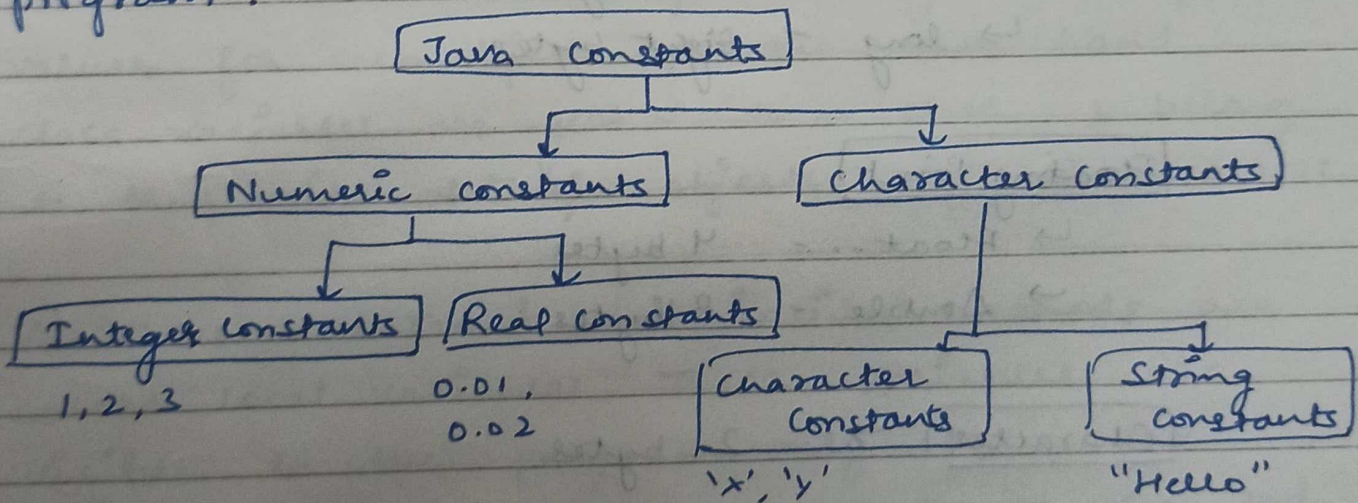
Unit - 2

Java Fundamentals, Data Types & Literal Variables, Wrapper Classes, Arrays, Arithmetic Operators, Logical Operators, Control of Flow, Classes & Instances, Class Member Modifiers, Anonymous Inner Class Interfaces and Abstract Classes, Inheritance, throw and throws clauses, User defined Exceptions, The String Buffer class, tokenizer, applets, life cycle of applet, and Security concepts.

Constants

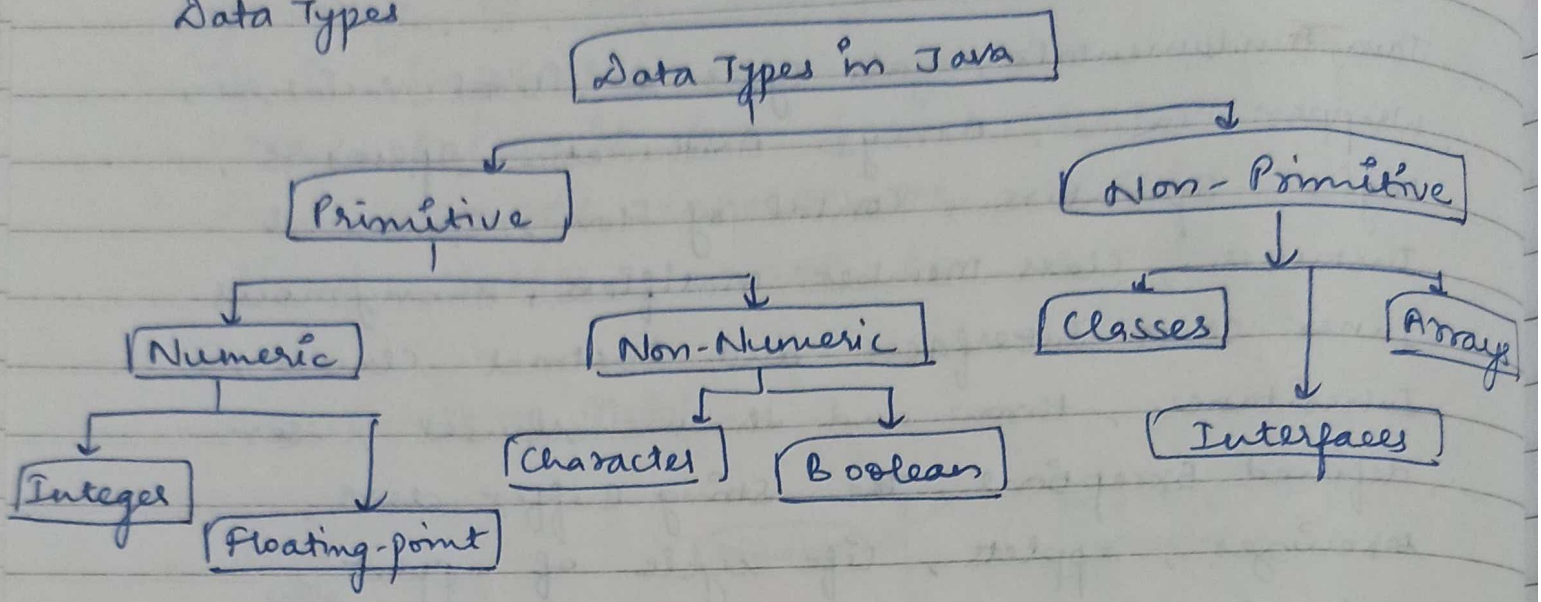
Constants in Java refer to fixed values that do not change during the execution of a program.



Variables

A variable is an identifier that denotes a storage location used to store a data values.

Data Types



Integer

- Byte - one byte - -128 to 127
- Short - Two bytes - -32768 to 32767
- int - Four bytes
- long - Eight bytes

Floating Point

- Float - 4 bytes
- Double - 8 bytes

Character → 2 bytes

Boolean → 1 bit True or False.

Declaration of variables

```
type variable1;
int count;
float x, y;
double pi;
```

Assignment Statement

variableName = value;

```
yes = 'x';
z = 0;
```

Scope of variables

- instance variables
- class variables
- local variables

Instance & class variables are declared inside a class. Instance variables are created when the objects are instantiated & therefore they are associated with the objects. They take different values for each object. On the other hand, class variables are global to a class & belong to the entire set of objects that class creates.

Variables declared & used inside methods are called local variables. These variables are visible to the program only from the beginning of its program block to the end of the program block.

Type casting

type var1 = (type) var2;

```
int m = 50;
byte n = (byte) m;
```

Operators.

An operator is an symbol that tells the computer to perform certain mathematical or logical manipulations.

1. Arithmetic operators.

+ - Addition
- - Subtraction
* - Multiplication
/ - Division
% - Modulo Division (Remainder)

Integer Arithmetic - Both operands integer
Real Arithmetic - Both operands real
Mixed-Mode " = One Real, One Integer.

2. Relational Operators.

< less than
<= less than or equal to
> greater than
>= greater than or equal to
== equal to
!= is not equal to

3. logical operators

&& logical AND
|| logical OR
! logical NOT

op1	op2	op1 && op2	op1 || op2
T	T	T	T
T	F	F	T
F	T	F	T
F	F	F	F

4. Assignment operators

v op = exp;

x += 3;

a = a + 1	a += 1
	a -= 1
a = a * (n+1)	a *= n+1
a = a / (n+1)	a /= n+1
a = a % b	a %= b

5. Increment & Decrement operators.

++

--

m = 5;
y = ++m;] y = 6

m = 5;
y = m++;] m = 6, y = 5

6. Conditional operators.

exp1 ? exp2 : exp3

```
a = 10;
b = 15;
x = (a > b) ? a : b;
```

7. Bitwise operators

&	bitwise AND
!	bitwise OR
^	bitwise exclusive OR
~	one's complement
<<	shift left
>>	shift right
>>>	shift right with zero fill.

8. Special operators.

Instance of operators

The instanceof is an object reference operator & returns true if the object on the left-hand side is an instance of the class given on the right-hand side

person instanceof student

is true if the object person belongs to the class student, otherwise it is false.

Dot operator (.)

is used to access the instance variables & methods of class objects.

person1. age

person1. salary()

Arithmetic Expressions

variable = expressions;

x = a * b - c;

y = b/c * a;

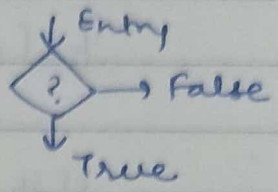
z = a-b/c+d;

BODMAS

() of Di Mu Add Sub

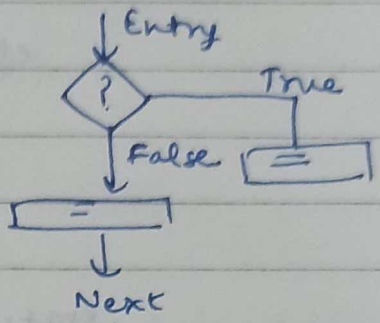
Decision Making with If Statement

```
if (test expression)
```



Simple if statement

```
if (test expression)
 {=
 }
statement-x;
```



The if-else statement

```
if (test-expression)
 {=
 }
else
 {=
 }
=
```

Nesting of if-else statements

```
if (test condition)
 {
    if (test condition)
      {=
      }
    else
      {=
      }
 }
else
 {=
 }
=
```

The Else if ladder

```
if (condition1)
    statement-1;
else if (condition2)
    ==
    else if (condition3)
        ==
        else
            ==
        statement-x;
```

Switch Statement

```
switch (expression)
{ case value-1:
        block-1
        break;
  case value-2:
        block-1
        break;
  ==
  default:
        default-block
        break;
}
```

while Statement

```
Initialization
while (test condition)
{
    body of the loop
}
```

do statement

```
Initialization;
do
{
  Body
}
while (test condition);
```

for statement

```
for (initialization; test condition ; increment)
{
=
}
```

Classes, Objects & Method

Defining a class

```
class classname [extends superclass name]
{
    [fields declarations ;]
    [methods declarations;]
}
```

Field declaration

```
int length, width;
```

Method declaration

```
type methodname ( parameter-list)
{
  =
}
```

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

Creating objects

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

or

```
Rectangle rect1 = new Rectangle();
```

Accessing class members.

```
rect1.length = 15;
rect1.width = 10;

rect1.getData (15, 10);
```

Constructors.

Constructors have the same name as the class itself. Secondly, they do not specify a return type, not even void. This is because they returns the instance of the class itself.

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

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

Method overloading

In Java, it is possible to create methods that have the same name, but different parameter lists and different definitions. This is called method overloading. Method overloading is used when objects are required to perform similar tasks but using different input parameters.

```
class Room
  {
    float length;
    float breadth;
    Room (float x, float y)
    {
      length = x;
      breadth = y;
    }
    Room (float x)
    {
      length = breadth = x;
    }
    int area ()
    {
      return (length * breadth);
    }
  }
```

Static Members

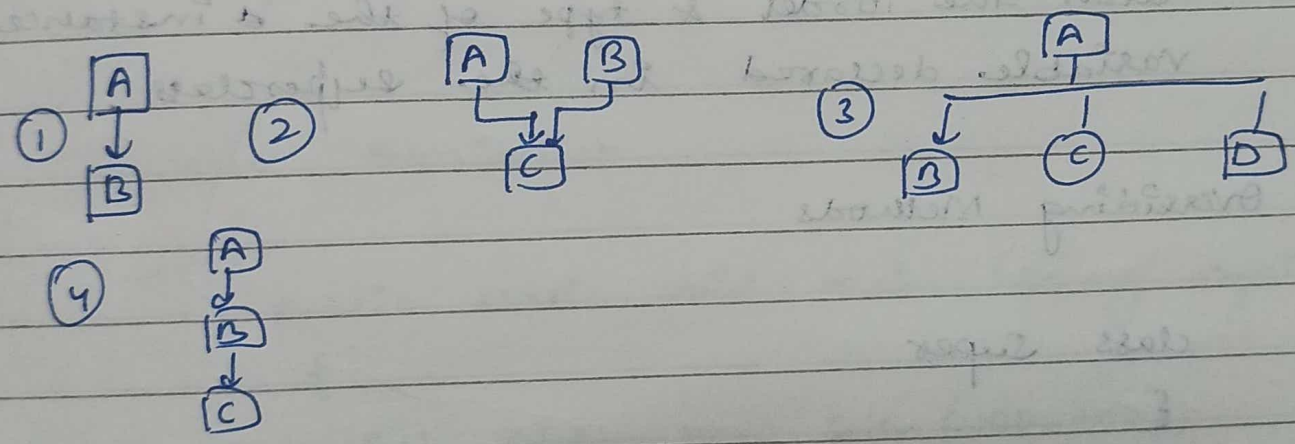
Static members are associated with the class itself rather than individual objects. The static variables & static methods are often referred to as class variables and class methods in order to distinguish them from instance variables & instance methods.

Inheritance

The mechanism of deriving a new class from an old one is called inheritance. The old class is known as the base class & / super / parent. & the new one is called as subclass / derived / child.

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

1. Single inheritance (only one super class)
2. Multiple (several super classes)
3. Hierarchical (one super class, many subclasses)
4. Multilevel (Derived from a derived class)



Defining a subclass

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

Subclass constructor

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

- 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 & type of the instance variable declared in the superclass.

Overriding Methods

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

```
class Sub extends Super
  {
    int y;
      Sub ( int x , int y)
       {
          super (x);
          this.y = y;
       }
      void display ( )
       {
         System.out.println ( "super x = " + x);
         System.out.println ( "Sub y = " + y);
       }
  }

class OverrideTest
 {
   public static void main (String args[])
    {
     Sub s1 = new Sub (100, 200);
      s1.display( );
    }
 }
```

O/P

super x = 100

Sub y = 200

Final Variables

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

```
final int SIZE = 100;
```

Final classes

A class that cannot be subclassed is called a final class.

```
final class AClass { . . . }
```

Finalize Methods

Java supports a concept called finalization, which is just opposite to initialization. Similarly, 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++.

Abstract Methods & Classes.

A class which is declared as abstract is known as an abstract class. It can have abstract & non-abstract methods. It needs to be extended & its method implemented. It cannot be instantiated.

Dynamic Method Dispatch

Dynamic Method Dispatch is an important mechanism in Java that is used to implement runtime polymorphism. In this method, method overriding is resolved at runtime instead of compile time. That means, the choice of the version of the overridden method to be executed in response to a method call is done at runtime.

```
class Super
  {
    public void method()
    {
      System.out.println("Method Super");
    }
  }
class Sub extends Super
  {
    public void method()
    {
      System.out.println("Method Sub");
    }
  }
```

```
class dyn-dis
{
  public static void main (String args[])
  {
    Super A = new Sub();
      // Sub's object reference assigned super
         type reference variable
    A.method(); // Sub's version of method will
                   be called at runtime.
  }
}
```

O/P

Method Sub

Visibility control

	Public	Protected	friendly (default)	private protected	private
Same class	Yes	Yes	Yes	Yes	Yes
Subclass in same package	Yes	Yes	Yes	Yes	No
Other classes in same package	Yes	Yes	Yes	No	No
Subclass in other package	Yes	Yes	No	(Yes)	No
Non-subclasses in other packages	Yes	No	No	No	No

1. Use public if the field is to be visible everywhere
2. Use protected if the field is to be visible everywhere in the current package & also subclasses in other packages.
3. Use "default" if the field is to be visible everywhere in the current package only.
4. Use private protected if the field is to be visible only in subclasses, regardless of package
5. Use private if the field is not to be visible anywhere except in its own class.

One Dimensional Arrays

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

number [0] = 35;
number [1] = 40;
number [2] = 20;
number [3] = 57;
number [4] = 19;
```

Creating an array

1. Declaring the array
2. Creating memory locations
3. Putting values into the memory locations.

Creation of array

```
arrayname = new type [size];
```

Initialization

```
arrayname [subscript] = value;
```

Array length

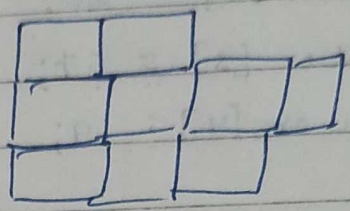
```
int asize = a.length;
```

Two-Dimensional Array

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

Variable Size Arrays

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



Strings

```
char charArray[] = new char[4];
charArray[0] = 'J';
charArray[1] = 'a';
charArray[2] = 'v';
charArray[3] = 'a';

String stringName;
stringName = new String("string");
```

String Arrays

```
String itemArray[] = new String[3];
```

String Methods

```
s1.toLowerCase;
s1.toUpperCase
s1.replace('x', 'y');
    trim();
s1.equals(s2)
s1.equalsIgnoreCase(s2)
    length();
```

charAt(n)
compareTo
concat
substring
index

StringBuffer Class

StringBuffer is a peer class of String. While String creates strings of fixed-length, StringBuffer creates strings of flexible length that can be modified in terms of both length & content. We can insert characters & substrings in the middle of a string or append another string to an end.

commonly used StringBuffer methods

sl. setCharAt(n,'x') - Modifies the nth char to x.

sl. append(s2) - Appends the string s2 to s1 at the end.

sl. insert(n,s2) - inserts the string s2 at the position n of the string s1.

sl. setLength(n) - sets the length of the string.
If n < s1.length() s1 is truncated.
If n > s1.length() zeros are added to s1.

Interfaces : Multiple Inheritance

Java doesn't support multiple inheritance. That is, the classes in Java cannot have more than one superclass.

class A extends B extends C ✗ Not Possible

An interface is basically a kind of class. Like classes, interfaces contain methods & variables but with a major difference. The difference is that interfaces define only abstract methods & final fields. This means that interfaces do not specify any code to implement these methods & data fields contain only constants. Therefore, it is the responsibility of the class that implements an interface to define the code for implementation of these methods.

```
interface InterfaceName
{
    variables declaration;
    methods declaration;
}
```

```
interface Item
{
    static final int code = 1001;
    static final String name = "Fan";
    void display();
}
```

Extending Interfaces

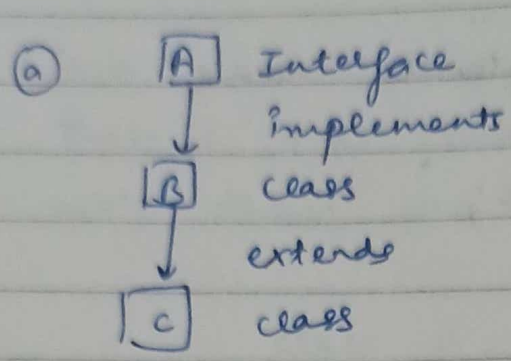
```
Interface name2 extends name1
{
    body of name2;
}
```

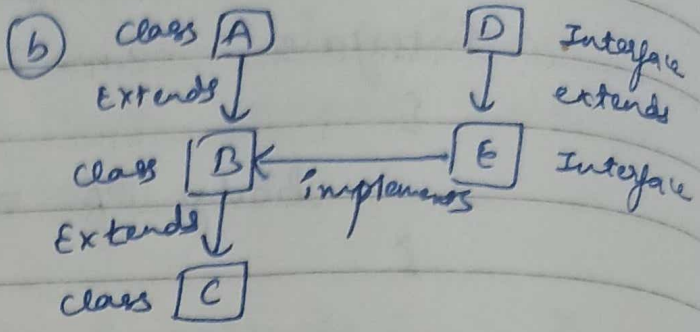
```
interface ItemConstants
{
    int code = 1001;
    String name = "fan";
}
interface ItemMethods
{
    void display();
}
interface Item extends ItemConstants, ItemMethods
{
}
```

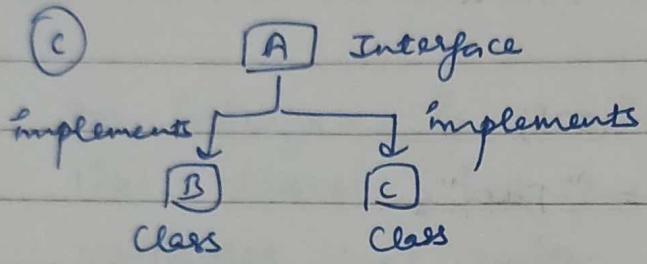
Implementing Interfaces

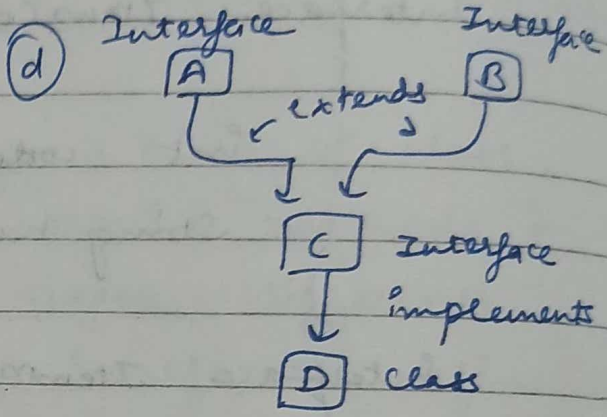
```
class classname extends superclass
    implements interface1, interface2
{
}
```

Various forms of Interface Implementation









Implementing Interfaces code

```
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 circ = new Circle ();
        Area area;                    // Interface object
        area = rect;                  // area refers to
                                      rect object.
        System.out.println (" Area of rectange = "
                            area.compute (10,20));

        area = circ;
        System.out.println (" Area of circle = "
                            + area.compute (10,0));
    }
}
```

O/P

Area of rectangle = 200
Area of circle = 314

Packages

Packages are Java's way of grouping a variety of classes & / or interfaces together. The grouping is usually done according to functionality. In fact, packages act as 'containers' for classes.

Java packages are classified into two types -

1. Java API packages
2. User defined packages.

Java API Packages

Java API provides a large number of classes grouped into different packages according to functionality.

- lang
- util
- io
- awt
- net
- applet

Using Java packages

```
import java.awt.Color;
import java.awt.*;
```

Naming Conventions

```
double y = java.lang.Math.sqrt(x);
```

↑ package name ↑ class name ↑ method name

Creating Packages

```
package firstPackage;
public class FirstClass
{
}
```

Accessing a Package

```
import packagename.*;
```

Using a package

Example code

```
package package1;
public class ClassA
{
    public void displayA()
    {
        System.out.println("Class A");
    }
}
```

Source file should be named as ClassA.java & stored in the subdirectory package1.

```
import package1.classA;
class PackageTest1
{
 public static void main (String args[])
  {
    class.A objectA = new classA();
     objectA.displayA();
  }
}
```

The source file should be saved as PackageTest1.java. The source file & the compiled file would be saved in the directory of which package1 was a subdirectory.

```
package package2;
public class classB
{
  protected int m = 10;
   public void displayB()
    {
     System.out.println("class B");
     System.out.println("m = " + m);
    }
}
```

Source file & the compiled file of this package are located in the subdirectory package2.

Importing classes from other packages.

```
import package1.ClassA;
import package2.*;
class PackageTest2
{
  public static void main(String args[])
  {
    ClassA objectA = new ClassA();
    ClassB objectB = new ClassB();
    objectA.displayA();
    objectB.displayB();
  }
}
```

O/P

```
Class A
Class B
m = 10
```

Hiding classes

```
package p1;
public class X
{
  //
}
class Y                    // not public, hidden
{
  ==
}
```

Managing Errors and Exceptions

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
2. Run-time errors

Compile-Time Errors

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

```
class Error1
{
  public static void main (String args[])
  {
    System.out.println ("Hello Java")
                                    //Missing;
  }
}
```

1. Missing semicolons
2. Missing brackets
3. Misspelling of keywords
4. Missing double quotes in strings
5. Use of undeclared variables
6. Incompatible types in assignments
7. Use of = in place to == operator.

Run-time Errors

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

1. Dividing an integer by zero.
2. Accessing an element that is out of the bounds of an array.
3. Trying to store a value into an array of an incompatible class or type.
4. Trying to cast an instance of a class to one of its subclasses.
5. Passing a parameter that is not in a valid range.
6. Trying to illegally change the state of a thread.
7. Attempting to use a negative size for an array.

```
class Error2
{
  public static void main(String args[])
  {
    int a = 10;
    int b = 5;
    int c = 5;
    int x = a / (b-c);        // Division by zero
    System.out.println("x = " + x);
    int y = a / (b+c);
    System.out.println("y = " + y);
  }
}
```

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 & throws it (i.e. informs us that an error has occurred).

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

ArithmeticException
ArrayIndexOutOfBoundsException
ArrayStoreException
FileNotFoundException
IOException
NullPointerException
OutOfMemoryException
StackOverFlowException

Types :-

Checked Exception - These exceptions are explicitly handled in the code itself with the help of try-catch blocks. Checked exceptions are extended from the java.lang.Exception class.

2. Unchecked Exceptions - These exceptions are not essentially handled in the program code; instead the JVM handles such exceptions. Unchecked exceptions are extended from the java.lang.RuntimeException class.

Syntax of Exception Handling code

```
try
{
  ==          // generates an exception
}
catch (Exception-type e)
{
  ==          // processes the exception.
}
```

```
class Error3
{
  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

```
try
{
}
catch (Exception-type-1 e)
{
}
catch (Exception-type-2 e)
{
}
=
catch(Exception-type-n e)
{
=
}
```

Using finally statement

Java supports another statement known as finally statement that can be used to handle an exception that is not caught by any of the previous catch statements. finally block can be used to handle any exception generated within a try block. It may be added immediately after the try block or after the last catch block.

When the finally block is defined, ~~that~~ this is guaranteed to execute.

```
try
{
=
}
finally
{=
}
```

```
try
{=}
catch()
{=
}
catch()
{=
}
finally
{=
}
```

Throwing our own exceptions

throw new Throwable subclass;

```
import java.lang.Exception;
class MyException extends Exception
{
  MyException(String message)
  {
    super(message);
  }
}
class TestMyException
{
  public static void main(String args[])
  {
    int x=5, y=1000;
    try
    {
      float z = (float)x / (float) y;
```

```
        if (z < 0.01)
        {
        throw new MyException("No. is too small");
        }
    }
    catch( myexception e)
    {
     System.out.println("caught exception");
     System.out.println(e.getMessage());
    }
    finally
    {
     System.out.println("I am always here");
    }
   }
}
```

Use of throws

There could be situations where there is a possibility that a method might throw certain kinds of exceptions but there is no exception handling mechanism prevalent within the method. In such a case, it is important that the method called is intimated explicitly that certain types of exceptions could be expected from the called method, & the caller must get prepared with some catching mechanism to deal with it.

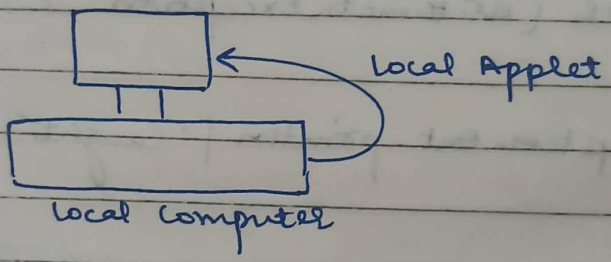
```
class Examplethrows
{
static void divide_m() throws ArithmeticException
{
 int x = 22, y = 0, z;
 z = x / y;
}
public static void main (String args[])
 {
  try
  {
   divide_m();
  }
  catch (ArithmeticException e)
  {
  System.out.println("Caught the exception" + e);
  }
 }
}
```

Applet Programming

Applets are small Java programs that are primarily used in Internet computing. It can perform arithmetic operations, display graphics, play sounds, accept user input, create animation, & play interactive games.

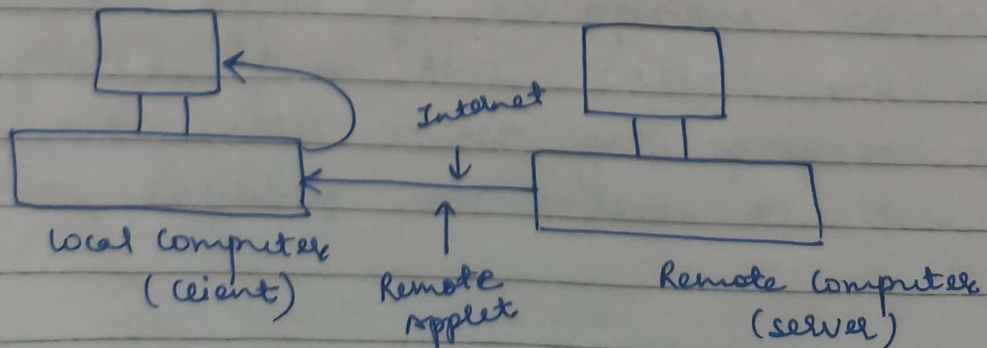
Local Applet

An applet developed locally & stored in a local system is known as a local applet. When a web page is trying to find a local applet, it doesnot need to use the internet.



Remote Applet

A remote applet is that which is developed by someone else & stored on a remote computer connected to the internet. If our system is connected to the Internet, we can download the remote applet onto our system via at the Internet & run it.



The steps involved in developing and testing in 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 webpage.
6. Creating HTML file.
7. Testing the applet code.

Example

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

The paint() method of the Applet class, when it is called, actually displays the result of the applet code on the screen.

Save file as
HelloJava.java

compile
javac HelloJava.java.

```
<HTML>
  <HEAD>
    <TITLE>
      Welcome to Java Applets
    </TITLE>
    <BO
  </HEAD>
  <BODY>
    <CENTER>
      <H1>
        Welcome
      </H1>
    </CENTER>
    <BR>
    <CENTER>
    <APPLET
      CODE = "HelloJava.class"
      Height = 400
      Width = 400 >
    </APPLET>
    </CENTER>
    </BODY>
  </HTML>
```

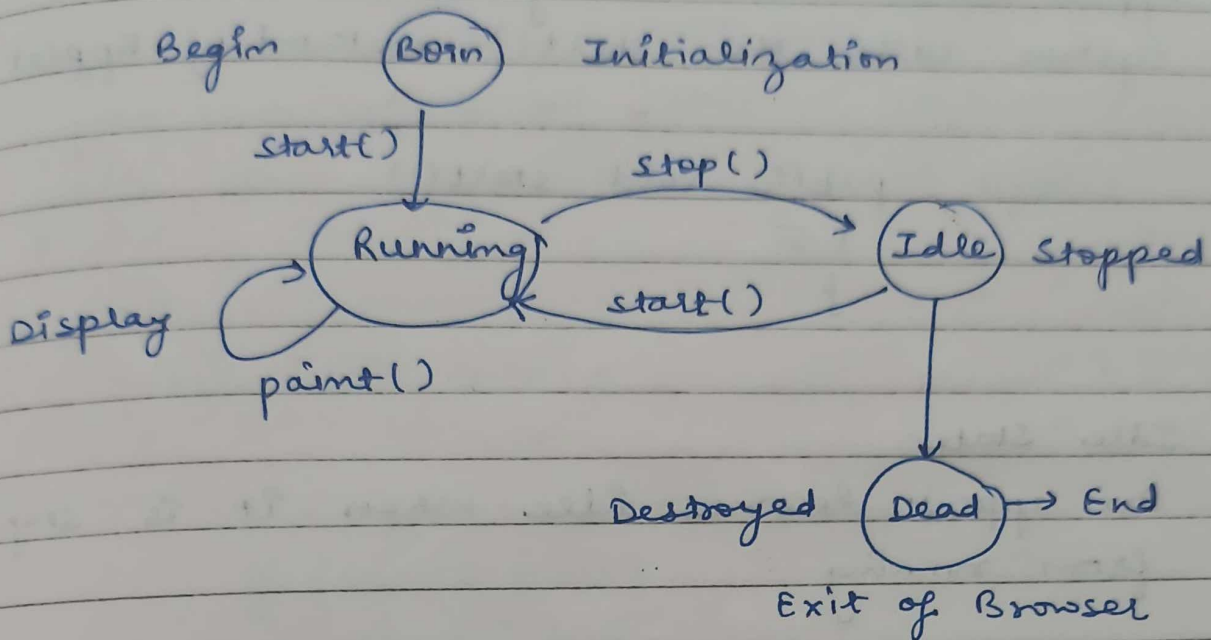
Save as HelloJava.html

appletviewer HelloJava.html

Applet is a subclass of the Panel class.

java.lang.Object
  \ java.awt.Component
    \ java.awt.Container
      \ java.awt.Panel
        \ java.applet.Applet

Applet Life Cycle



The applet state includes:

1. Born state
2. Running state
3. Idle State
4. Dead state

Initialization State

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

```
public void init()
{
==
}
```

Running State

Applet enters the running state when the system calls the start() method of Applet class

```
public void start()
{ ≡
}
```

Idle State

An applet becomes idle when it is stopped from running.

```
public void stop()
{ ≡
}
```

Dead State

An applet is said to be dead when it is removed from memory.

```
public void destroy()
{ ≡
}
```

Display State

The paint() method performs output operations on the screen.

Passing Parameters to Applets

```
import java.awt.*;
import java.applet.*;
public class HelloJavaParam extends Applet
{
   String str;
   public void init()
    {
       str = getParameter("string");
       if (str == null)
           str = "Java";
       str = "Hello" + str;
    }
   public void paint (Graphics g)
    {
       g.drawString (str, 10, 100);
    }
}
```

```
<HTML>
 <HEAD>
  <TITLE> Welcome </TITLE>
 </HEAD>
 <BODY>
  <APPLET CODE = HelloJavaParam.class
          WIDTH = 400
          HEIGHT = 200 >
        <PARAM NAME = "string"
               VALUE = "Applet!" >
  </APPLET>
 </BODY>
</HTML>
```

Aligning the Display

ALIGN = RIGHT

LEFT, TOP, MIDDLE, BOTTOM

Displaying Numerical Values

```
import java.awt.*;
import java.applet.*;
public class NumValues extends Applet
{
  public void paint (Graphics g)
  {
    int value1 = 10;
    int value2 = 20;
    int sum = value1 + value2;
    String s = "sum:" + String.valueOf (sum);
    g.drawString (s, 100, 100);
  }
}
```

```
<HTML>
<APPLET>
        CODE = NumValues.class
        WIDTH = 300
        HEIGHT = 300 >
</APPLET>
</HTML>
```

Getting Input from the User

```
import java.awt.*;
import java.applet.*;
public class UserIn extends Applet
 {
  Textfield text1, text2;
  public void init()
   {
    text1 = new Textfield (8);
    text2 = new Textfield (8);
    add (text1);
    add (text2);
    text1. setText ("0");
    text2. setText ("0");
   }
  public void paint( Graphics g)
   {
    int x=0, y=0, z=0;
    String s1, s2, s;
    g.drawString ("Input a number in each
                   box", 10, 50);
    try
    {
      s1 = text1. getText ();
       x = Integer. parseInt (s1);
      s2 = text2. getText ();
       y = Integer. parseInt (s2);
    }
    catch (Exception ex) { }
    z = x + y;
     s = String. valueOf (z);
```

```
    g.drawString ("The Sum is:", 10, 75);
    g.drawString (s, 100, 75);
  }
  public Boolean action (Event event,
                         Object object)
    {
      repaint();
      return true;
    }
}
```

```
<HTML>
  <APPLET
      CODE = UserIn.class
      WIDTH = 300
      HEIGHT = 200 >
  </applet>
</HTML>
```

Event Handling

Event Handling is a mechanism that is used to handle events generated by applets. An event could be the occurrence of any activity such as mouse click or a key press. Some of the key events in Java are -

1. ActionEvent is triggered whenever a user interface element is activated, such as selection of a menu item.
2. ItemEvent is triggered at the selection or deselection of an itemized or list element, such as check box.

3. TextEvent is triggered when a textfield is modified.
4. WindowEvent is triggered whenever a window-related operation is performed, such as closing or activating a window.
5. KeyEvent is triggered whenever a key is pressed on the keyboard.

Keyboard event in an applet

```
<html>
<body>
<applet code = eg-event.class
        width = 200
        height = 200>
</applet>
</body>
</html>
```

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

public class eg-event extends Appet
                implements KeyListener
{
  public void init()
  {
     addKeyListener(this);
  }
  public void KeyTyped(KeyEvent KB){}
  public void KeyReleased(KeyEvent KB)
  {
  showstatus(" Key on the keyboard is
                released");
  }
```

```
public void KeyPressed ( KeyEvent KB)
 {
  showStatus ("A Key on the Keyboard is
                pressed");
 }
Font f1 = new Font ("Courier New",
                    FONT.BOLD , 20);
  public void paint ( Graphics GA)
   {
    GA. setFont (f1);
    GA. setColor ( Color . blue);
    GA. drawString ("This applet sense the
    up / down motion of keys", 20, 120);
   }
}
```