Unit - 3

3. Exception handlings.

example for c program

```
void main()
{
    int a,b,c;
    a=100, b=0;
    c=a/b
    printf("%d",c);
}
```

⇒ It will not show error.

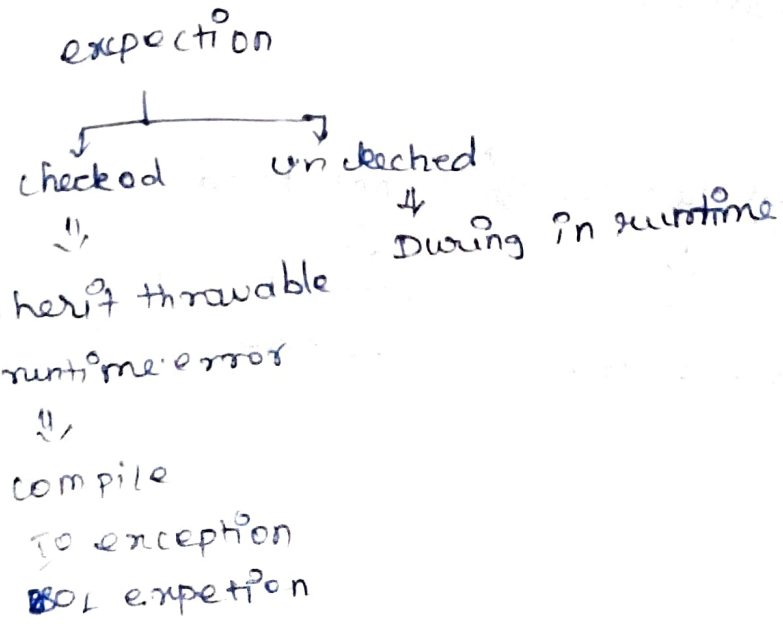
It means exception; there is no systax error, handlings. ⇓ It is unwanted event that distrub the normal flow of execution during runtime.

Advantage of Exception handling

⇒

⇒

To avoid runtime distrabance we use exception



Keywords

try => to try exception

catch => to catch exception

```
Public class main
{
  Public static void main (String[] args)
  {
    int x = 100/0;
    sop (x);
```

} Arithmetic exception

one datatype to another datatype

```
String s = "abc";
int i = Integer.ParseInt(s);
sop(i);
```

} Number format exception

any operation with null

```
String st = null;
sop (st.length());
```

} NULL pointer exception

try to access an element out of an array

```
int a = new int[5];
a[7] = 10;
```

} Array Index Out Of Bounds exception

Class not found / file not found

↓ compile a Class not saved

↓ access file not accessed

Exception Handling Mechanism

```
Public class Main
{
  Public static void main(String[] args)
  {
    try
    {
      int x = 100/0;
      sop(x);   String s="abc"
                sop(s.length());  } => it will not excecute
    }                                because exception is
    catch (Arithmetic exception e)   occurs next will go only
                                     catch.
    {
      sop("Divide by 0");
    }
    sop(" Hi exceptional handling");
  }
}
```

Try can be followed by any number of catch blocks.

We can use super class as Exception in super class

* Try can have any number of catch blocks.

```
try
{ int x = 100/0;
  String st = NULL;
  sop(st.length());
}
catch (ArithmeticException e)
{ sop("Divide by zero");
}
catch ("NULL pointer exception e)
{ sop("string is null");
}
sop("Hi exception is handled");
}
```

o/p
Divide by zero
Hi exception handled

because start of below exc will not be executed

* superclass

```
try
{ int x = 100/00;
  String st = NULL;
  sop(st.length());
}
catch (exception e)
{ sop(e);
}
catch (ArithmeticException e)
{ sop(e);}
catch(NullpointerExceptione)
{ sop(e);}
```

This is enough

- If we use super class first error will occur because it will encompass the error itself
- It can be given at last.

Finally blocks → Statements that is to be executed always must be present in finally

case 1 : [Exception occurs]

```
try
{ int z = 13/0;
  sop(z);
}
catch(ArithmeticException e)
{ sop(e);
}
finally
{ sop("Always executed");
}
sop("Hi exception");
```

Exception occurs

o/p ⇒ (Arithmetic caught
Always executed
Hi exception)

* case 2 : [exception throw back and it will not caught by catch]

exption is not caught & finally is executed.

```
try
{ int z = 13/0;
  sop(z);
}
catch(NullPointerException e)
{ sop(e);
}
finally
{ sop("Always exception");
}
```

o/p

Always executed

case 3: [Exception doesn't occurs.]

```
try
{ int z = 13/2;
}
catch(ArithmeticException e)
{ sop(e);
}
```

o/p

finally block

Nested try

Try block inside another try block

```
try
{ int x = 100/0;
  sop(x);
  try
  {String st = NULL;
   sop("st.length()");
  }
catch(NullPointe)
  {
   sop(e);
  }
}
catch(Arithmetic... )
{ sop("Divide by zero");
  }
sop("Hi exception");
}
```

o/p

Divide by zero

Hi exception

= instead we can give super class outside too

If inner catch doesn't support it will check outside.

try → catch → finally → order to In between we should insert any block.

(If catch block cannot handle inner try it goes to outer try);

Java throw keyword ⇒ without try and catch we can use // correct way is try{ throw } catch{};

throws * used declare an exception

* Throw the exception in this keyword

Syntax throw new (whatexception you want)

exception alone (or) Ari exc

example = Manuall throwing;

PCM → if n is not present, we have do create object

{

static void validate (int age)

{ if(age<18)

throw new ArithmeticException ("not valid");

Syntax

else

S.O.P("welcome to vote");

}

P SVM(String[] args)

{

validate (13); // Main M1 = new new Main(); m1.validate;

25

S.O.P("rest of the code...");

}

}

o/p ⇒ Exception throwback to "Main" : not valid

(or) o/p ⇒ welcome to vote

rest of the code;

---

try

{

}

throw

{ = default throw is present in java.

} ⇒ it is already predefined in java.

// if we want written anything we can use it. (Manuall throwing)

catch

{

}

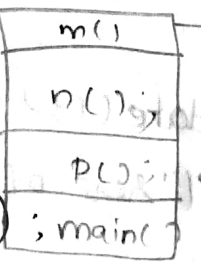
example.

(x) exception propagation (unchecked exception)

example

```
class Test
{
 void m()
  { int data = 50/0;
  }
 void n()
  {
    m(); // function call.
  }
 void P(){
   try
    {
     m();
    }
   catch (Exception e)
   { sop("exception hanled");
  }}
Psvm(String[] args)
  {
     Main O = new main();
      O.p();
    } s.o.p("Normal flow...");
  }
```

It check next next outer block, the exception is hanled, it goes to try block



call stack like this.

o/p exception hanled
Normal flow ...

IO (checked exception) exception

⇓

It cannot be propagation.

checked → not propagated.

unchecked → can propagated

```
class Test
{
  void m()
  {
  throw new IOException // checked Exception
  }
```

⇓

Showing error.

checked exception can be done by thr

```
class Test
  {
    void m() throws IOException
    {
      throw new IOException("device error");
    }
  void n() throws IOException
    {
      m();
    }
  void p()
    {
    try{
        n();
      } catch(Exception e)
        { sop("exception handled");
        }
  psvm(s)
    Test t = new Test();
    t.p();
    sop("Normal"); } }
```

// o/p : IO exception
Normal flow.

we can not propagate checked exception

- we can propagate unchecked exception

throws → keyword
↳ checked exception

⇒ manual throw of an exception to d
an exception

```
void m() throws IOException
    {
     throw new IoException("  ");
    }
void n() throwsIoException
```

case1: you caught the exceptio
case2: de clear the exc | specificing throws with the method.

Case1:

```
void method othrows IOE Exception
 {
   throw new IOE exception( );
   }
 }
PcM
PSVM( )
trys
     main M : new Main()
      m. method ))
    }
   catch (Exception e)
    {
      Sop (" ");
    }
Sop( );
}
```

case 2

```
class M
{ void method() throw Exception
  { SOP(" ");
  }
}
Class Test throws
{
PSVM(String args[]) throw IOException
                          ↓↓
                       declare
Main m = new Main()
 m.method();
 SOP();
}
```

↳ followed by instance

throws new java.io IOE Eception("device error")

↓↓

Parametrized constructor

Instance is also called as object

throws => followed by class

Throw	Throws
Explicity throw can exception	Declare an exception
cannot throw multiple exception	can throw multiple exception

user defined exception

```
class validage extends exception
{
  validage (string s)
   {
     super(s);
   }
}
```
} base call exception

```
Public class Main
{
  void validage (int age)
   { if (age<18)
      throw new validage ("not vote");
  else
     s.op("yes vote");
}
  psvm( )
  { main m1 = new main( )
     m1. validate (15);
     m1. validate (25);
  }
}
```
⇒ It doesn't occure because exception is before line.

// 25)
(15) } -> two are worker

Threads

Multitasks
↳ read two line of code on same time.

example
Audio music and doing with any work.

⇒ Doing more than one activity is multitasks.

Multitasking
- Process Based. (Multiprocessing)
- Thread Based (Multithreading)

Multiprocessing
⇒ Doing more than on processing is multiprocess

[Program is exuted is called as processing.
Program is set of code]

⇒ Each an every processing spread memory.
⇒ heavy weight.
⇒ the cost of communication is high. (minum time)
⇒ switch is high.

Threads
Sub units of programs is called Threads.

Multithreads
⇒ It consists of two (or) more

⇒ smallest unit of dispatchful.

⇒

⇒ light weight

⇒ switch one thread to another thread is easy.

Programs

life cycle of a thread

⇒ newborn thread (thread is created)

⇒ (Active thread) Runnable

⇒ Running thread

⇒ (killed thread) Blocked

⇒ Dead (killed thread);

Threads (1)

Multitasking - doing more than one tasks in the system

Program - code
process - executing

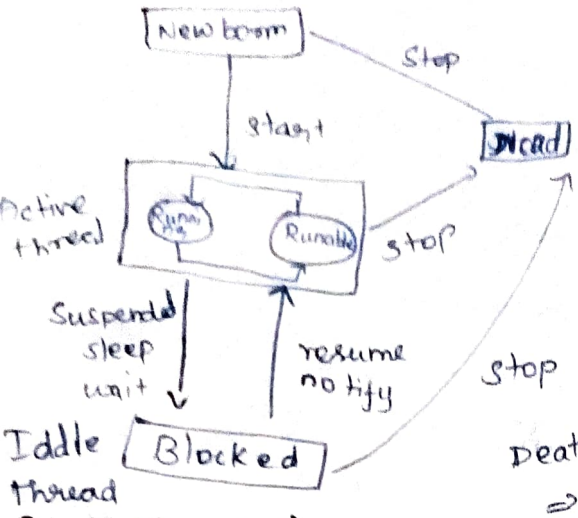
Process based
* Multiprocessing
* Doing more than one process
* separate memore will be there
* Heaves weight
* cost of communication and time taken by CPU is high.

Thread based
* Multithreading
* part of a program is thread (subunits of a program).
* consistin of more than 2 parts in a program that runs concurrently
* smallest unit of dispatchable code - Thread
* light weight

Advantage of Multithreading
* Shares the same memory space
* thread is light weight
* minimal cost of communication, time
* doesn't block the user

Application
Games, Animation

Life cycle of thread
Many states it can enter
* Newborn thread -> thread is created
* Running [Active thread) -> Thread is executing
* Runnable ["] => ready for exution, waiting for process
* Blocked [not runnable (Iddle thread) => Prevente form 2,3 (Sleeping, Suspended
* Dead [killed thread) => when it is completed



Death
=> Natural death
=> premature death

Default Thread class
Thread in a class found in java.long.package.

Thread priority

Base on priority CPU sets for execuition

1 to 10 = priority range
Normal priorty = 5
we can set priority using setpriority fun.

* each thread will be having priority

Single processing

Only 1 program, CPU will be sitting iddle when input/output operation works.

In order to use CPU time efficiently we are going to multiprocessing

Context switching

* Switching of one thread to another thread or one process to another process
* based on priority it will be done
* If all the thread are of same priority will run for all the thread in round robin way

Two ways it takes place

* thread can voluntarily gives to next thread.
* Thread is given by the high priority (or permitted)

Thread functions

1) String getname() - gets the name of running thread
2) start() - start new thread of execution
3) void run() - Execution of thread
4) void sleep(int sleeptime) - Suspended for some time
5) Void yield() - The current thread pauses for some times & allows other to execute
6) void join() - once a thread is called, current thread will wait till calling thread completes its execution

Program

PCM

```
{ psvm( )
    { Thread t = Thread.currentThread( );   // static member fun
      // runs the current thread.
      sop("current thread " + t);
      sop(t.getname());
      t.setName("My thread");
      sop("After changing " + t);
      try
      { for(int i=1; i<3; i++)
          { sop(i);
            thread.sleep(1000);
          }
      }
```

```
catch (InterruptedException e)
    { sop("main thread Interrupted");
}
}
```

o/p

current thread : Thread [main, 5, main]

main (name)

After changing [My thread, 5, main]

name, priority

1
->after 1000ms it will
2 be printed
3

[Default sleeping time in milliseconds]

. we can change it.

```
Static thread currentthread ()
{ }
```

returntype, name

```
final void setName (String ThreadName)
{
}
final String getName()
{
}
Static void sleep (long ms, int ns) throws
                    InterruptedException
{ }
```

* t.SetPriority(8) => priority to 8

Creating thread

* implemented in the form of objects
* run() initiated with help of start(), Two ways
  - → Extending thread class
  - → By Implement Runable Interface

1) Extending class thread

```
class multi extands thread
{ Public void run()
  { sop("run");
  }
psvm()
{ multi t1 = new multi();
  t.start(); // run() called through start();
}
}
```

o/p
run

2) class multi implement Runnable

```
{
Public void run()
 { sop("run");
 }
psvm
 { multi m = new multi();
   thread t1 = new thread(m);
   t1.start();
} }
```

o/p → run

—

```
PCM extants thread
{ public void run()
  { for(int i=1; i<=5; i++)
    { try
      { thread.sleep(1000);
      catch } (Interrupted Exception e)
        { sop(e);
        }
} } sop(i)
```

Perm

```
{
    main m1 = new main();
    main m2 = new main();

    m1.start();
    sop("m1");
    m2.start();
    sop("m2");
}
```

o/p

```
m1
m2
1   (since it have
1    same priority
2    CPU shares
2    in round robin
3    way).
3
4
4
5
5
```

Multithreading

```
class test Implements Runable
{
    Thread t ;  → // create object (since the lifetime of object to ... so give here)
    test()
    {
        //t = new thread (this, "Demo thread");  // thread
        S.op("child Thread" + t);
        t.start();
    }
    Public void run()
    {
        try
        {
            for(int i=5 ; i>0; i--)
            {
                S.o.p("child thread :" + i);
                Thread.sleep(1000);
            }
        }
        catch (Interrupted exception e
        {
            S.op(e);
        }
        sop("Main Thread Exiting")
```

→ Parametried constructor
Passing Parameter
current thread
name of thread

o/p child thread : Thread [Demo Thread, 5, main]

Main Thread : 5
child Thread : 5
4
3
2
1

main thread is existing.

```
class test extends thread
{ test()
  {
    super("Demo Thread"); // constroer thread class
    S.O.P("child Thread;"+this);
                              ⇓ name / priority will give
    start();
  }
Public void run()
  { try
    { for(int i=5 ; i>0; i--)
      { sop("child thread;"+i)
        Thread.sleep(1000);
      }
    sop("child class existing.
  }
  catch
    { sop(e);
    }
    sop("Main thread exisitings");
}
```

o/p =>

"

Implement runnable is prefer mostly...
we can use implement runble, because when u add, lots create is needed extands, we can over written in only implement.

Multithreading

```
class test Implement Runable
{
    Thread t;
    String name;
    test (string thread name)
    {
        name = threadname
        t = new Thread (this, name);
        sop("child Thread :"++);
        t.start();
    }
    Public void run()
    {
        try
        {
            for(int i=5; i>0; i++
            {
                sop(name + " : " +i);
                Thread.sleep(1500);
            }
        }
        catch
        {
            sop(e);
        }
        sop(name + "Exiting");
    }
}
```

```
Public class Main
{
  ps void main(String [] args)
  {
    test t1 = new test ("one");
    test t2 = new test ("two");
    test t3 = new test ("three");
    try
    {
      Thread.sleep (2000);
    }
  }
  catch (Exception e)
  { sop(e);
  }
  sop("Main Thread Exiting");
  }
}
```

o/p => child Thread : Thread [one, 5, main]
child Thread : Thread [Two, 5, main]
One : 5
child Thread : Thread [Three, 5, main]

Two : 5
Three : 5
one : 4
Two : 4
Three : 4
Main Thread Exiting
one : 3
Two : 3
Three : 4
Main thread Exiting
one : 2
Two : 1
one : 1
Three 1
one exitising
Two exitising

Decided Implements only because of create method / class.

Start a thread with the code / After no child thread only Mainthread execu.

objectname : Start

Invoke a thread is Run

Join:

Sleeptime is difficult to predict so we are going to 2 types

⇒ Join
⇒ Isalive

Is Join : Until waits thread terminated

Final Boolean Isalive( )
↓ ↳ T/F
can be over written

Join & Sleep should be placed in try block

Isalive -> False → terminates Thread

Join => waits for thread to die
↳ Mainly to achieve Synchronization
First - thread complete then, only next thread exe

Extends Thread class : m, . Join

Implements Thread class : thread . object . Join
obj