**Exception Handling**

**Topics:**

1. **introduction**
2. **Runtime stack mechanism**
3. **default exception handling in java**
4. **exception hierarchy**
5. **customised exception handling by using try catch**
6. **control flow in try catch**
7. **methods to print exception information**
8. **try with multiple catch block**
9. **finally block**
10. **control flow in try catch finally**
11. **control flow in nested try catch finally -4.20cc-4.48**
12. **possible combination of try catch finally**
13. **throw keyword**
14. **throws keyword**
15. **exception handling keywords summary**
16. **various possible compile time errors in exception handling**
17. **customized or user defined exception**
18. **top -10 exception**
19. **1.7 enhancements- try with resource , multi catch block**
20. **difference between final,finally and finalise**

**1.Introduction:**

* **an unwanted and un expected event that disturbs normal flow of program is known as exception handling.**
* **purpose of exception handling is grace ful termaination. if something goes wrong we should not miss or loose anything. such type of termination is grac fult termination**
* **if we didnot handle the exception program will terminate abnormally**
* **eg: if i m going to open db connection while reading data i got sql exception and didnot hndle it . so program will termainate this stage only so no one is going to close that db connection.one db connection is waste. now exception is blocking our resources. so we have to handle it.**

**Exception handling means:**

* **exception handling means we are providing solution or fixing the issue, we are defining alternative way to continue rest of program normally is exception handling.**

**eg:**

**try{**

**read data from london file**

**}**

**catch(FileNotFoundException){**

**use local file**

**read data**

**}**

**main flow will taken into try block. alternative flow available in catch block.**

***2.Runtime stack mechanism:***

* **for every java program we have atleast one thred, which is main thread**
* **for every thread jvm will create one RUNTIME stack.**
* **every method call performed by this thread, one entry will be stored inside corresponding stack.this entry is called stack frame(or) activation record.**
* **for ex thread t1 called method m1 , m1 entry will be stored in stack. once m1 copleted corresponding entry(m1) removed from stack**
* **after completing all the method calls(stack is empty ) thread wilb l be terminated.before terminate any thread corresponding empty stack will be destroyed by jvm**

***3.Default Exception Handling in java***

* **if we are not handling the exception by default what will happen**

**eg:class Test{**

**p.s.v.m(String[] args){**

**doStuff();**

**}**

**p.s.v.doStuff(){**

**domoreStuff();**

**}**

**ps.v.domorestuff(){**

**s.o.p(10/0);**

**}**

**}**

* **inside method if an exception raised , the corresponding method is responsible to create exception object with healp of jvm**
* **in this program domorestuff() is responsible to create exception object**
* **in created exception object, name of exception, description ofexception, location are added**
* **eg: name: ArithmeticException, description: divided by zero, location:domorestuff,dstuff,main**
* **since domorestuuf() dont have any exceptiin handling, jvm will terminate the method abnormally without executing remaining code and remove from runtime stack**
* **now dostuff() is called domorestuff(), and it is also not having any exception handling so jvm will terminate dostuff() abnormally without executing remaining code and remove from runtime stack.same way main method also will terminated**
* **since main method is called by jvm, so jvm is responsible to handle this exception.**
* **to handle such type of eception jvm will have assistant (or) mechanism called Ddefault exception handler.so jvm will handover the exception object to default exception handler**
* **default exception handler will print the exception messge in console and terminate the program abnormally**
* **format of exception message:**

**Exception in thread "main":AE: devision by zero**

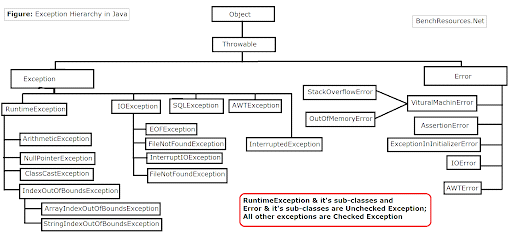
**at Test.domorestuff()**

**at Test.doStuff)**

**at Test.main()**

***4.Exception Heirarchy***

* **Throwable class is root for Exception Heirarchy**
* **most of the time exception are caused by our program only.**
* **Exception are recoverable, which means filenotfoundexception occurs,use local file and continue rest of program normally.it is recoverable**
* **Errors are most of time not caused by our program and non recoverable. eg: out of memory error, errors are caused by lack of system resources, may be memory. hardware problem. virus problem**

****

**Google it for diff hierarchy**

**Checked vs Unchecked Exception**

* **exception is classified as checked and unchecked exception**

**Unchecked Exception:**

* **exceptions which are not checked by compiler whether the programmer handing or not such type of exceptions by default consider as unchecked exception**
* **eg:Arithmetic Exception:division by 0,Nullpoint exception, classcastexception,index bound of exception**
* **RuntimeException and its child classes are unchecked**
* **compiler wort check whether the errors are handled or not.so all errors are called as unchecked exception**

**Checked Exception:**

* **exceptions which are checked by compiler whether the programmer handing or not such type of exceptions by default considered as checked exception**
* **compiler telling there is a chance of exception so first handle it after that only you can run the program**
* **except Error and RuntimeException other things in exception heirarchy are checked exception**
* **eg: Throable, Exception,IOException,EOFException, filenotfound exception,interupted exception, servlet exception, remote exception,**
* **checked exception categerise as fully checked and partially checked exception.**

**Fully Checked :**

* **if both parent and child are checked exception then it is called fully checked( Airport Checking)**
* **IOException (child -EOFException,FileNotFound Exception), InterptedException,ServletException,RemoteException**

**Partially checked Exception :**

* **Parent is checked exception but child is unchecked exception then it is called Partially checked Exception ( Security Checking in Shopping Mall, wont check child)**
* **eg: Exception(some childs(RE) are unchecked), throwable(Error is unchecked,RE is unchecked)**

***5.Customised Exception handling by using try catch***

**EG: without try catch**

**class Test{**

**p.s.v.m(String[] arg){**

**s.o.p("stmnt- 1");**

**s.o.p(10/0);**

**s.o.p("stmnt -3");**

**}**

**}**

**o/p:**

**stmnt -1**

**RE: AE: devision by zero**

**Abnormal Termination and it is not a graceful terminaion we missed stmnt 3.**

**EG 2: with try vatch**

**the code which may rise an exception is called risky code.**

**risky code goes into try block.here stnmt 2 is risky code.**

**inside catch block we will have handing code for risky code**

**class Test{**

**p.s.v.m(String[] arg){**

**s.o.p("stmnt- 1");**

**try{**

**s.o.p(10/0);// stmnt-2**

**}catch(ArithmeticException ae){**

**s.o.p(10/2);**

**}**

**s.o.p("stmnt -3");**

**}**

**}**

**o/p:**

**stmnt-1**

**5 (catch block)**

**stmnt-3**

**graceful termnination(we are not missing anything) and normal termination.**

***6.control flow in try catch block***

**try{**

**stmnt-1;**

**stmnt-2;**

**stmnt-3;**

**}catch(Exception e){**

**stmnt-4;**

**}**

**stmnt-5;**

**case 1: if there is no exception**

**stmnt 1,2,3,5 will be excuted . stmnt 4(inside catch wont executed)**

**Noral Termination**

**case 2: if exception raised at stmnt 2 and corresponding catch block matched**

**stmnt 1 ,4,5 executed**

**Normal Termination**

**inside try block if exception occur , remaining code in try block wont execued. so kept main code in try not normal code.**

**case 3: if an exception raised at stmnt-2 and corresponding catch is not matched**

**it is always abnormal termination**

**o/p: stmnt-1,Abnormal Termninaion**

**case 4:if exception occured in stmnt-4 or stmnt-5**

**those are not part of try block. so it is always abnormal termination**

**we can include try, catch inside catch block also if we feel stmnt-4 will cause exception.**

**o/p: stmnt-1, Abnormal Termnination**

***7. methods to print exception information***

**in throwable class contains 3 methods to print exception information to the console**

1. **e.printStackTrace();**
2. **e.toString();**
3. **e.getmessage();**

**class Test{**

**p.s.v.m(String[] args){**

**try{**

**s.o.p(10/0);**

**}**

**catch(AE e){**

**e.printStackTrace(); //name,description,stacktrace**

**//AE: / by zero**

**// at Test.main(Test.java:7)**

**s.o.p(e);**

**//(or)**

**s.o.p(e.toString()) //name , description**

**// AE : / by zero**

**s.o.p(e.getMessage()); // only description**

**// / by zero**

**}**

**}**

**}**

**defualt exception handle always use e.printStackTrace() to print exception information to the console. this method internally having s.o.p**

**but other mehods return string but not having s.o.p so we have to use s.o.p**

***8.try with multiple catch block***

* **for every exception the way of handling is different. so it is recommend to use different catch block for different exception**

**try{**

**AE;**

**File Not Found;**

**SQL Excetion;**

**}**

**catch(Exception e){**

**}**

**this try catch is not recommended.having same catch block for all type of excption.**

**try{**

**AE;**

**File Not Found;**

**SQL Excetion;**

**}**

**catch(ArithmeticException e){**

**// perform alternate arithmetic operation**

**}**

**catch(FileNotFoundException e){**

**//use local file**

**}catch(SQLException e){**

**//use oracle db instead of mysql db**

**}**

**catch(Exception e){**

**//default handling code**

**}**

**it is recommended. bcz wy of handling is different for different exception**

**try{**

**s.o.p(10/0);**

**}catch(Exception e){**

**s.o.p(Exception");**

**}**

**catch(ArithmeticException e){**

**s.o.p("AE");**

**}**

**C.E throws : Exception j.l.AE has already been caught**

**in try with multiple catch the order of catch block is very important. compulsary we have to take from child to parent not parent to child**

**in above program, AE has caught in Exception catch block itself. forst catch block can catch all the exception. so second catch block never called**

**try{**

**s.o.p(10/0);**

**}**

**catch(ArithmeticException e){**

**s.o.p("AE");**

**}**

**catch(Exception e){**

**s.o.p(Exception");**

**}**

**now if exception occur first compiler checks the first catch block, if it is Arithmetic exception it will handle otherwise second catch block will handle.. so it will successfully.**

**try{**

**s.o.p(10/0);**

**}**

**catch(ArithmeticException e){**

**s.o.p("AE");**

**}**

**catch(ArithmeticException e){**

**s.o.p("AE");**

**}**

**C.E throws bcz same exception we shold not have two catch block.**

***9.finally block***

**open db connection**

**read data**

**close dbconnection -- resource deallocation code, we are deallocating the resource**

**such type of code is called clean up code.**

**try{**

**open db connection**

**read data**

**close dbconnection**

**}**

**catch(Exception e){**

**}**

**write cleanup code inside try block is never recommeded. if exception occurs who will close the db connection.**

**try{**

**open db connection**

**read data**

**}**

**catch(Exception e){**

**close dbconnection**

**}**

**it is also not recommended. if no exception ocurs catch block wont excuted so db connection wont close**

**try{**

**open db connection**

**read data**

**}**

**catch(Exception e){**

**}**

**finally{**

**close dbconnection**

**}**

**finally block will be always executed whether exception occurs or not handle or noteven if it is abnormal trmination also finally block will be executed.**

**the main objective of finally block is to maintian cleanup code. the speciality of finally block is it will executed always**

**case 1: if no exception**

**try{**

**s.o.p("try");**

**}**

**catch(Exception e){**

**s.o.p("catch");**

**}**

**finally{**

**s.o.p("finally");**

**}**

**o/p:**

**try**

**finally**

**case 2: if exception raised and handled**

**try{**

**s.o.p("try");**

**s.o.p(10/0);**

**}**

**catch(Exception e){**

**s.o.p("catch");**

**}**

**finally{**

**s.o.p("finally");**

**}**

**o/p:**

**try**

**catch**

**finally**

**case 3: if an exception raiised but not handled**

**try{**

**s.o.p("try");**

**s.o.p(10/0);**

**}**

**catch(NullPointerException e){**

**s.o.p("catch");**

**}**

**finally{**

**s.o.p("finally");**

**}**

**o/p:**

**try**

**finally**

**Exception in thread main : AE: / by zero**

**abnoramal termination**

**finally VS return statement**

* **inside method if there is return statement there then rest of method wont be executed**

**class Test{**

**p.s.v. m(String[] arrgs){**

**try{**

**s.o.p("try");**

**return;**

**}catch(Exception e){**

**s.o.p(Exception e);**

**}**

**finally{**

**s.o.p("finally");**

**}**

**o/p:**

**try**

**finally**

**whether return is in try or catch first fnally executed before return statment**

**class Test{**

**p.s.v.m(String[] args){**

**s.o.p(m1());**

**}**

**p.s.int.m1(){**

**try{**

**return 777;**

**}catch(Exception e){**

**return 888;**

**}finally{**

**return 999;**

**}**

**}**

**}**

**o/p:**

**999**

**bcz finally block hv highest priority**

**class Test{**

**p.s.v.m(String[] args){**

**s.o.p(m1());**

**}**

**p.s.int.m1(){**

**try{**

**s.o.p("try");**

**system.exist(0);//jvm is shutdown**

**}catch(Exception e){**

**return 888;**

**}finally{**

**return 999;**

**}**

**}**

**}**

**o/p:**

**try**

**there is only one suitivation where the finalyy block wont executed which is system.exit(0);**

**here system.exit(0) is dominatin finally block**

**the argument represent status code**

**0 is normal termination**

**non zero is abnormal termination**

***10.control in try catch finaly***

**try{**

**stmt-1;**

**stmnt-2;**

**stmnt-3;**

**}catch(Exception e){**

**stmnt-4;**

**}**

**finally{**

**stmmnt-5**

**}**

**stmnt-6;**

**case 1:**

**if there is no exception**

**o/p:**

**1, 2, 3,5,6,NT**

**case 2:**

**if there is exception raised at stmnt-2 and catch block is matched**

**o/p: 1,4,5,6,NT**

**case 3:**

**if exception raised at stmnt -2 and catch block not matched**

**o/p:**

**1,5,AT**

**case 4:**

**if an excption raised at stmnt -4**

**AT bcz it is not in try . but finally block will executed**

**cae 5 : exception raised at stmnt -5 or stmnt-6**

**AT, bcz outside try.**

**We can have try,finaly without catch**

***11. Control flow in Nested try-catch-finally:***

* **inside try block we can have try catch finally, inside catch block we can have try catch finally, inside finally block also we can have try catch finally, which means nesting of try catch finally is always possible.**
* **inside try (risky code) we have some very risky code, whiich needs to be in try.**
* **so inside outer try block we have low risky code and innner try block we are having too much risky code. which will be handled so remaining code in outer try will execute even if exception occurs. this kind of suitivation we can have nested try catch**

**class Test{**

**public s v m (string args[]){**

**try{**

**s.o.p("outer block");// 1 line in o/p**

**try{**

**s.o.p("inner try block");/// 2 line in o/p**

**}Catch(AE ae){**

**s.o.p("inner catch block");/// 2 line in o/p**

**}**

**s.o.p("outside of inner try catch block");**

**}Catch(Exception e){**

**s.o.p("outer catch block");**

**}**

**finally{**

**s.o.p("outer finally block");**

**}**

**}**

**}**

**if inner catch block not matched control will comes to outer catch block and outer finaly, outer catch also not matched then AT will happen.**

**Eg:2**

**class Test{**

**p.s.v.m(String[] args){**

**stmn1-1;**

**stmt-2;**

**stmnt-3;**

**try{**

**stmnt-4;**

**stmnt-5;**

**stmnt-6;**

**}**

**Catch(X e){**

**stmnt 7;**

**}**

**finally{**

**stmnt 8;**

**}**

**stmnt -9;**

**}**

**Catcj (Y e){**

**stmnt 10;**

**}**

**finally{**

**stmnt 11;**

**}**

**stmnt-12;**

**Case1:if there is no exception**

**o/p: 1,2,3,4,5,6,8,9,11,12,NT**

**Case 2: IF an exception raised at stmnt-2 and corresonding catch block matched,**

**for stmnt-2 corresponding catch block is outer catch block**

**o/p: 1,10,11,12, NT**

**inner finaly block wont executed.bcz it is related to inner try which we didnt enter.**

**Case 3: if exception raised at stmnt 2 corresponding catch block not matched**

**o/p: 1,11,AT**

**Case 4: Exception raised at stmnt 5 and corresponding inner catch block mathed**

**o/p:1,2,3,4,7,8,9,11,12,NT**

**case 5: exception stmnt5, outer catch block matched and inner catch not matched**

**o/p:1,2,3,4,8,10,11,12,NT**

**we entered already in inner try so before control goes to outer catch finally will be excuter first.**

**case:6**

**Multi-Threading**

**Multi-tasking:**

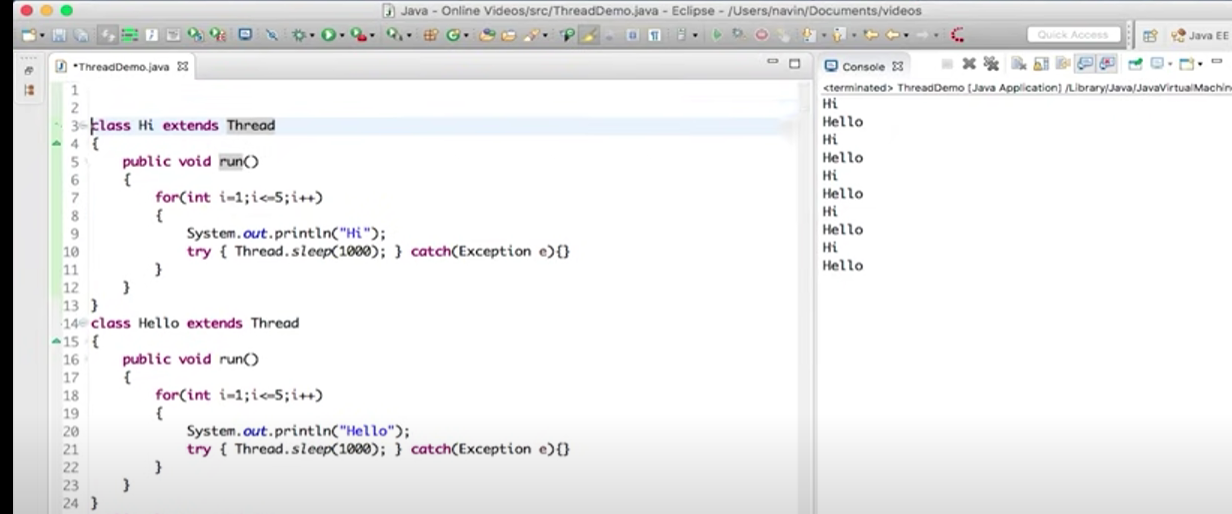
* **multiple task running at a time is known as multi-tasking**
* **threads are sub process of process or unit of process**
* **by default in every java application, we will have at least 1 main thread.**

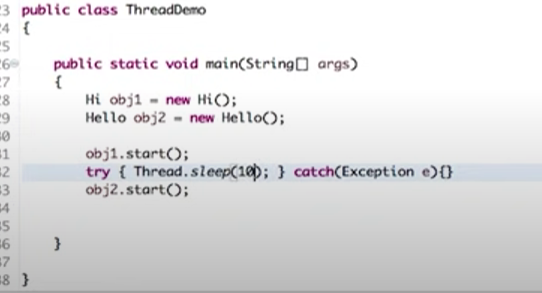
**Multi threading:**

* **The process of executing multiple threads simultaneously is known as multithreading.**
* **All threads of a process share the common memory.**
* **It is save cpu time**
* **So, threads are light-weight processes within a process**

**Ways to create thread class**

1. **By extends thread class**

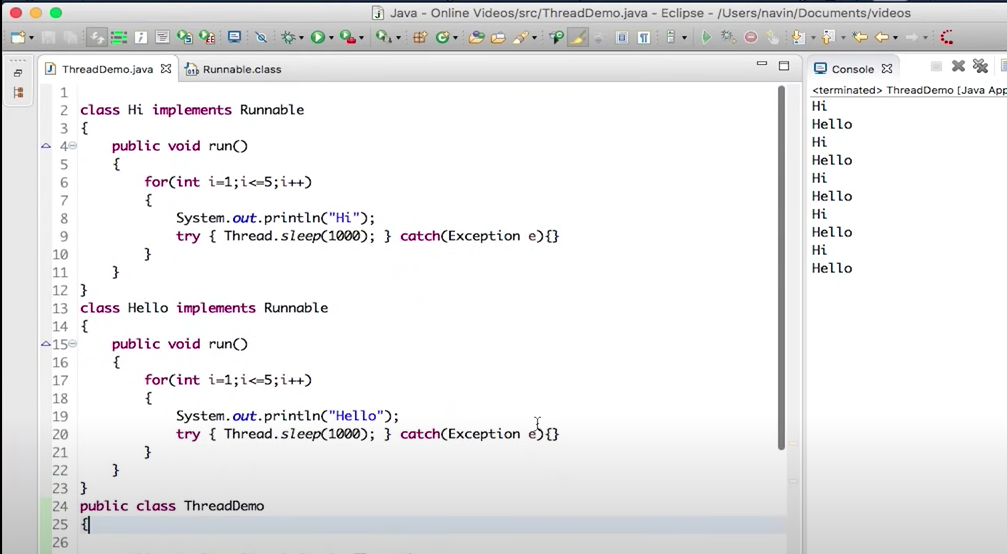
****

****

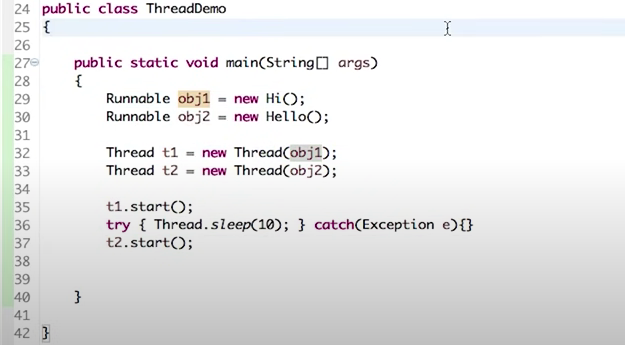
**t.start();**

**without calling start method new thread will not created execution will happen in the same thread only.**

1. **By implement Runnable interface**

****

* **Here we need to create obj for our custom thread class, create thread class obj by using custom class obj, then start the thread like below**

****

**MultiThreading using Lambda Expression**

* **Instead of creating separate thread class we can create anonymous class like below if those objects are used in only one class, so memory will be saved**

**Runnable obj1 = new Runnable(){**

**Public void run(){**

**For(int i-0;i<5;i++){**

**System.out.println(“hi”);**

**}**

**}**

**}**

* **We can rewrite this code like below sing lambda expression, so that number of code reduced**

**Runnable obj1 =()->{**

**Public void run(){**

**For(int i-0;i<5;i++){**

**System.out.println(“hi”);**

**}**

**}**

**}**

**Or**

**Thread t1 = new Thread(()->{**

**Public void run(){**

**For(int i-0;i<5;i++){**

**System.out.println(“hi”);**

**}**

**}**

**}**

**);**

**Thread Class vs Runnable Interface**

1. If we extend the Thread class, our class cannot extend any other class because Java doesn’t support multiple inheritance. But, if we implement the Runnable interface, our class can still extend other base classes.
2. We can achieve basic functionality of a thread by extending Thread class because it provides some inbuilt methods like yield(), interrupt() etc. that are not available in Runnable interface.
3. Using runnable will give you an object that can be shared amongst multiple threads.

**Thread LifeCycle**

**Inter thread communication/Poducer or consumer problem**

**Deadlock**

**Wait**

**sleep**