors, window system fonts etc.
- The finalize() method, which is similar to destructor is used for that.
- The finalize() method can be added to any class.
- The finalize() method is only called just before prior to garbage collection.
- Inside the finalize() method, we'll specify those actions to be performed before an object is destroyed.

Example

```
protected void finalize()
{
//finalization code
}
```

Abstract methods and classes

- The function of abstract modifier is just opposite to final modifier.
- The abstract modifier is used to indicate that a method must always be redefined in a subclass.
- When a class contains one or more abstract methods, it should also be declared as abstract.
- We can’t use abstract classes to instantiate objects directly.

Example

```
 abstract class A
{
 abstract void callme();
 void callmetoo()
 {
 System.out.println("This is a concrete method");
 }
}
 class B extends A
 {
 void callme()
 {
 System.out.println("B's implementation of callme:");
 }
}
```

Example

```
class AbstractDemo
{
public static void main(String args[])
{
B b=new B();
b.callme();
b.callmetoo();
}
}
```

Visibility Control

- In some situations, it is necessary to restrict the access to certain variables and methods from outside the class.
- Visibility modifiers/access modifiers in java determine the visibility and scope of the java elements.

The private modifier:

- Achieves the lowest level of accessibility.
- Are accessible only with their own class.
- Cannot be inherited by sub class.
- Sub class cannot invoke the private members of the base class.
- A method declared as private behaves like a method declared as final.
- Example:

```
private int number;
private void sum();
```

The protected modifier

- Accessibility lies in between the public access and private access.
- Child class can access the base class protected data directly (in the same package and other packages too).
- Non- sub classes in other packages cannot access the protected members.
- Example:

```
protected int number;
protected void sum();
```

The public modifier

- Achieves the highest level of accessibility.
- The most known public method is method is the main() method.
- The public members can be accessed from anywhere.
- Example

  public int number;

  public void sum();
- Is not good OOP technique to define instance variables as public.

The default modifier

- If we don’t set access modifier for a member, the compiler will consider it as the default modifier.
- There is no default modifier keyword.
- Using the members can be accessed while we are in the same package.

The difference between the default and protected modifier

- In default, to access a member from the child class, the class must be in the same package of the super class while in the protected case, it can be from out of the super class package.

Arrays

- Group of related data items that share a common name.
- Array elements can be accessed by using the array name and the respective index.
- In java, subscripts start with the value 0.

Example:

salary[10]

Declaration of arrays

- The two ways to declare array in java are:

type arrayname[];

or

type[] arrayname;

Examples:

int number[];

or

int[] number;

- The size of the array is not mentioned in the declaration.

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 a one dimensional array.
- Example

  number[10]

Creating memory locations

- This is done by new operator.

  arrayname = new type[size];

Example:

  number = new int[3];

- It is also possible to combine declaration and creating memory locations as shown below

  int number [] = new int[3];

Creating an array

- Arrays must be declared and created in the computer memory before they are used.
- Creation of array involves 3 steps:

  1. Declaring an array
  2. Creating memory locations
  3. Putting values into the memory locations or initialization of arrays.

- Putting values into the memory location or initialization of arrays

Syntax: arrayname[subscript]=value;

Example: number[0]=20;

number[1]=15;

number[2]=35;

type arrayname[] = {list of values};

Example: int number[] = {20, 15,35};

- it is possible to assign an array object to another.

```
Example:
int a[]={1,2,3};
int b[];
b=a;
```

- Loops may be used to initialize large size arrays.

Example:

```
for(int i=0;i<100;i++)
{
if(i<50)
sum[i]=0.0;
else
sum[i]=1.0;
}
```

Two-dimensional arrays

- Two-dimensional arrays are used to store a table of values.
- 2 subscripts are used here – one for row and the other for column.
- Creation of two-dimensional array is same as that of one-dimensional array.
- Example:

```
int myarray[][];
myArray=new int[2][3];
        or
int myArray=new int[2][3];
```

- We can obtain the length of the array using a.length.

Example

```
int asize=a.length;
```

- Initialization of two-dimensional array can be done in different ways:

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

or

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

Sample program for Array

```
class Average
    {
    public static void main(String args[])
    {
    int nums[]={10,23,12,60};
    float result=0;
    int i;
    for(i=0;i<4;i++)
    result= result+ nums[i];
    System.out.println("Average is:" +result/4);
    }
    }
```

Strings

- String represents a sequence of characters.
- In Java, strings are class objects.
- In Java, strings are implemented using String and StringBuffer classes.
- A Java string is an instantiated object of the string class.
- Because of bounds checking, Java strings are more reliable.
- A Java string is not a character array.
- A Java string is not NULL terminated.

Strings

Declaration and Creation of string

String stringName;

stringName = new String(“content of the string”);

Example

String firstName;

firstName=new String(“welcoming”);

Easiest way of creation of a string

String stringName=“content of the string”;

Example

String sample=“welcome to Java Programming”;

StringBuffer Class

- StringBuffer is a peer class of String.
- While String class creates strings of fixed length, StringBuffer class creates strings of flexible length that can be modified in terms of both length and content.
- It provides an efficient approach to dealing with strings, especially for large dynamic string data.
- Using StringBuffer class the memory allocated to an object is automatically expanded to take up additional data.
- StringBuffer is faster than String when performing concatenations.
- A string that can be modified or changed is known as mutable string. StringBuffer and StringBuilder classes are used for creating mutable string.

Finding the length of the string

length() method is used

Example

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

Concatenation of strings

Java strings can be concatenated by using the + operator

Examples

```
String sample="I"+"like"+"Java";
System.out.println(firstName + "Everyone");
```

Commonly used StringBuffer class methods

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) – inserts the string s2 at the position n of the string s1

s1.setLength(n) – sets the length of the string s1 to n.

String arrays

Can create and use arrays that contain strings.

Example

String itemArray[]=new String[3];

String Methods

- The string class defines a number of methods that allow to do string manipulation tasks.
- The most commonly used string methods are given below:

str.charAt(k)

-returns a char at position k in str.

str2=str1.toLowerCase

-converts the string str1 to all lower case

str2=str1.toUpperCase

- converts the string str1 to all uppercase

str.substring(k)

- returns a substring from index k to the end of str

String Methods

s.substring(k,n)
- returns a substring from index k to index n-1 of str

str.indexOf(s)
- returns an index of the first occurrence of string s in str

str.indexOf(s,k)
- returns an index of string s starting from an index k in str

str.startsWith(s)
- returns true if str starts with s

str.startsWith(s,k)
- returns true if str starts with s at index k

str1.equals(str2)
- returns true if the two strings have equal values.

str1.equalsIgnoreCase(str2)
- same as above ignoring case

str1.compareTo(str2)
- compares two strings

str1.compareToIgnoreCase(str2)
- same as above ignoring case

Vectors

- Vectors is a class contained in the java.util package.
- Vector class is used to create dynamic array known as vector that can hold objects of any type and any number.

Examples for creation of vectors

Vector int Vect=new Vector(); // declaring without size

or

Vector list=new Vector(3); //declaring with size

- A vector **without any size** can accommodate an unknown number of items.
- Even, **when a size is specified,** after the initial capacity is reached, the next time when we attempt to store an object in the vector, the vector can automatically allocate space for additional objects.

Vectors

Advantages of Vector over Arrays

- It is convenient to use vector to store objects.
- A vector can be used to store a list of objects that may vary in size.
- We can add and delete objects from the list as and when required.

Major constraints in using vectors

- Cannot directly store simple data type in a vector.
- Can only store objects.
- Need to convert simple types to objects which is done using wrapper classes.

Commonly used Vector class methods

- list.addElement(item) – adds the item to the list at the end.
- list.elementAt(5) – gives the name of the 5th object.
- list.size() – gives the number of objects present.
- list.removeElement(item) – removes the specified item from the list.
- list.removeElementAt(n) – removes the item stored at the nth position.

Wrapper Classes

- The primitive data types are not objects: they do not belong to any class; they are defined in the language itself.
- Wrapper classes are used to convert any data type into an object.
- At the name says, a wrapper class (encloses) around a data type and gives it an object appearance.
- Wherever, the data type is required as an object, this object can be used.
- Wrapper classes include methods to unwrap the object and give back the data type.

Wrapper Classes

Converting primitive numbers to object number using constructor methods:

- Integer IntVal=new Integer(i);

// primitive int to Integer object

- Float FloatVal=new Float(f);

// primitive float to Float object

- Double DoubleVal = new Double(d);

//primitive double to Double object

- Long LongVal=new Long(l);

//primitive long to Long object

Wrapper Classes

Primitive Data type	Wrapper Class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

Converting object numbers to primitive numbers using typeValue() method

- int i =IntVal.intValue();

// object to primitive integer

- float f= FloatVal.floatValue();

// object to primitive float

- double d=DoubleVal.doubleValue();

//object to primitive double

- Long l = LongVal.longValue();

//object to primitive object

Importance of Wrapper classes

- There are mainly 2 uses with wrapper classes.
- To convert simple data types into objects, that is, to give object form to a data type; here constructors are used.
- To convert strings into data types (known as parsing operations), here methods of type parseXXX() are used.

Converting numeric Strings to primitive numbers using parsing methods

- int i=Integer.parseInt(str); // converts string to primitive integer.
- long l=Long.parseLong(str); // converts string to primitive long

Chapter-3
Interfaces

Multiple Inheritance: Introduction

- A large number of real – life applications require the use of multiple inheritance (inheriting methods and properties from distinct classes)
- C++'s implementation of multiple inheritance is difficult and complexity.
- Java provides an alternate approach known as Interfaces to support the concept of multiple inheritance.

Defining Interfaces

- An Interfaces is basically a kind of class, but with a major difference.
- The difference is that interfaces define only abstract methods and final fields (do not specify any code and implement these methods and data fields contain only constants).
- The class that implements the interface will define the implementation of methods in the interface.

Interfaces

Syntax

```
interface Interfacename
{
    variables declaration;
    methods declarations;
}
```

Syntax for declaring variable in interface

```
static final datatype variablename=value;
```

Syntax for declaring methods in interface

```
returntype methodName (parameter_list);
```

Example

```
interface Sample
{
    static final int n=100;
        void display();
float compute(float x, float y);
}
```

Extending Interface

- The Interface can be extended.
- The subinterface will inherit all the members of superinterface.

Syntax

```
interface InterfaceName2 extends InterfaceName1
{
    body of InterfaceName2
}
```

Example - 1

```
 interface A
{
 int x=50;
 void meth1();
 void meth2();
}
 interface B extends A
 {
 int y=30;
 void meth3();
}
```

Example - 2

```
interface A
{
 int x=50;
 void meth1();
 void meth2();
}
 interface B
 {
 int y=30;
 void meth3();
}
 interface C extends A,B
{
 int z=40;
 void meth4();
}
```

Implementing Interface

- Once an interface has been defines, one or more classes can implement that interface.
- To do this, include the implements clause in the class definition and then define the methods of the interface.

Syntax

```
class Classname  implements interfacename
{
 body of the class
}
```

Another general form of interface implementation

class Classname extends Superclassname implements interfacename1, interfacename2...

{

body of the class

}

In the above form, a class is extending another class while implementing interfaces

Example program for Interface

```
 interface Area
{
 final static float pi=3.14F;
 float compute(float x, float y);
}
 class Rectangle implements Area
 {
 public float compute(float x, float y)
{
 return(x * y);
}
}
```

```
class Circle implements Area
{
public float compute(float x, float y)
{
return(pi * x * x);
}
}
class InterfaceTest
{
public static void main(String args[])
{
```

```
 Rectangle rect = new Rectangle();
 Circle cir=new Circle();
 Area z;        // Interface Object
 z= rect;            // z refers to rect object
 System.out.println("area of rectangle=" +
z.compute(10,20);
z= cir;             // z refers to cir object
 System.out.println("area of Circle=" + z.compute(20);
}
}
```

Example Program2 for Interface

```
interface A
{
void meth1();
void meth2();
}
interface B extends A
{
void meth3();
}
```

```
 class Myclass implements B
 {
 public void meth1()
{
System.out.println(“implement meth1() method”);
}
 public void meth2()
{
System.out.println(“implement meth2() method”);
}
 public void meth3()
{
System.out.println(“implement meth3() method”);
}
```

```
class IFExtend
{
public static void main(String args[])
{
Myclass obj=new Myclass();
obj.meth1();
obj.meth2();
obj.meth3();
}
}
```

Accessing Interface Variables

- The interfaces can be used to declare a set of constants that can be used in different classes.
- The constant values will be available to any class that implements the interface.

Example

```
 interface A
{
 int m=10;
 int n=50;
 }
 class B implements A
{
 int x=m;
 void meth()
 {
 ----------
 ----------
}
}
```

Packages: putting classes together
Introduction

- Packages are java's way of grouping a variety of classes and/or interfaces together according to the functionality.
- Using packages is a way to achieve the reusability in java.
- Java packages can be classified into 2 groups:
  1. Java API packages
  2. User defined packages

java.lang -> Languages support classes. Automatically imported. Include classes for primitive types, strings, Math functions, threads and exceptions.

java.util -> Languages utility classes such as vectors, random numbers, date etc.

java.io -> input/ouput support classes.

java.awt -> classes for implementing GUI, include classes for windows, buttons, lists, menus etc.

java.net -> classes for networking

java.applet -> classes for creating and implementing applets

Using System packages

- The packages are organized in a hierarchical structure.
- There are 2 ways of accessing the classes stored in a package.

1. By using the package name containing the class and then appending the class name to it using the dot operator.

example: java.awt.Font;

2. By using the import statement.

```
import packagename.classname;
        or
import packagename.*;
```

example: import java.awt.Font;

or

import java.awt.*; //will bring all the classes of java.awt package

Naming Conventions

- Standard naming rules are used.
- By convention, packages begin with lower case letters.
- Every package name must be unique because duplicate names will cause run-time errors.
- The package naming convention that ensures uniqueness suggests to use the domain names as prefix to the preferred package name.
- Can also create hierarchy of packages.

Creating packages

- Declare the package at the beginning of the file using the form

  package packagename;
- Define the class that is to be put in the package and declare it public

  public class Classname
  {
  body of the class
  }
- Create a subdirectory named packagename under the directory. The subdirectory must match the packagename.
- Save the file by giving classname.java as the file name in the subdirectory created.
- Compile the file to create the .class file in the subdirectory.

Multithreading Programming - Introduction

- The ability to execute several programs simultaneously is known as multitasking. In system's terminology, it is called multithreading.
- Multithreading is a conceptual programming paradigm where a program(process) is divided into two or more subprograms(processes), which can be implemented at the same time in parallel.
- A thread is similar to a program that has a single flow of control, i.e., a beginning a body and an end and executes commands sequentially.

- Java enables to use multiple flows of control in developing programs.
- Each flow of control that runs in parallel is a separate tiny program known as thread.
- A program that contains multiple flows of control is known as multithreaded program.
- The ability of a programming language to support multiple threads is referred to as concurrency.
- Since threads in java are subprograms of a main application program and share the same memory space, they are known as lightweight threads or lightweight processes.

- Threads running in parallel does not actually run at the same time.
- The flow of execution is shared between threads.
- The java interpreter handles the switching of control between threads in such a way that it appears they are running concurrently.

Difference between multi-tasking and multi-threading

Multitasking	Multithreading
More than one program gets executed simultaneously.	More than one part of a program called threads is executed simultaneously.
Multitasking is a timesharing process. CPU switches from one program to another program so quickly to complete all the programs simultaneously.	Multithreading is also a timesharing process. CPU switches from one activity to another activity within the program so quickly to complete all the activities simultaneously.
Since each program occupies different memory location, Multitasking is called as heavyweight process.	Multithreading is a light weight process because the activities/ threads share the same memory space.

- A new Thread can be created in two ways:-

1. By creating a Thread class: Define a class that extends thread class and override its run() method with the code required by the thread.
2. By converting a class to a Thread: define a class that implements Runnable interface which has only one method run().

Extending a Thread class

- Declare a class as extending the Thread class.
- Implement the run() method that is responsible for executing the sequence of code that the thread will execute.
- Create a thread object and call the start() method to initiate the thread execution.

Declaring the class

```
class MyThread extends Thread
{
---------
---------
}
```

Implementing the run() method

```
public void run()
{
---------
----------
}
```

Creating a thread object

```
MyThread aThread = new MyThread();
aThread.start();
```

Example

```
class A extends Thread
{
public void run()
{
for(int i=1;i<=5;i++)
{
System.out.println("From Thread A: i=" +i);
}
System.out.println("Exit from A");
}
}
```

```
class B extends Thread
{
public void run()
{
for(int j=1;j<=5;j++)
{
System.out.println(“From Thread B: j=“ +j);
}
System.out.println(“Exit from B”);
}
}
```

```
class C extends Thread
 {
 public void run()
 {
 for(int k=1;k<=5;k++)
 {
 System.out.println(“From Thread C: K=“ +k);
}
 System.out.println(“Exit from C”);
}
}
```

```
class ThreadTest
{
public static void main(String args[])
{
A threadA=new A();
threadA.start();
B threadB=new B();
threadB.start();
C threadC=new C();
threadC.start();
}
}
```

Stopping and blocking a Thread

Stopping a Thread

- The stop() method is used.
- It causes the thread to move to the dead state.
- It is used when the premature death of a thread is desired.

Ex: threadA. stop();

Blocking a Thread

- A thread can be temporarily suspended by using either of the following thread methods:

sleep(); //blocked for a specified time

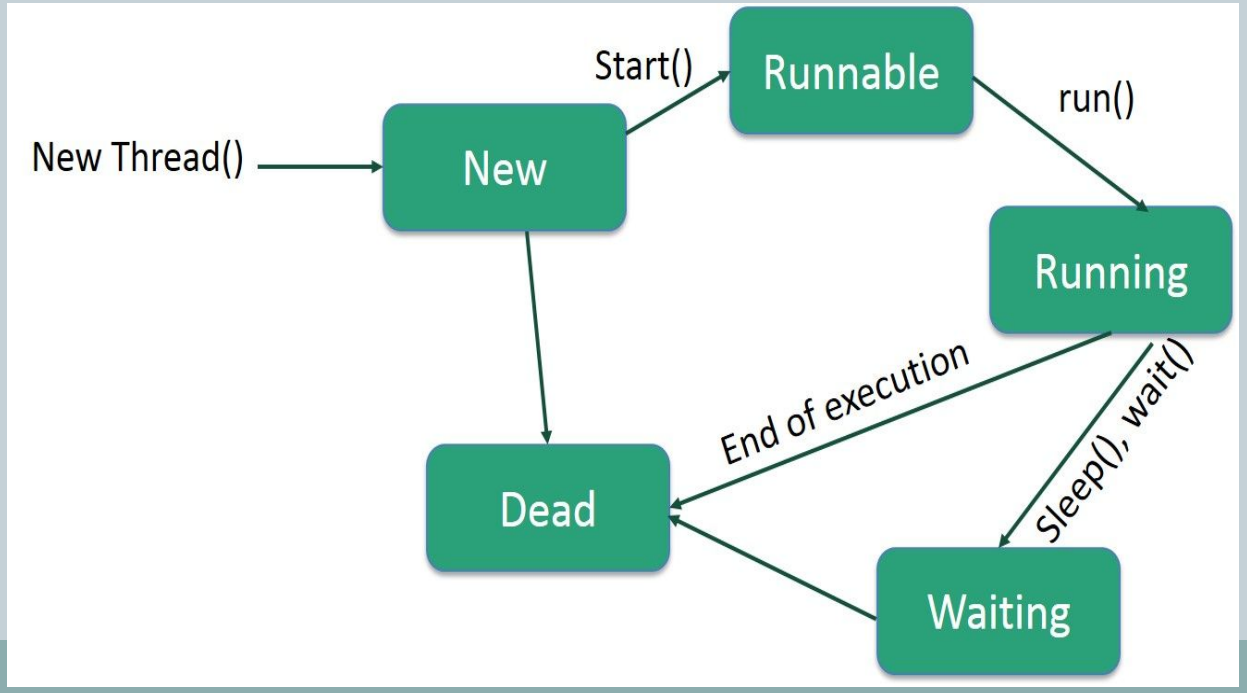
suspend(); //blocked until further orders

wait() //blocked until certain condition occurs

- The thread will return to runnable state when the specified time is elapsed in the case of sleep(), the resume() method invoked in the case of suspend(), and the notify() method is called in the case of wait().

Life Cycle of a Thread

- A thread goes through various stages in its life cycle. For example, a thread is born, started, runs, and then dies.
- Following diagram shows complete life cycle of a thread.



- **New:** A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a born thread.
- **Runnable:** After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.
- **Waiting:** Sometimes, a thread transitions to the waiting state while the thread waits for another thread to perform a task. A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.
- **Timed waiting:** A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.
- **Terminated (Dead):** A runnable thread enters the terminated state when it completes its task or otherwise terminates.

Using Thread Methods

- The class Thread contains methods which can be used to control the behavior of a thread.

Example:

```
class A extends Thread
{
public void run()
{
for(int i=1;i<=5; i++)
{
if(i==1) yield();    //yield() method is used to relinquish
                control to another thread of equal
    priority before its turn comes
```

```
 System.out.println("from thread A:i="+i);
}
System.out.println("Exit from A");
}
}
 class B extends Thread
 {
 public void run()
 {
 for(int j=1;j<=5;j++)
 {
System.out.println("From Thread B:j="+j);
 if(j==3) stop();
}
System.out.println("exit from B")'
}
}
```

```
class ThreadTest
{
public static void main(String args[])
{
A threadA = new A();
threadA.start();
B threadB = new B();
threadB.start();
}
}
```

Thread Exceptions

- Java run – time system will throw IllegalThreadException whenever an attempt to invoke a method that a thread cannot handle in the given state.

Example

A sleeping thread cannot deal with the resume() method because a sleeping thread cannot receive any instructions.

```
class C extends Thread
{
public void run()
{
for(int k=1;k<=5;k++)
{
System.out.println("From Thread C:k="+k);
if(k==z)
try
{
sleep(1000);
}
catch(Exception e)
{
}
}
System.out.println("Exit from C");
}
}
```

Thread Priority

- Each thread has a priority.
- Priorities are represented by a number between 1 and 10.
- In most cases, thread scheduler schedules the threads according to their priority (known as preemptive scheduling). But it is not guaranteed because it depends it depends on JVM specification that which scheduling it chooses.
- Java permits us to set the priority of a thread using the setPriority() method as follows:-

```
ThreadName.setPriority(value);
```

3 constants defined in Thread class:

- public static int MIN_PRIORITY
- public static int NORM_PRIORITY
- public static int MAX_PRIORITY

Default priority of a Thread is 5 (NORM_PRIORITY).

The value of MIN_PRIORITY is 1 and the value of MAX_PRIORITY is 10.

Example

```
class A extends Thread
 {
 public void run()
 {
 for(int i=1;i<=5;i++)
 {
 System.out.println(“From Thread A: i=“ +i);
 }
System.out.println(“Exit from A”);
}
}
```

```
class B extends Thread
 {
 public void run()
 {
 for(int j=1;j<=5;j++)
 {
 System.out.println(“From Thread B: j=“ +j);
}
 System.out.println(“Exit from B”);
}
}
```

```
class ThreadTest
 {
 public static void main(String args[])
 {
A threadA=new A();
B threadB=new B();
threadB.setPriority(Thread.MAX_PRIORITY);
threadA.setPriority(Thread.MIN_PRIORITY);
threadA.start();
threadB.start();
}
}
```

Synchronization

- Synchronization in java is the capability of controlling the access of multiple threads to any shared resource.
- The Synchronization is mainly used to

1. To prevent thread interference.
2. To prevent consistency problem.

- There are 2 types of synchronization

1. Process synchronization
2. Thread synchronization

Thread Synchronization

- If we declare any method as synchronized, it is known as synchronized method.
- Synchronized method is used to lock an object for any shared resource.
- When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.
- For example, the method that will read information from an file and the method that will update the same file may be declared as synchronized.
- The keyword synchronized is used for this purpose.

Syntax

```
class Classname extends Thread
{
synchronized methodname()
{
--------------
--------------
}
}
```

Implementing the Runnable Interface

- Declare the class that implements the Runnable interface.
- Define the run() method.
- Create a thread by defining an object for the class that implements the Runnable interface.
- Call the thread's start() method to run the thread.

Example

```
class X implements Runnable
{
public void run()
{
for(int i=1;i<=5;i++)
{
System.out.println("From Thread X: i=" +i);
}
System.out.println("Exit from X");
}
}
```

```
class RunnableTest
{
public static void main(String args[])
{
X obj=new X();
Thread threadX=new Thread(obj);
threadX.start();
System.out.println("End of the main thread");
}
}
```

Chapter-4
Managing Errors and Exceptions

Introduction

- Errors are the wrongs that can make a program go wrong.
- An error may produce an incorrect output or may terminate the execution of the program abruptly or even may cause the system to crash.

Types of errors

1. Compile – time errors:

- Syntax errors detected and displayed by the java compiler comes under this category.

Ex: missing semicolons, misspelling of identifiers and keywords, use of undeclared variables etc.

2. Run-time errors:

- Errors that occur during run-time.

Ex: dividing an integer by zero, accessing an element that is out of the bounds of an array, attempting to use a negative size for an array etc.

Exceptions

- An exception is a condition that is caused by a run-time error in the program.

Examples: dividing an integer by zero, trying to store a value into an array of an incompatible type, accessing a character that is out of bounds of a string etc.

Types of Exceptions

The 2 types of exceptions in java are checked exceptions and unchecked exceptions.

- Checked exceptions: is an exception that occurs at the compile time, these are called as compile time exceptions. For example, if we use FileReader class in our program to read data from a file, if the file specified in its constructor doesn't exist, then a FileNotFoundException occurs, and compiler prompts the programmer to handle the exception.

Unchecked Exceptions:

- An unchecked exception is an exception that occurs at the time of execution, these are also called as Runtime Exceptions, and these include programming bugs, such as logic errors improper use of an API.
- Runtime exceptions are ignored at the time of compilation.
- For example, if you have declared an array of size 5 in your program and trying to call the 6^{th} element of the array then an ArrayIndexOutOfBoundsException exception occurs.

Exception handling

- When the java interpreter encounters an exception, it creates an exception object and throws it.
- If the exception object is not caught and handled properly, the program will be terminated.
- If the program should continue with the execution of the remaining code, we should try to catch the exception object thrown and then take the corrective actions.
- This task is known as exception handling.

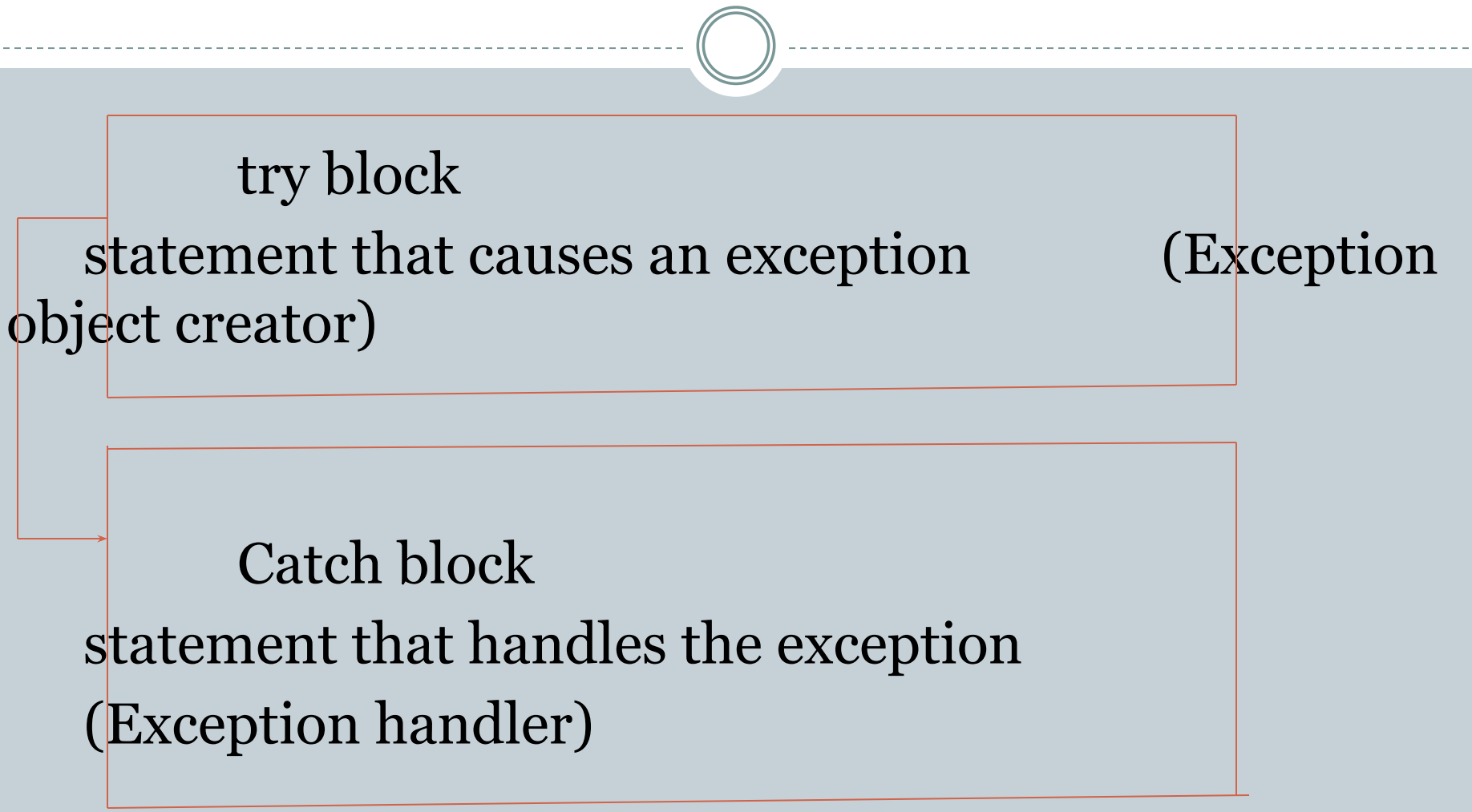
Exception handling mechanism performs the following tasks:

- Find the problem (hit the exception)
- Inform that an error has occurred (throw the exception)
- Receive the error information (catch the exception)
- Take corrective actions (handle the exception)

Common Java Exceptions

Exception Type	Cause of Exception
1. ArithmeticException	caused by Math errors such as division by zero.
2. ArrayIndexOutOfBoundsException	caused by an wrong array indexes.
3. FileNotFoundException	caused by an attempt to access a nonexistent file
4. NullPointerException	caused by referencing a null object.
5. IOException	caused by general i/o failures, such as inability to read from a file.

```
----------
----------
 try
{
     statement;
}
 catch()
{
     statement;
}
----------
----------
```


try block
statement that causes an exception (Exception object creator)
Catch block
statement that handles the exception
(Exception handler)

- The try block can have one or more statements that would generate an exception.
- The catch block too can have one or more statements that are necessary to process the exception.
- Every try block must be followed by at least one catch statement, otherwise compilation error will occur.

Example:

```
class X
{
public static void main(String args[])
{
int a=10;
int b=5;
int c=5;
int x,y;
try
{
x=a/(b-c);
}
catch(ArithmeticException e)
{
System.out.println("Division by zero");
}
y=a/(b+c);
System.out.println("y="+y);
}
}
```

Multiple catch statements

- It is possible to have more than one catch statement.

Syntax:

```
------------
------------
 try
{
 statement;
}
 catch(Exception-type1 e)
{
 statement;
}
catch(Exception-type2 e)
{
 statement;
}
 catch(Exception-typen e)
{
 statement;
}
-----------------
-----------------
```

Example:

```
 class  Y
{
Public static void main(String args[])
{
 int a[] = {5,10};
 int b = 5,x,y;
try
{
 int x=a[2]/b-a[1];
 }
 catch(ArithmeticException e)
{
System.out.println(“Division by zero”);
}
 catch(ArrayIndexOutBoundsException e)
{
System.out.println(“ Array  index  error”);
}

 y= a[1]/a[0];
 System.out.println(“y=“+y);
}
}
```

Using finally statement

- The finally statement is used to handle an exception that is not caught by any of the previous catch statement.
- The finally block can be used to handle any exception generated within a try block.

Syntax 1:

```
 try
{
 ------------
 ------------
}
 finally
{
 ------------
 ------------
}
```

Syntax 2:

```
 try
 {
        ----------
        ----------
        }
Catch(.....)
 {
        ----------
        ----------
}
Catch(.....)
 {
        ----------
        ----------
}
 finally
{
        ----------
        ----------
}
```

Throwing our own exceptions

- can do this by using the keyword throw.

examples

```
throw new ArithmeticException();
throw new NumberFormatException();
```

Using exceptions for debugging

- Exception handling mechanism can be used to hide errors from rest of the program.
- Exception handling mechanism may be effectively used to locate the type and place of errors.

Example

```
class ThrowDemo
{
static void demoproc()
{
try
{
throw new NullPointerException("demo");
}
catch
{
System.out.println("caught inside demoproc");
}
}
public static void main(String args[])
{
demoproc();
}
}
```

Applet Programming

Introduction

- Applets are small java programs that are primarily used in Internet computing.
- Applets can be transported from one computer to another computer.
- Applets can be run using the AppletViewer or any web browser that supports Java.
- Java applets can produce graphics, sounds and moving images.
- Local applets: an applet which is developed locally and stored in a local system.
- Remote applet: an applet which is developed by someone else and stored on a remote computer connected to the internet.

How applets differ from applications?

- Applets are not full – featured application programs.
- Applets usually written to accomplish a small task or a component of a task.
- Applets do not use the main() method.
- Applets can't be run independently. They are run from inside a web page using HTML tag.
- Applets can't read from or write to the files in the local computer.
- Applets can't communicate with other servers on the n/w.
- Applets can't run any program from the local computer.
- Applets are restricted from using libraries from other languages such as C or C++.

Preparing to write applets

The steps involved in developing and testing applet are:

1. Building an applet code (.java file)
2. Creating an executable applet (.class file)
3. Designing a web page using HTML tags.
4. Preparing <APPLET> tag.
5. Incorporating <APPLET> tag into the web page.
6. Creating HTML file.
7. Testing the applet code.

Building applet code

- Applet code uses the services of two classes, namely, Applet and Graphics from the Java class library.
- Applet class which is contained in the java.applet package provides life and behavior to the applet through its methods such as init(), start() and paint().
- Applet class maintains the life cycle of an applet.
- The paint() method displays the result of the applet code on the screen.
- Syntax of paint(): public void paint(Graphics g)
- Applet code imports java.awt package that contains Graphics class.

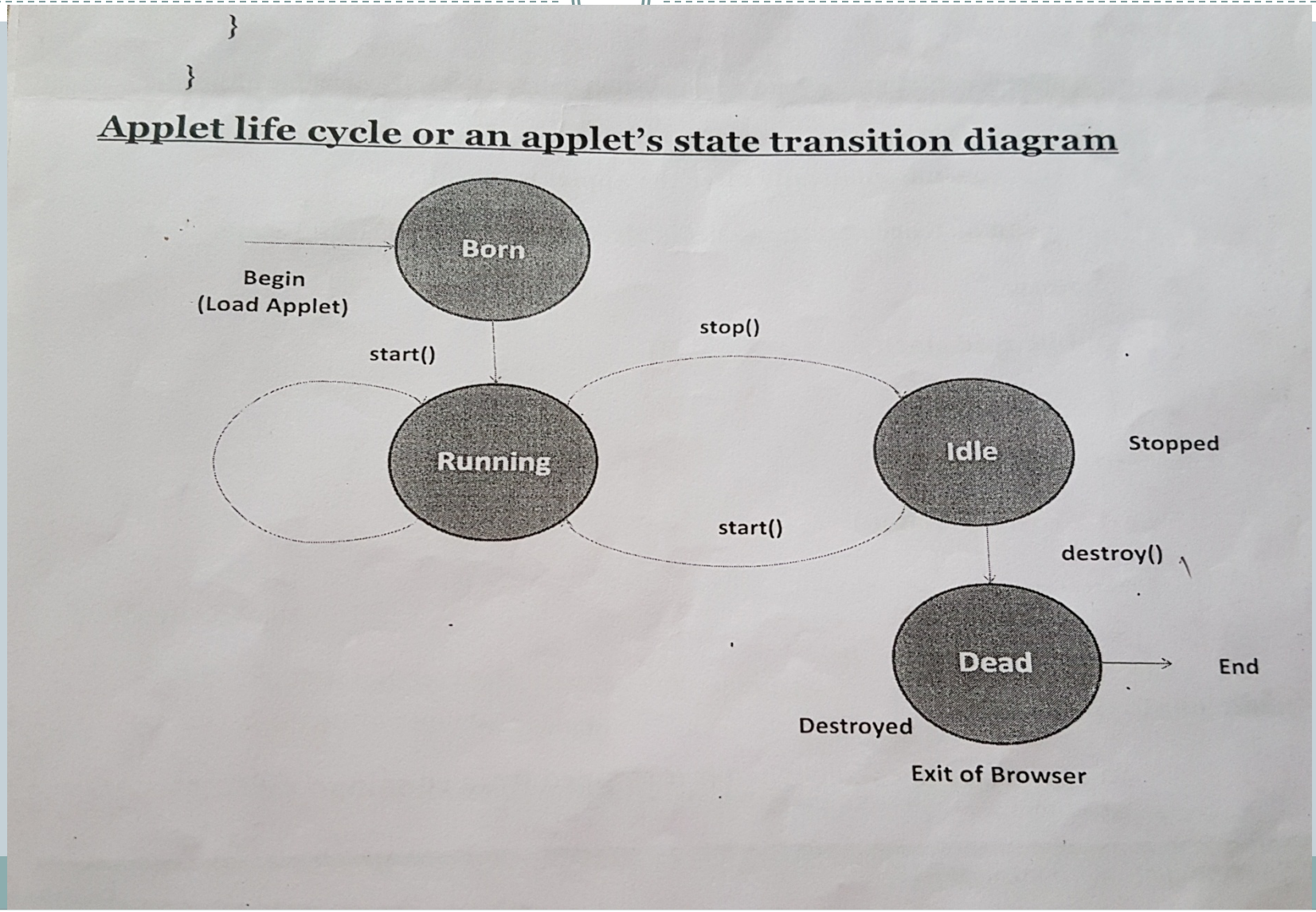
General format of an applet code

```
import java.awt.*;
import java.applet.*;
.....
public class appletclassname extends Applet
{
.....
public void paint(Graphics g)
{
.....
}
.....
}
```

Example

```
import java.awt.*;
import java.applet.*;
public class HelloJava extends Applet
{
public void paint(Graphics g)
{
g.drawString("Hello Java", 10,100);
}
}
```

Applet Life Cycle or an Applet's state transition diagram



✔ Initialization state

- Applet enters the initialization state when it is first loaded.
- This is achieved by calling the init() method of Applet class.

Syntax:

```
public void init()
{
.....
..... (Action)
.....
}
```

Running state

- Applet enters running state when system called the start() method of applet class.
- This occurs automatically after the applet is initialized.
- This can also occur if the applet is already in 'stopped' (idle) state.

Syntax:

```
public void start()
{
.....
..... (Action)
.....
}
```

✔ Idle or Stopped state

- An applet becomes idle when it is stopped from running.
- It occurs automatically when we leave the page containing the currently running applet.
- It can also occur by calling the stop() method explicitly.

Syntax:

```
public void stop()
{
.....
..... (Action)
.....
}
```

Dead state

- An applet is said to be dead when it is moved from memory.
- This occurs automatically by invoking the destroy() method when we quit the browser.

Syntax:

```
public void destroy()
{
.....
..... (Action)
.....
}
```

✔ Display state

- Applet moves to the display state whenever it has to perform some output operations on the screen.
- This occurs immediately after the applet enters into the running state.
- The paint() method is called to accomplish this task.

Syntax:

```
public void paint(Graphics g)
{
.....
..... (Display statements)
.....
}
```

```
import java.awt.*;
import java.applet.*;
public class HelloJavaParam extends Applet
 {
String str;
public void init()
{
 str=getParameter("string"); //getting parameter value
 if (str==null)
 str="are you?";
 str="How" + str; //using the value
 }
 public void paint(Graphics g)
 {
 g.drawString(str,10,100);
 }
}
```

```
<html>
<head>
<title> Welcome to Applet Codes </title>
</head>
<body>
<applet code = HelloJavaParam.class  width=400 height=200>
<param name="string" value="do you feel about applet???>
</applet>
</body>
</html>
```

Graphics Programming

- Java's Graphics class includes methods for drawing many different types of shapes, from simple lines to polygons to text in a variety of fonts.
- Lines and Rectangles

✔ The simplest shape we can draw with the Graphics class is a line.

✔ The drawLine() method takes 2 pair of coordinates, (x1, y1) and (x2, y2) as arguments and draws a line between them.

```
g.drawLine(10,10, 50,50);
```

- We can draw a rectangle using the drawRect() method.
- This method takes four arguments.
- The first two represent the X and Y coordinates of the top left corner of the rectangle, and the remaining two represent the width and the height of the rectangle.
- drawRect(10,60, 40,30);
- fillRect(), drawRoundRect() and fillRoundRect()

Example

```
import java.awt.*;
import java.applet.*;
public class LineRect extends Applet
{
public void paint(Graphics g)
{
g.drawLine(10, 10, 50, 50);
g.drawRect(10, 60, 40, 30);
g.fillRect(60, 10, 30, 80);
g.drawRoundRect(10, 100, 80, 50,10,10);
g.drawLine(100, 10, 230, 140);
g.drawLine(100, 10, 230, 140);
}
}
```

CIRCLES AND ELLIPSES

- drawOval() method can be used to draw a circle or an ellipse.
- drawOval() method takes four arguments: the first two represent the top left corner of the imaginary rectangle and the other two represent the width and height of the oval itself.
- drawOval() method draws outline of an oval, and the fillOval() method draws a solid oval.

```
public void paint(Graphics g)
{
g.drawOval(20, 20, 200, 120);
g.setcolor(Color.green);
g.fillOval(70,30,100,100);
}
```

- drawArc() designed to draw arcs taken six arguments. The first four are the same as the arguments for the drawOval() method and the last two represent the starting angle of the arc and the number of degrees (sweep angle) around the arc.

Example

```
import java.awt.*;
import java.applet.*;
public class Face extends Applet
{
public void paint(Graphics g)
{
g.drawOval(40, 40, 120, 150);
g.drawOval(57, 75, 30, 20);
g.drawOval(110, 75, 30, 20);
g.fillOval(68, 81, 10, 10);
g.fillOval(121,81,10,10);
g.drawOval(85, 100, 30,30);
g.fillArc(60,125,80,40,180,180);
g.drawOval(25, 92, 15, 30);
g.drawOval(160, 92, 15, 30);
}
}
```

- drawPolygon() method takes 3 arguments:
  - An array of integers containing x coordinates.
  - An array of integers containing y coordinates.
  - An array for the total number of points.

```
public void paint(Graphics g)
{
int xpoints [] = {10, 170,80,10};
int ypoints[]= {20, 40, 140, 20};
int npoints[]= xpoints.length;
g.drawPolygon (xpoints, ypoints, npoints);
}
```

Line Graphs

```
import java.awt.*;
import java.applet.*;
public class TableGraph extends Applet
{
int xpoints [] = {0, 60,120,180,240,300,360,400};
int ypoints[]= {400, 280, 220, 140,60,60,100,220};
int n= x.length;
public void paint(graphics g)
{
g.drawPolygon(x,y,n);
}
}
```

Using Control Loops in Applets

```
import java.awt.*;
import java.applet.*;
public class ControlLoop extends Applet
{
public void paint(Graphics g)
{
for(int i=0; i<=4;i++)
{
if(i%2 ==0)
g.drawOval(120,i*60+10, 50,50);
else
g.fillOval(120, i*60+10, 50,50);
}
}
}
```

Drawing Bar Charts

- The method getParameter() is used to fetch the data values from the HTML files.
- The method returns only string values and therefore we use the wrapper class method parseInt() to convert strings to integer values.

Stream

- A stream is a flow of information from a source to a destination. Some examples of elements that form a flow of communication between a source and a destination are:
- A keyboard(source) and a monitor(destination)
- A page of text on a monitor(source) and a file on a hard drive(destination)
- A file on a hard drive(source) and a monitor(destination)
- A keyboard(source) and a string(destination)

Classification of I/O Stream

- Classes in the I/O package are classified into 2 categories:-

✔ Byte Oriented Classes (Byte Streams)

- 2 types: InputStream and Outputstream

InputStream – super class of all the classes which are used to read data in the form of bytes.

OutputStream – super class of all the classes which are used to write data in the form of bytes.

✔ Character Oriented Classes(Character Streams)

- 2 types: Reader and Writer

Reader class – super class of all the classes which are used to read data in the form of characters.

Writer class – super class of all the classes which are used to write data in the form of characters.

✔ Object is the super class for all the above streams.

Type Casting

- Type casting is when you assign a value of one primitive data type to another type.
- In Java, there are two types of casting:
- Widening Casting (automatically) - converting a smaller type to a larger type size
- byte -> short -> char -> int -> long -> float -> double
- Narrowing Casting (manually) - converting a larger type to a smaller size type
- double -> float -> long -> int -> char -> short -> byte

- Widening casting is done automatically when passing a smaller size type to a larger size type.

```
public class MyClass
{
  public static void main(String[] args)
 {
    int myInt = 9;
    double myDouble = myInt; // Automatic casting: int to double

    System.out.println(myInt);      // Outputs 9
    System.out.println(myDouble);   // Outputs 9.0
  }
}
```

Narrowing Casting

- Narrowing casting must be done manually by placing the type in parentheses in front of the value.

```
public class MyClass {
  public static void main(String[] args) {
    double myDouble = 9.78;
    int myInt = (int) myDouble; // Manual casting: double to
int


    System.out.println(myDouble);   // Outputs 9.78
    System.out.println(myInt);      // Outputs 9
  }
}
```