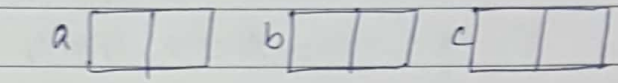
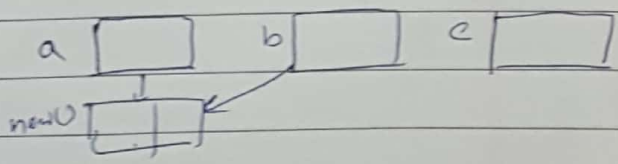
~~01 October~~ 04/10/2018

#

```
# include <iostream>      //C++ program of copy object
using namespace std;
Class K { public : int x, y; };
main ()
{ ~~K = a, b, c;~~
  K  a, b, c;
  a.x = 7 ; a.y = 9;
  Cout << a.x << a.y;  → 7, 9
  b = a ;
Cout << b.x << b.y;  → 7, 9
  b.x = 6;
Cout << b.x << b.y ;  → 6, 9
Cout << a.x << a.y ;  → 7, 9
```




```
Class K { int x, y; }      // Java program of Copy object
class m {
   public static void main ( string t[] ) {
      K a, b, c;
      a = new K();
      b = new K();
      a.x = 7; a.y = 9;
      System.out.println ( a.x + "  " + a.y); → 7, 9
      b = a
      System.out.println ( b.x + "  " + b.y); → 7 9
      b.x = 6;
      System.out.println ( b.x + "  " + b.y ); → 6 9
      System.out.println ( a.x + "  " + a.y); → 6 9
```



Scanned by CamScanner

```
class k { int
    int x, y;
    void pt() { System.out.println(x+" "+y); }
    void cpy(k u) { x=u.x, y=u.y; }
    static k cpy(k u){return u; }
    static k cpy1(k u){ k m = new k();
                        m.x = u.x; m.y = u.y;
                        return m; }
}
class m {
    public static void main(String args[]) {
        k a,b,c;
        a = new k();
        b = new k();
        a.x = 1; a.y = 2; a.pt() → 1 2
        b.cpy(a) // or b = k.cpy(a)
                 // or b = k.cpy1(a);
        b.pt(); → 1 2
        b.x = 3;
        b.pt(); → 3 2
        a.pt(); → 1 2
    }
}
```

	c.pt	b.pt	b.pt	a.pt	c.pt
b.cpy(a)	1 2	1 2	3 2	1 2	3 2
b=k.cpy(a)	0 0	1 2	3 2	3 2	0 0
b=kcpy1(a)	0 0	1 2	3 2	1 2	0 0

Scanned by CamScanner

03/10/2018

function overloading The FO is defined is in the following way:

(i) Same function name can be used for different purposes The Compiler resolves which function is to be used by looking at its Signature which is defined as the function name along with their parameters, their types, order and number.

(ii) function overloading do not identify the types of parameters and their returns types.

EX:-

```
int fixeddeposit(float x)
float fixeddepsit(int x)
```

(iii) Some function may appears in main( ) and in Some of Class of the program

E.X- Write a Simple program add two numbers via function overloading.

```
# include<iostream>
    float add(float x, float y)
class number{
        private: int i, int j;
        public: number( )
                {
                    i=0;
                    j=0;
                }
        int add(int i, int j);
        };
int main( ){
    int K, m=3, n=4, float Z;
    a=35, b=4.5, number N;
    K = N. add(m,n);
    Z = add(a,b);
    return 0;
}
```

Scanned by CamScanner

Ex:- Use function overloading to find the square of an integers and a floating point quantity.

Soln:-
```
#include<iostream>
int square(int x)
{
  Cout << "square of an integer" <<x << "is";
     return (x*x);
}
double square(double y){
Cout <<" square of double" <<y<< "is" <<
  return (y*y);
}
int main(){
Cout <<square(5) << square(1.5) << endl;
}
```

Ex:- Write an operator overloading to overload the operator ⊕ to add strings.

Soln:-
```
# include <iostream>
# include <string>
 Const int Size = 80;
  class String {
    Private: char s[size];
public String()
{ strcpy(s, "");} }
  String (char ss[]){
    strcpy (s, ss); }
String operator + (string ss){
 if ((strlen(s) + strlen(ss))<size)
 { String temp;
    strcpy (temp.s)
    strcpy (temp.s, ss)
 }  return (temp);
```

Scanned by CamScanner

```
else {
    cout << "string overflow";
    return;
    }
};
```

Arrays :-

~~(i) Entire array~~

~~(ii)~~ This is a sequence of characters where each sub-script has a value. It also has following properties:

(i) Entire array can be input/output

(ii) overloaded sub-script operator for non-constant array be reference return to int type.

```
Ex- int &Array:: operator [] (int subscript) {
    if (subscript < 0 || subscript >= size)
    { cerr << subscript << " out of range" << endl;
      exit(1);
    }
    return ptr[subscript];
}
```

3. Overloaded subscript operator for constant array by reference return int type

```
Ex:- int Array :: operator []
        (int subscript) const {
    if (subscript << " out of range" << endl;
    if (subscript < 0 || subscript >= size) {
        cerr << subscript << " out of range" << endl;
        exit(1);
    } return ptr[subscript];
}
```

Scanned by CamScanner

Limitions of Arrays:

1. Subscript checking is not possible.
2. Range of n-dimensional array varies from 0 to n-1 and other rangs are not possible.
3. When we pass an array to a function both array name and size of the array is to be passed.
4. Input/output of array is done by individual elements of the array.
5. Comparison of arrays by "==" or relational operators

Ex:- Example related to passing an array as input/output

```
#include <iostream>
istream & operator >> (istream & input, Array &a)
{
    for(int i=0; i<a.size ; i++)
        input >> a.ptr[i];
    return input;
}

Ostream & operator << (Ostream & output, Const Array &a)
{
    for(int i=0; i<a.size ; i++)
    Output << setw(12) << a.ptr[i];
    if((i+1)%4) == 0)
        output << endl;
    if (i%.4 != 0 )
        return Output;
}
```

Scanned by CamScanner

```
int main ( ) {
    Array Integers1 (7);
    Array integers2( );
    Cout << integers1. getSize( ) << integer1 << endl;
    Cout << integer1. getSize ( ) << integer2 << endl;
    Cin >> integer1 >> integers2;
    if( integers1 != integers2)
        Cout << " not equal" << endl;

Array integers3 (integers1);
    if (integers
        Cout << integers 3
        integers1 = integers2;
        Cout << integers1;
    }
```

Inheritance :- A user creates a new class from the existing class by inheriting the data and the behavior of the existing class, changes the the behavior of already existing behavior and adds a new behavior to it.

Advantages :- 1. Software development becomes Simplified

2. The new class(es) do not need to specify the predefined operations / functions in the present class.

3. Declaration of data structure is not necessary in child class

4. The newest class is called the derived class and the existing class is called the base class.

Disadvantages :- (i) It's a one way relationship.

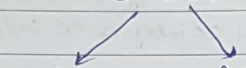
(ii) operator & functions overloading are not inherited

(iii) Not all function (friend function) are inherited.

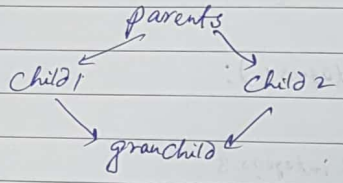
Scanned by CamScanner

(iv) Except very simple type of Constructors, all Constructors are not inherited.

Ex- Student



one label inheritance

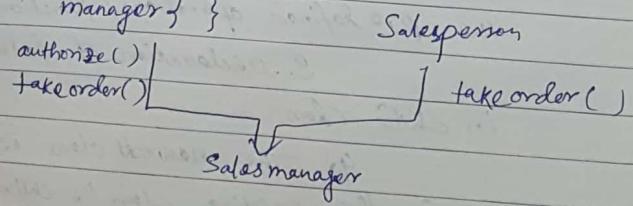


⇒ Multi label inheritance.

04/10/2018

problem with multi-label inheritance.

Consider the example of Sales manager. A Sales manager can take orders from a Customer (as a sales person) and authorise order (like a manager would do). The Sales managess class would inherit the authorize( ) from manager class and ~~take~~ takeorder( ) takeorder( ) from Sales person class,



Scanned by CamScanner

Manager

↓

Sales Manager

↓

Sales person

friend function :- If a non-member functions of a class wants to access its private data then the class should be declared as friend of the class having non-member functions.

Ex:-

```
#include <iostream>
class beta;
class alpha {
        private: int data;
        public : alpha ()
                { data = 3; }
friend int gamma (alpha, beta) };
class beta
{
  private : int data;
  public : beta ( ) :
          { data = 7; }
friend int gamma (alpha a, beta b)
    int gamma (alpha a, beta b)
  { return (a.data + b. data); }
};          // output = 10
```

Scanned by CamScanner

Que:- Consider a Companies payroll application which has two types of employes one type of employee paid a percentage on their total Sales and other types of employees in addition to Commision on the percentage on sales also get a basic payes.

Soln:-

```
# include <iostream>
# include <string>
Class Employee {
    private:
        String fname;
        String lname;
        String SSnumber;
        double tsales;
        double Crate;
    public:
        Employee(Const string &, Const String&, Const String&,
            double =0.0, double = 0.0);
        Void Setfname(Const String &);
        String getfname( ) Const;
        Void Settsales (double);
        double gettsales( ) Const;
        double getCrate ( ) Const;
        double earning ( );
        Void print ( );
};
```

Scanned by CamScanner

```
Employee::  Employee( const string &first, const string &last, const string
                &ssnumber, double tsales, double crate )
{       fname = first;
        lname = last;
        SSnumbers = ssnumber;
        tsales  = tsales;
        create  = crate;
}
Void Employee :: setfname (const string &first)
        { fname = first; }                              fname;
String Employee.getfname() const {return ~~fname~~; }

Void Employee:: setlname (const string &last)
        { lname = last; }
String Employee :: get lastname() const {return ~~lname~~; }

    Set SSnumber ( ) ——
    getssnumber ( ) ——
    (settsaless ( ) gettsaless ( )
{ tsales = (tsales <0.0)? 0.0 : tsales; }
    double getSales () const {return tsales; }
Void Employee : setcrate (double crate) {
        crate = (crate <0.0) ? 0.0 : crate; }
    double Employee :: getcreate() const {return (crate); }
double employee :: earnings ( ) {
            return (create * tsales); }
            Void Employee :: print ( );
{ Cout << " firstname " << fname << " lastna. " << lname
<< " ssnumber " << snumber << " Total Sales " << tsales
<< " Create " << crate << " Earnings " << earnings() << endl;
```

Scanned by CamScanner

08/10/2018

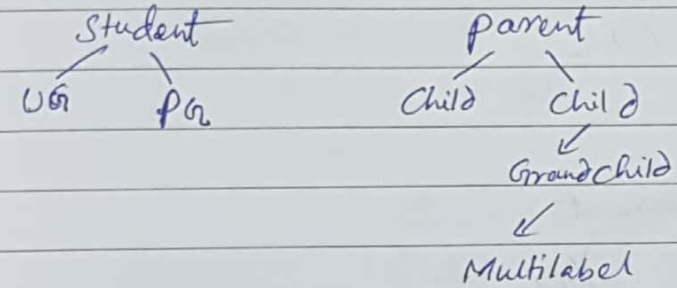
```
Void pt(int x) { Cout << x; }
main( )
{ int p, q, r, s, t, u;
    p=12;   p.g(12); → p.a=12; p.b=0;
    q=17;
    r=p/5; pt(r); → 2   2.4
    s=q/3; pt(s); → 5   5.67
    t=r+s; pt(t); → 7   8.07
}
```

Class hour

Scanned by CamScanner

10/10/2018

Inheritance :- (i) IT is one way relationship
(ii) operator and function overloading are not inherited
(iii) friend function could not inherited.
(iv) Except very simple type of ~~Customers~~, Constructors Constructor are not inherited.



Virtual functions :- A function is called virtual function if it has no ~~body~~ and expressed as virtual ~~body~~ void show = 0

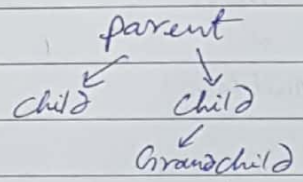
These functions are used to create abstract classes. A class is called abstract if it has atleast one virtual function.

EX-1
```
# include <iostream>
Class Base {
        public:
                Virtual Void show ( ) {
                Cout << "\n Base"
                }
Class Dev1 public Base
    { public :
        Void show ( )
    { Cout << "\n Dev1; } }
Class Dev2 public Base
    { public:
        Void show ( )
    { Cout << " Dev2" } };
```

Scanned by CamScanner

```
void main ( ){
DEV1  dev1;
DEV2  dev2;
Base  * ptr;
ptr = &dev1;
ptr = show ();
ptr = &dev2 ();
ptr = show ();
}
```

output:- DEV1
DEV2




```
Class parent{
        protected:
            int Basedata;
        }
Class child1 : Virtual public parent{ ... };
class child2 : Virtual public parent{ ... };
Class Grandchild : public child1, public child2
{ public:
    int getdata ()
    { return basedata; }
  };
```

In this case parent class will become virtual abstract class.

Scanned by CamScanner

friend function :- If a non-member function of a class wants to access its private data then the class should be declared as friend of the class having non-member function. The function is called the friend function.

EX-2

```
#include<iostream>
class beta;
  class alpha{
      private: int data;
      public:
          alpha()
          { data = 3; }
        friend int Gamma(alpha, beta)
  }
  Class beta{
      private: int data;
     public: beta()
          { data = 7; }
    friend int Gamma(alpha a, beta b);
       int Gamma(alpha a, beta b)
    {
       return (a.data + b.data); }
    };
```

10

EX-3

```
#include<iostream>
  class Distance{
          private feet;
          float inches;
public: Distance()
      { feet = 0;
        inches = 0.0;
      }
```

Scanned by CamScanner

```
Distance (int f, float in)
{ feet = f;
  inches = in;
}
friend dista
friend Distance operator + (Distance, Distance);
friend Distance square (Distance);
};

Distance Distance : operator + (Distance d1, Distance d2)
{  int f = d1.feet + d2.feet;
   float l = d1.inches + d2.inches;
   if (l > 12.0) {
        l -= 12.0;
        f++;
   }
    return (Distance (f, l));
}

Distance Distance : square (Distance d)
{ float feetl = d.feet + d.inches/12.0;
  float feetsquare = feetl * feetl;
   int f = int (feetSquare);
   float l = 12 * (feetsquare - f);
    return (Distance (f, l));
}
```

Static

programm

Ex-

Soln;-

Scanned by CamScanner

Static function :- Static functions are used to access static data in terms of object oriented programming to share the data amongst all the objects.

Ex- Consider the following program to find out/identify the ID of each object in an application.

Soln:-

```
#include <iostream>
Class Gamma {
private:
        Static int total;
        int id;
public;
        gamma ( )
        { total ++;
          id = total;
        }
        ~ gamma ( )
        { total --;
        Cout << " Destroying id number = " <<id;
        Static void show total ( )
        { Cout << " Total is " <<total;
    void show id ( )
    { Cout << " \n id number is = " <<id; }
    void main ( )
  { Gamma g1;
    gamma :: show total ( );
    gamma g2, g3;
    gamma :: showtotal ( );
        g1. showid ( );
        g2. showid ( );
        g3. showid ( );
```

Scanned by CamScanner

Total is = 10
Total is = 5
" " = 3
id no is = 1
id no " = 2
" " = 3
Destroying id no is = 3
" " = 2
" " = 1

Template :- It is utilized for software reuse and is a generic programming Concept. There are two types of template.

(i) function template
(ii) class template.

Function Template :- A function template is defined once and using the same logic Compiler will Create with different codes for different data types.

~~A class template~~

Class template :- A class template is also written once to contain some data type and same class type can be used to contain other data types.

Template <class/type name parameter of template>

Template <class T>
Template <typename T>
Template <class element type, int Count>

Scanned by CamScanner

Ex:- Write a function template to find all the max. of three integers or floating point nos.

Soln:-
```
#include <iostream>
template <int/float T>
Tmaximum (Tvalue1, Tvalue2, Tvalue3)
{
int maximumValue = value1;
    if(value2 > maximumValue;
        maximumvalue = value2;
if(value3 > maximumvalue)
    maximumvalue = value3;
    return (maximum.value); }

int main()
{ int x1, x2, x3;
    maximum (x1, x2, x3);
    Cout << "The maximum value is" << endl;
}
```

Ex ⇒ Write a function template to print array of integers, floating point numbers and user defined data types, say distances:

Soln:-
```
#include <iostream>
    template <typename T>
    void printarray (Const T * Array, int Count)
                    ↓ As the array cannot be modified.
    { for(int i=0; i<Count; i++)
        Cout << Array [i] << " ";
        Cout << endl;
    }
int main ()
{ Const int Acount = 5;
    Const int Bcount = 6;
```

Scanned by CamScanner

```
Const int Ccount = 8;
int a[Acount] = {2, 3, 4, 5, 7}
Int b[Bcount] ?[3, 6, 8, 9, 7, 5}
int c[ccount] = { - - - - }

printarray(a, Acount);
printarray (b, Bcount);
printarray (c, ccount);
  return ();
}
```

11/10/2018

Class Template: We Consider a class template as a stack with out contents.

Ex:-

```
#include <iostream>
template <Typename T>
    class stack {
        public:
            stack (int=10)
        ~stack() {delete [] stack ptr; }
        bool push (Const T&);
        bool pop (T &f);
        bool isempty ( ) Const {return top == -1}
        bool isfull ( ) Const {return top == Size -1;}
        private:
            int Size;
            int top;
            T * stack ptr;
        };
```

Scanned by CamScanner

```
template <Typename T>
  bool Class <T>::
      push (const T & push value)
      if (! isfull)
      {
        Stack ptr [++top]
            = push value;
      }
      else
          return false;
      class <T> :: Stack (int s)
      { if (s > 0) { size ? 10;
        top (-1);
        Stack ptr = new T (s);
      }
      template <Typename T >
        bool stack <T>: pop (T & popvalue)
        { if ( ! isemptyc))
          { popvalue = stack ptr [top -1]
              return true;
          } else
              return false;
      int main ( ) {
          Stack <double> double stack (5);
        double value = 1.1;
      Cout << " push element into stack " << \n" <<
      while (double stack. push (double value))
      { Cout << double value << " ";
              double value += 1.1;
      }
      Cout << " Stack is full, Cannot be pushed " <<
          double value << "\n \n popping elements from the
                                  Stack ";
```

Scanned by CamScanner

while (double stack.pop(double value))
{ cout << "Stack is empty, Can not pop \n"; };
return 0;
}

Output:- push element onto Stalk
1.1 2.2 3.3 4.4 5.5
Stack is full, Can not be pushed.
poping element from the Stack
5.5 4.4 3.3 2.2 1.1
Stack is empty, Can not pop.

Exercise: write a function template to sort array of int, real and strings in OOP, by using insertion sort algorithm and pointers.

Exception Handling: An exception is an error which occurs frequently during execution of program. Some of the error requires program to correct and continue whereas other requires abnormal/sudden termination of the program.

EX:- 1. Integers/reals divided by zero → terminate program.
2. New is unable to allocate the memory.
3. Array subscripts are out of range.

It is handled by 3 mechanism.
try(), catch(), throw();

Scanned by CamScanner

25/10/2018

classmate
Date
Page

Exceptions :- Exception handlers
~~by~~ try, Catch, throw

Exception handlers are introduced by the Catch clause within the try block prefix of which the following code fragment is representing.

```
Class X {
  - - -
  T m() {
  - - -
    try {
      Yb = new Y( );
      b.performOpeation( );
      - - -
    } Catch (Exception en) { errorrecovery( ); }
    Catch (IOException e) { error report( ); }
    m( );
  } }
```

Raising Exceptions :- An exception condition is ultimately represented by an exception object derived from the predefined Exception class. A condition is made known by throwing an appropriate object via throw ( ) statement, to be subsequently caught by an associated handler.

```
Class Y {
  void performOperation( ) {
    if (Statement)
      throw new transmissionError( );
  }
}
```

Scanned by CamScanner

Ex:-

```
class Stack{
    int height;
    Object items[];
    Void push(Object x) through fullstack{
        if (items.length == height)
            throws new fullstack();
            items[height++] = x;
    }
Object pop() throws Emptystack{
        if(height == 0)
            throws new Emptystack();
            return (items[height--]);
    }
    void init (int s){
        height = 0;
        item = new Object[s];
    }
    stack(int s)
{ init(s); }
    stack(){
        init(10); }
}
```

Scanned by CamScanner

Ex:

Regular exception handling: Where there are multiple code fragments with similar error handling logic i.e global exception handler could be header.

```
class Email{
- - - - - -
Void makeTestConnection (String Host){
        openSocket (host);
        if (! IOerror ()){
        Check response ();
         give greetings ();
        } }
        void giveGreeting (){
          writemessage ("Hello" + hostname);
          if (IOerror ())
              error code = 404;
          else
              checkresponse ();
          }
Void  verifyUser (String user)
{
   writemessage ("verify" + user);
     if (IOerror ())
         errorCode = 404;
     else
        check response ();
   }
}
```

Scanned by CamScanner

```
class Email{
  void send(String address){
    try{
      errorcode = 404;
      makehost Connection (emailhost of (address));
      verify user(email Host of (address));
    }
    Catch (IOException x){
      // network error detected
    }
  void makeHostConnection (String ){
      opensocket (host);
      Check response ( )
      give greeting ( );
    }
    void give greeting ( ){
    Write message("Hello" + hostname);
      check reponse ( ); }
    void verifyuser (String user){
      Write message("verify" + user);
      Check response ( ); }
```

Sub-Condition

Layered Exception

29/10/2018

```
class a
{ void g(){sop("A");}
  void h(){sop("B");g();}
}
class b
{ void g(){sop("c");}
}
class m{
    public static void main(String tL])
  {  b x; x=new b();
     x.g(); ⟶ e
  }
```

file input/output Stream

Data not saved will be lost, and if we ever want to work with data again, then we must save it in a file. We call the action of saving or writting, data to file as "file upload" "file output" and the ~~acting~~ action of reading the data as "file input".

Before we can access the data from file, we must create the "file" object (from the java.io package) and associate it to the file. We do so by calling the constructor.

file infile = new file("Sample.data");

We use the term "file access" to access or write the data in "Sample.data", to be precise, read access or write access.

Scanned by CamScanner

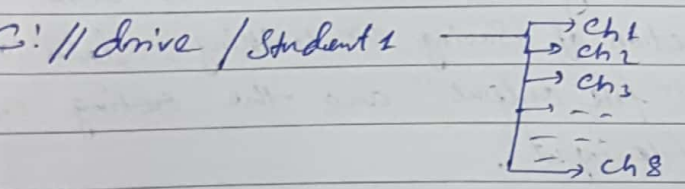
When you run a program whose source file is located in the directory X, then the current directory in X.

It is also possible to open/access file that is stored in a directory other than current directory. Suppose there is a file "Sample1.data" in the Students directory; then we open

```
file infile = new file("C:\\Students\\Sample1.data");
```

We can check if a file object is associated correctly to an existing file.

```
if (infile.exists()){
        //infile is associated correctly to an
existing file
else } // infile is not ass. with the existing file}
```



A file object can also be associated to a directory. Suppose we want to the contents of the directory ch8 we can first create a file object and associate to the directory.

```
File directory = new file("C:\\drive\\Student\ch8");
String filename[ ] = directory.list( );
for(int i=0; i< file name.length ; i++)
{ System.out.println(file nam[i]) };
```

Check whether file object associated to the file or directory by calling its boolean isfile.

Scanned by CamScanner

```
File file = new File ("c:\\drive\\ student\\ch8");
    if (file.isFile()){
      System.out.println ("I am a file");
    else{
      System.out.println("I am a directory");
      }
```

We can also use JFileChooser in java to user Select a file.

```
JFileChooser chooser = new JFileChooser();
    ~~chooser.showOpenDirectory(null);~~
    chooser.showOpenDialog(null);
```

When We create an instance of JFilechooser by passing no argument to the Constructor, as in previous example, it will list the Content of the user's home directory.

eg. My documents.

We can also Set the filechooser to list the Contents of a desired directory when it first appears on the Screen.

```
JFileChooser chooser = new filechooser("e:\\drive\studdt
\ch8");
    Chosser.showOpenDialog(null);
```

Scanned by CamScanner

31/10/2018

```
JFileChooser =new JFileChooser( );
Chooser. showopenDialog (null);
```

The null argument to Show open Dialog indicates that there is no ~~parameter~~ parentframe window, and the dialog is displayed at the Color of the Screen. When we Create an instance of a JFileChooser by passing on arguments to the Constructor, as in the above example, then it will always list the Content of user's home directory.

If we want the file chooser to list the Content of a desired directory when it first appears on the Screen. We write following.

```
JFilechooser chooser = new JFilechooser("C:\\ drive
                        \student1\ ch8");
Chooser. showopenDialog (null);
```

listed of designing a fixed directory, we may wish to begin the listing from the current directory. Since the current directory is different when the program is executed ~~a~~ di from a different directory. We need a

```
String Current = System. getproperty ("user.dir");
JFile Chooser = new JFile chooser (current);
chooser . Show openDialog ();
```

The Content of Current is the path name to the Current directory. To check whether the user has clicked on the open or cancel button, We test a return value from the Show open Dialog method

Scanned by CamScanner

```
int Status = chooser.showOpenDialog(null);
if (status == JFileChooser.APPROVE_OPTION){
    System.out.println("open is clicked");
} else{
    System.out.println("Cancel is clicked");
}
```

Once we determine that open is clicked, than we can retrieve the selected file/directory as:

```
File SelectedFile;
SelectedFile = Chooser.getSelectedFile();
String CurrentDirectory;
CurrentDirectory = Chooser.getCurrentDirectory();
```

To find out the name and the path name of a selected file. We can getName and getAbsolutePath methods of the fileclass.

```
File file = Chooser.getSelectedFile();
System.out.println("Selected file" + file.getName());
System.out.println("FullPath" + file.getAbsolutePath());
```

To display a JFileChooser with a "Save" button, we write

```
Chooser.showSaveDialog(null);
```

Scanned by CamScanner

BX-1

Write a simple specifying the use of JFileChooser filename, directory name and its path, open and Save buttons to run the program.

Soln:-

```
import java.io.*;
import java.swing;
Class Ch8testfilechooser{
    public Static Void main(String [] args) {
    Jfilechooser chooser;
     File file, directory;
     int Status;
     Chooser = new JfileChooser( );
     Status = chooser. Show openDialog (null);
    if (Status == JFileChoosn. APPROVE_OPTION) {
      file = chooser.getSelectedfile( );
      directory = chooser. getCurrentDirectory( );
    System. out. println ("Directory is " + directory.getName());
    System. out. println (" file is " + file.getName ());
    System.out. println(" full pathname is" + file get
                        Absolutepath ( ));
    } else {
       System.out. println ("openfiled dialog is cancled");
    }
    System.out. println ("\n\n");

Status = chooser. ShowSaveDialog(null);
  if (Status == JFilechooser. APPROVE OPTION){
      file = chooser. getSelected file( );
      directory = Choosn. getCurrentDirectory( );
    System.out. println ("Directory is " + directory. getName());
  System.out.println ("file Selected for Saving data is"
                      + file.getName());
```

Scanned by CamScanner

```
System.out.println ("file full pathname is" + file.getAbsolutePath());
} else {
    System.out.println ("Cancel");
    }
```

There is no distinction between the open and Save dialogs created, respectively by ShowOpenDialog and ShowSaveDialog other than the difference in button label. In fact, they are really a shorthand for calling "showDialog" method. By calling ShowDialog method, we can specify the button lable and the dialog title

Ex:-
```
JFileChooser chooser = new JFileChooser();
chooser.showDialog (null, "Run");
```

file filter:- file filter is used to remove unwanted files from the list. let's say we want to apply a files so that only java source files (those with extension java) are listed is your file chooser. To do so we must define a subclass of the Javax.Swing.file ChoosFileFilter class. and provide accept and getDescription.

```
public boolean accept (File file);
public String getDescription();
```

The accept method return true if the parameter files to be include in the list. The getDescription method returns a text that will be displayed as one of entries for the "Files of type:" drop down list.

Scanned by CamScanner

Ex-2 Write a program to filter only java source files for listing in JFileChooser.

Soln:-

```
import java.io.*;
import javax.swing.filechooser.*;
Class Javafilter extends filechooser
Class Javafilter extends filefilter {
    private static final String JAVA = "java";
    private static final String DOT = ".";
    public boolean accept (File f){
        if (f.isDirectory()){
            return true;
        }
        if else(extension(f).equalsIgnoreCase(JAVA)){
            return true;}
        else {return false;}
    public String getDescription(){
        return "Java Source files (.java)";
    }
    private String extention(File f){
        String filenam = f.getName();
        int loc = filename.lastIndexOf(DOT);
        if(loc>0 && filename.length()
        if (loc>0 && loc<filenam.length()-1){
            return filename.Substring(loc+1);
        }
        else return "";
    }
}
}
```

Scanned by CamScanner

classmate
Date
Page

with the filter class "JavaFilter" in place we can set a file chooser to list only directories and Java source files by writing

```
JFileChooser chooser = new JFileChooser();
chooser.setFileFilter(new JavaFilter());
int status = chooser.showOpenDialog(null);
```

01/11/2018

Example :-
try Catch throw

1) Write a program to find/print the age of a person. (bornyear)

Soln :-

```
import java.util.*;
class AgeInputMain{
    public static void main(String[] args){
    GregorianCalendar today;
    int age, thisyear, bornyear;
    String answer;
    Scanner scanner = new Scanner(System.in);
    AgeInputVer1 input = new AgeInputVer1();
    age = input.getAge("How old are you?");
    today = new GregorianCalendar();
    thisyear = today.get(Calendar.YEAR);
    bornyear = thisyear - age;
    System.out.println("Already has your birthday this year
        (yorN)?");
    answer = Scanner.next();
```

Scanned by CamScanner

```
if (answer.equals("N") || answer.equals("n"))
{ bornyear--;
}else{
        System.out.println("\n you are born is "+ bornyear);
}


# age = Scanner.nextInt();
   System.out.print(prompt);
   try{
        age = Scanner.nextInt();
      }
   exception(IoException ex){System.out.println("ex.toString());
```

Scanned by CamScanner

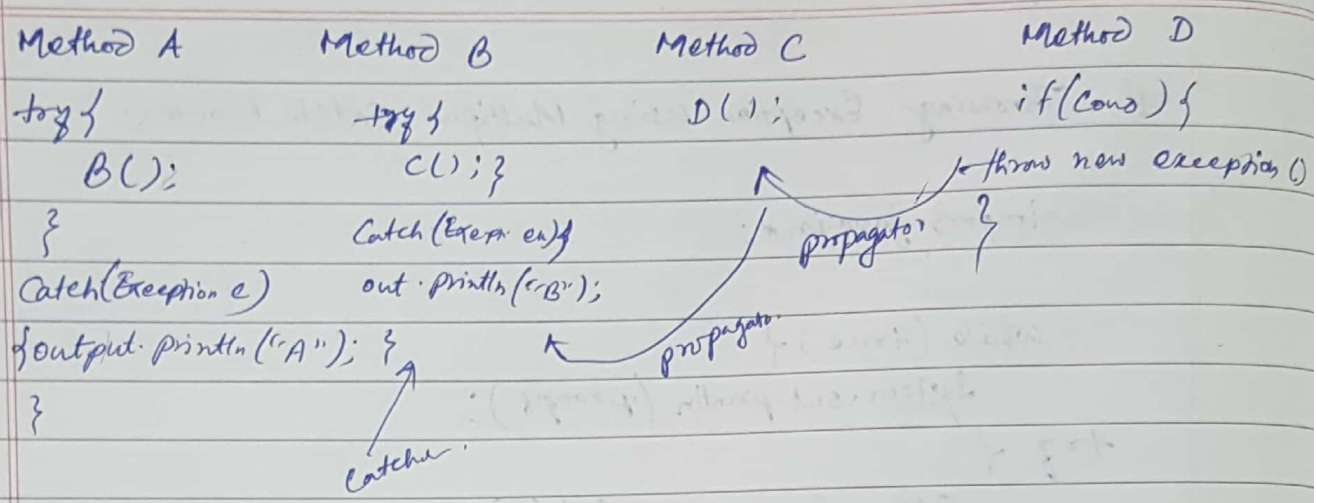
8/11/2018

Throwing Exception using Multiple catch blocks:

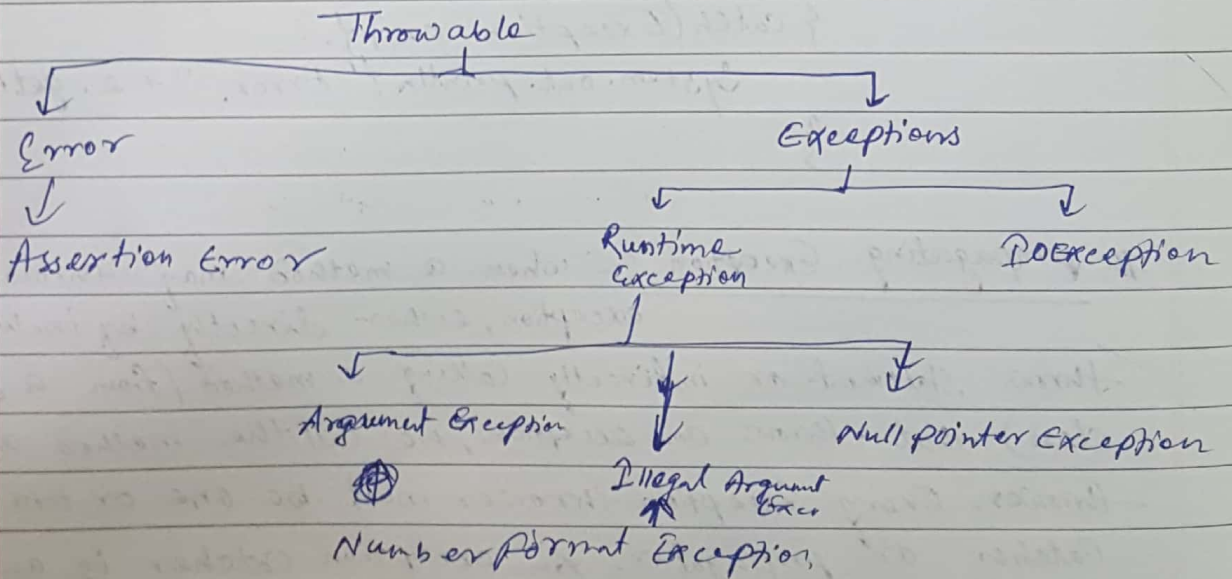
```
import java.io.*;

while (true){
System.out.println (prompt);
try {
age = scanner.nextInt ();
if (age<0){
throw new Exception ("Negative Intege is invalid");
}
return age; // input okay so return the value & exit.
} Catch (InputMismatchException e){
Scanner.next ();
System.out.println ("Input is invalid.\n" + "please
enter digists only");
} catch (Exception ex){
System.out.println ("Error: " + e.getMessage());
}
```

propagating Exception :- when a method may throw an exception, either directly by including a throw statement or indirectly calling a method (from a different class) that throws an exception, we call the method an exception thrower. Every Exception thrower must be one or two type: Catcher or propagator. An exception catcher is an exception thrower that includes a matching catch block for the thrown exception where an exception propagator does not. The designation a method of a method being a catcher or propagator is based on a single exception.

Scanned by CamScanner



Types of Exceptions :- There are two types of Exceptions. Checked and unchecked. A checked exception is an exception that is checked at compile time. All other exceptions are unchecked exceptions, also called as "Runtime Exception", Ex. Trying to divide by 0 (Arth exep) or trying Convert a string of letters to an integers are runtime errors.



Scanned by CamScanner

12/11/2018

JfileChooser chooser = new filechooser();

chooser.setfilechooser(new Javafilter());

Int status = Chooser.showopenDialog(null);

with the filter class Java filter in place we can set a file chooser to list only directories and java source files.

file outfile = new file("Sample.data");

Ex:- A Sample program to Save data to a file using file output Stream.

Ex:- A test program to read data from a file using file input Stream.

High-level file I/O :- By using Data output Stream we can output Java primitive data type values. A DataOutput Stream will take care of the details of converting the primitive data type values to a sequence of bytes. In the following program, we write out the values of various Java primitive data types to a file.

Ex.- Write a test program to Save data to a file using DataOutputStream for high-level I/O.

Sol.-

```
import java.io.*;
class MiscMath {
    public static void main(String[] args)
        throws IOException {
// set up the Streams.
```

Scanned by CamScanner

```
File outfile = new File("Mse.data");
FileOutputStream outfilestream = new FileOutputStream(outfile);
DataOutputStream outDatastream = new DataOutputStream(outfilestream);
// write values of primitive data types to a stream
    outDatastream.writeInt(974321 5390);
    outDatastream.writeLong(11111L);
        "     .writeFloat(1.2356F);
        "     .writeDouble(3333D);
        "     .writeChar("B");
        "     .writeBoolean(true);
    // output is done; So close the stream
        outDatastream.close();
}}
```

Scanned by CamScanner