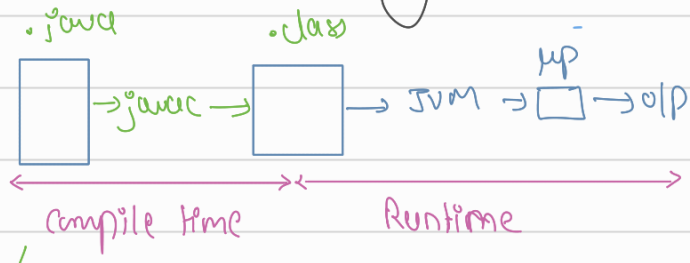
Exception Handling

Exception Handling



Error: coz of type casting mistake, syntax errors

logic errors } runtime errors

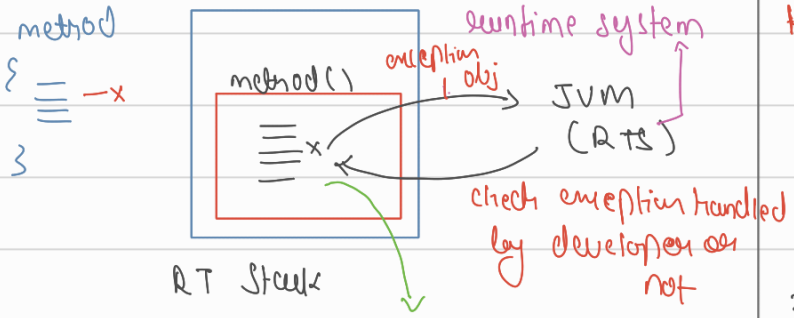
Internet or web

connection b/w devices — service of internet

Exception: Unwanted event or Mistake occur in Runtime which result in abrupt termination of a program.

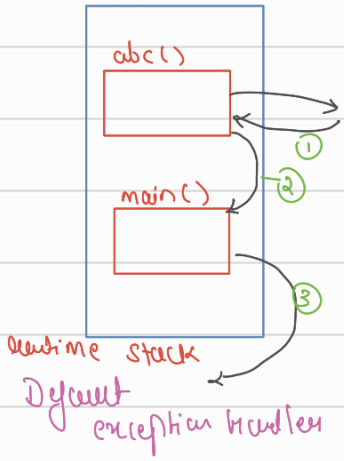
Exception handling:

Try, Catch, throws, throw, finally



Checked Exception	Unchecked Exception
- Compiler give warning for some exception.	- developer responsible of risky stmts.

JVM check exception handling:



① method only
② caller method
③ default exception handler.

⇒ whenever there is an Exception:

1. Handle exception (try - catch - ES)
2. Duck the Exception (throws)
3. Re-throwing an Exception (throw),

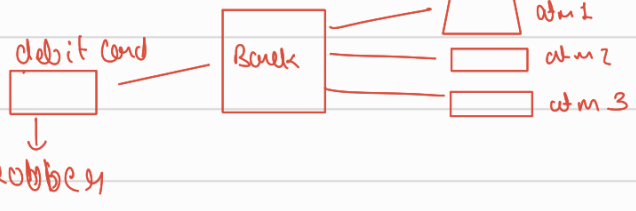
→ giving it to caller() to handle (team lead) (just informing other developers to handle exception!)

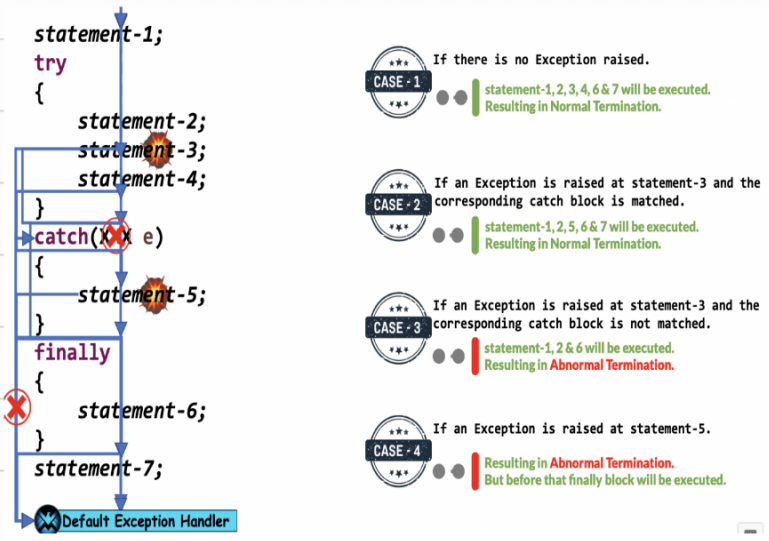
- when to duck? (ignoring)

unchecked exception	checked exception
- don't duck, handle there only.	- you can duck.

3. Re-throwing Exception: Even after handling exception in same method, there are some situations where we need to handle exception by caller() also.

e.g





Differences between throw & throws

throw	throws
throw keyword is used to explicitly throw an exception to the JVM.	throws keyword is used to declare an exception to delegate exception handling responsibility to the caller.
throw keyword is followed by an instance of Throwable or a subclass of Throwable.	throws keyword is followed by exception class name.
throw keyword is used within the method body.	throws keyword is used with the method signature.
Multiple exceptions can't be thrown with a single throw clause.	Multiple exceptions can be declared with a single throws clause.
Used in rethrowing an exception.	Used in both rethrowing and ducking an exception.

try {
}
catch (Exception e) {
throw e;
}
	void test() throws Exception
	{
	
	}

- only for unchecked exceptions.
- both checked & unchecked exceptions.

Try-Catch : To handle exception

Throws : Duck & method signature

Throw : inside catch → (rethrow) (requirement)

finally : close resource.

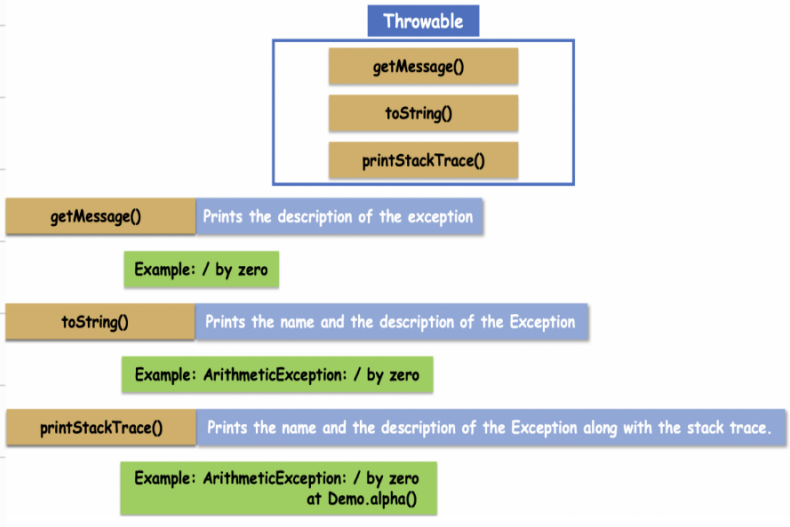
Exception Handling Skeleton

```
try
{
    Risky / Suspicious code
}
catch (Exception e)
{
    Handling / Alternate / Pre Cautionary code
}
finally
{
    Resource deallocation / Clean up code
}
```

What does the Exception Object contain?

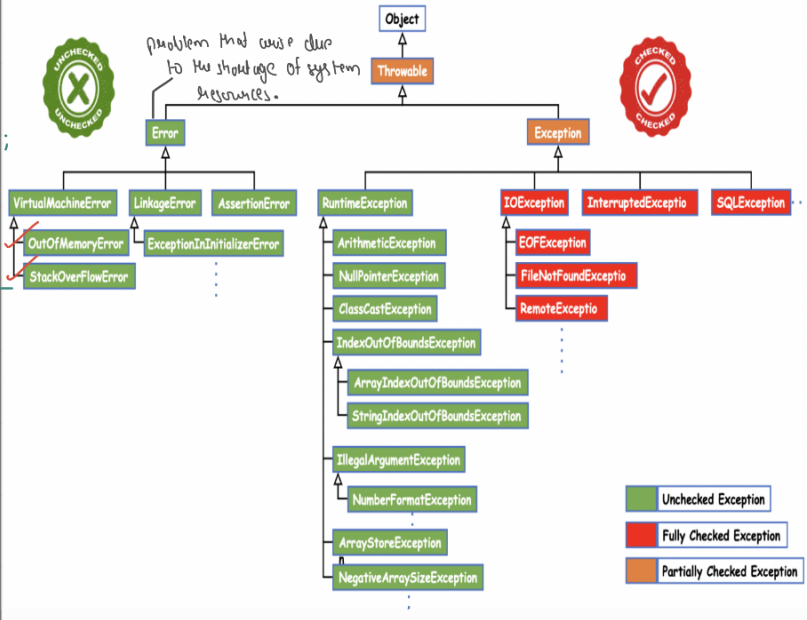
```
class Launch
{
    public static void main(String[] args)
    {
        System.out.println("Connection1 is established");
        Demo d1 = new Demo();
        d1.alpha();
        System.out.println("Connection1 is terminated");
    }
}

class Demo
{
    public void alpha()
    {
        System.out.println("Connection2 is establis
        Scanner scan = new Scanner(System.in);
        System.out.println("Enter the numerator:
        int a = scan.nextInt();
        System.out.println("Enter the denominator
        int b = scan.nextInt();
        int c = a / b;
        System.out.println(c);
        System.out.println("Connection2 is terminated
    }
}
```


Methods to Print Exception Information



int[] a = new int[10];

fn()
{ fn(); }



26 nov

29 November 2022 14:20

```
//Re-throwing an exception
class Alpha1
{
    void alpha() throws ArithmeticException
    {
        System.out.println("Connection to Calc app is established");
        try
        {
        Scanner scan=new Scanner(System.in);
        System.out.println("Enter the first num to divide");
        int num1=scan.nextInt();
        System.out.println("Enter the 2nd num to divide");
        int num2=scan.nextInt();

         int res=num1/num2;

        System.out.println("The res is "+ res);
        }
        catch(ArithmeticException e)
        {
        System.out.println("Exception handled in alpha only");
        throw e;
        }
        finally {
        System.out.println("Connection is terminated");
        }
    }
}
public class LaunchException6 {
    public static void main(String[] args)
    {
        try
        {
        System.out.println("Main method connection");
        Alpha1 a=new Alpha1();
        a.alpha();
        }
        catch(ArithmeticException e)
        {
        System.out.println("Exception handled in main");
        }
    }
}
```

indication to the caller that this method contain risky code (exception) so you have to handle it, despite of handled by some method or thrown by it.

after handling intentionally i'm giving exception obj to caller method.

throw e; ——> it's like return stmt, stmt below this doesn't execute until they are inside finally.

o/p

```
Main method connection
Connection to Calc app is established
Enter the first num to divide
100
Enter the 2nd num to divide
0
Exception handled in alpha only
Connection is terminated
Exception handled in main
```

Synchronous Exceptions

Occurs at a specific program statement.

```
class Launch
{
    public static void main(String[] args)
    {
        String str = null;
        str.toUpperCase();
    }
}
```

OUTPUT

```
java.lang.NullPointerExceptio
n
at Launch.main(Launch.java:6)
```

```
class Launch
{
    public static void main(String[] args)
    {
        int[] a = new int[5];
        a[5] = 10;
    }
}
```

OUTPUT

```
java.lang.ArrayIndexOutOfBoundsException:
at Launch.main(Launch.java:6)
```

Asynchronous Exceptions

Occurs anywhere in the program.

```
class Launch
{
    public static void main(String[] args)
    {
        Scanner scan = new Scanner(System.in);
        System.out.println("Enter your name:");
        String name = scan.next();
        System.out.println("Enter your grades:");
        String grade = scan.next();
    }
}
```

OUTPUT

```
Enter your name:
Sachin
Enter your grades:
CTRL + C
Exception in thread "main" Terminate batch job (Y/N)?
```

Keyboard Interrupt

Examples:-

Keyboard interrupts, "stop" signals, etc.

Execution control flow in Try, Catch & finally

```
try {
    statement-1;
    statement-2;
    statement-3;
    try {
        statement-4;
        statement-5;
        statement-6;
    }
    catch(XXX e) {
        statement-7;
    }
    finally {
        statement-8;
    }
    statement-9;
}
catch(YYY e) {
    statement-10;
}
finally {
    statement-11;
}
statement-12;
```

CASE - 1: If no Exception occurs.
Statements - 1, 2, 3, 4, 5, 6, 8, 9, 11 & 12 will be executed. Resulting in Normal Termination.

CASE - 2: If an Exception occurs at statement-2 and the corresponding catch block is matched.
Statements - 1, 10, 11 & 12 will be executed. Resulting in Normal Termination.

CASE - 3: If an Exception occurs at statement-2 and the corresponding catch block is not matched.
Statements - 1 & 11 will be executed. Resulting in Abnormal Termination.

```
try {
    statement-1;
    statement-2;
    statement-3;
    try {
        statement-4;
        statement-5;
        statement-6;
    }
    catch(XXX e) {
        statement-7;
    }
    finally {
        statement-8;
    }
    statement-9;
}
catch(YYY e) {
    statement-10;
}
finally {
    statement-11;
}
statement-12;
```

CASE - 4: If an Exception occurs at statement-5 and the corresponding inner catch block is matched.
Statements - 1, 2, 3, 4, 7, 8, 9, 11 & 12 will be executed. Resulting in Normal Termination.

CASE - 5: If an Exception occurs at statement-5 and the corresponding inner catch block is not matched, but outer catch block is matched.
Statements - 1, 2, 3, 4, 8, 10, 11 & 12 will be executed. Resulting in Normal Termination.

CASE - 6: If an Exception occurs at statement-5 and both inner and outer catch blocks are not matched.
Statements - 1, 2, 3, 4, 8 & 11 will be executed. Resulting in Abnormal Termination.

```
try {
    statement-1;
    statement-2;
    statement-3;
    try {
        statement-4;
        statement-5;
        statement-6;
    }
    catch(XXX e) {
        statement-7;
    }
    finally {
        statement-8;
    }
    statement-9;
}
catch(YYY e) {
    statement-10;
}
finally {
    statement-11;
}
statement-12;
```

CASE - 7: If an Exception occurs at Statement-7 and the corresponding catch block is matched.
Statements - 1, 2, 3, (Exception may occur at one of the Statements - 4 or 5 or 6), 8, 10, 11, 12 will be Executed with Normal termination.

CASE - 8: If an Exception occurs at Statement-7 and the corresponding catch block is not matched.
Statements - 1, 2, 3, (Exception may occur at one of the Statements - 4 or 5 or 6), 8, 11 will be Executed with Abnormal termination.

```
try {
    statement-1;
    statement-2;
    statement-3;
    try {
        statement-4;
        statement-5;
        statement-6;
    }
    catch(XXX e) {
        statement-7;
    }
    finally {
        statement-8;
    }
    statement-9;
}
catch(YYY e) {
    statement-10;
}
finally {
    statement-11;
}
statement-12;
```

CASE - 9: If an Exception occurs at Statement-8 and the corresponding catch block is matched.
Statements - 1, 2, 3, X, X, X, X (4, 5, 6, 7 optional), 10, 11, 12 will be Executed with Normal termination.

CASE - 10: If an Exception occurs at Statement-8 and the corresponding catch block is not matched.
Statements - 1, 2, 3, X, X, X, X (4, 5, 6, 7 optional), 11 will be Executed with Abnormal termination.

Possible Combinations of try-catch-finally

CASE - 1 — Only try

```
try
{

}
```

✗

CASE - 2 — Only catch

```
catch(XXX e)
{

}
```

✗

CASE - 3 — Only finally

```
finally
{

}
```

✗

CASE - 4 — try - catch

```
try
{

}
catch(XXX e)
{

}
```

✓

CASE - 5 — Reverse order

```
catch(XXX e)
{

}
try
{

}
```

✗

CASE - 6 — Multiple try

```
try
{

}
try
{

}
catch(XXX e)
{

}
```

✗

CASE - 7 — Multiple try

```
try
{

}
catch(XXX e)
{

}
try
{

}
```

✗

CASE - 8 — Multiple try - catch

```
try
{

}
catch(XXX e)
{

}
try
{

}
catch(XXX e)
{

}
```

✓

CASE - 9 — Multiple catch

```
try
{

}
catch(XXX e)
{

}
catch(YYY e)
{

}
```

✓

CASE - 10 — Multiple catch

```
try
{

}
catch(XXX e)
{

}
catch(XXX e)
{

}
```

✗

CASE - 11 — Multi - catch

```
try
{

}
catch(XXX | YYY e)
{

}
```

✓

CASE - 12 — try-catch-finally

```
try
{

}
catch(XXX e)
{

}
finally
{

}
```

✓

CASE - 13 — try - finally

```
try
{

}
finally
{

}
```

Unknown term as no catch().

✓

CASE - 14 — catch - finally

```
catch(XXX e)
{

}
finally
{

}
```

✗

CASE - 15 — Unordered

```
try
{

}
finally
{

}
catch(XXX e)
{

}
```

✗

CASE - 16 — try-multiple catch-finally

```
try
{

}
catch(XXX e)
{

}
catch(YYY e)
{

}
finally
{

}
```

✓

CASE - 17 — Multiple finally

```
try
{

}
catch(XXX e)
{

}
finally
{

}
finally
{

}
```

✗

CASE - 18 — Statements in-between try-catch-finally

```
try
{

}
System.out.println("Hi");
catch(XXX e)
{

}
System.out.println("Hello")
;
finally
{

}
```

✗

CASE - 19 — Only try with resource

```
try (R)
{

}
```

✓

CASE - 20 — try-with-resource -catch-finally

```
try (R)
{

}
catch(XXX e)
{

}
finally
{

}
```

✓

CASE - 24 — Nested try-catch-finally

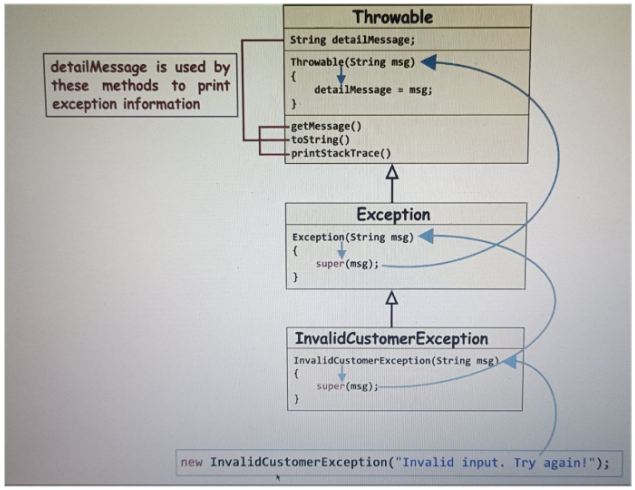
✓

```
try
{
  try
  {
  }
  catch(XXX e)
  {
  }
  finally
  {
  }
}
catch(XXX e)
{
  try
  {
  }
  catch(XXX e)
  {
  }
  finally
  {
  }
}
finally
{
  try
  {
  }
  catch(XXX e)
  {
  }
  finally
  {
  }
```

29 nov Custom(User Define) Exception

30 November 2022 23:47

: to make our own exception, we need to create class make it a child of Exception().




```java
class InvalidCustomerException extends Exception {
    public InvalidCustomerException(String msg) {
        super(msg);
    }
}
class Atm {
    int userid = 1212;
    int password = 1111;
    int pw;
    int uid;
    public void input() {
        Scanner scan = new Scanner(System.in);
        System.out.println("Kindly enter the user id");
        uid = scan.nextInt();
        System.out.println("Kindly enter the pasword");
        pw = scan.nextInt();
    }
    public void verify() throws InvalidCustomerException {
        if ((userid == uid) && (password == pw)) {
            System.out.println("Take your cash");
        } else {
            InvalidCustomerException ice = new InvalidCustomerException("Are you sure? try again bcz wrong input");
            // System.out.println(ice);
            System.out.println(ice.getMessage());
            throw ice;
        }
    }
}
class Bank {
    public void initiate() {
        Atm a = new Atm();
        try {
            a.input();
            a.verify();
        } catch (InvalidCustomerException e1) {
            try {
                a.input();
                a.verify();
            } catch (InvalidCustomerException e2) {
                try {
                    a.input();
                    a.verify();
                } catch (InvalidCustomerException e3) {
                    System.out.println("Oh Choor dude we caught you card is blocked!!");
                    System.exit(0);
                }
            }
        }
    }
}
public class LlaunchCE {
    public static void main(String[] args) {
        Bank b = new Bank();
        b.initiate();
    }
}
```

we can handle it with if else also but

then, - we can't able to throw exception obj. so called did have to write some if-else logic to handle it. again

we are giving 3 chance to use debit- if wrong pin, then it goes to bank & the card will be blocked.

30 nov

02 December 2022 00:22

Exception Handling

==========

1. try with resource.
2. try with multi-catch block.
3. Rules of Overriding associated with Exception.

Remaining topics to be discussed

==========

1. instanceof vs isInstanceOf(Object obj)
2. How to create a userdefined package and in realtime project how it is used?

1.7 version Enhancements

==========

1. try with resource
2. try with multicatch block

untill jdk1.6, it is compulsorily required to write finally block to close all the resources which are open as a part of try block.

```
eg:: BufferReader br=null
 try{
 br=new BufferedReader(new FileReader("abc.txt"));
 }catch(IOException ie){
 ie.printStackTrace();
 }finally{
 try{
 if(br!=null){
br.close();
 }
 }catch(IOException ie){
ie.printStackTrace();
 }
 }
```

Problems in the apporach

==========

1. Compulsorily the programmer is required to close all opened resources which increases the complexity of the program
2. Compulsorily we should write finally block explicitly, which increases the length of the code and reviews readablity.

To Overcome this problem SUN MS introduced try with resources in "1.7" version of jdk

try with resources purpose to remove finally block

==========

In this apporach, the resources which are opened as a part of try block will be closed automatically once the control reaches to the end of try block normally or abnormally,so it is not required to close explicitly so the complexity of the program would be reduced.

It is not required to write finally block explicitly,so length of the code would be reduced and readability is improved.

```
try(BufferedReader br=new BufferedReader(new FileReader("abc.txt")){
    //use br and perform the necessary operation
     //once the control reaches the end of try automatically br will be closed
    }
    catch(IOException ie){
     //handling code
    }
```

Rules of using try with resource

==========

1. we can declare any no of resources, but all these resources should be seperated with ;

```
eg#1.try(R1;R2;R3;){
    //use the resources
    }
```

2. All resources are said to be AutoCloseable resources iff the class implements an interface called "java.lang.AutoCloseable" either directly or indirectly

eg:: java.io package classes, java.sql.package classes

```
public interface java.lang.AutoCloseable {
    public abstract void close() throws java.lang.Exception;
   }
```

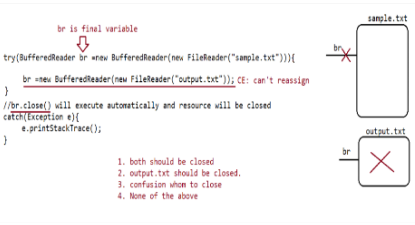
Note: which ever class has implemented this interface those classes objects are refered as "resources".

3. All resource reference by default are treated as implicitly final and hence we can't perform reassignment with in try block.

Jvm make it final, so it cannot be modified.

```
try(BufferedReader br=new BufferedReader(new FileWriter("abc.txt")){
    br=new BufferedReader(new FileWriter("abc.txt"));
     }
```

output::CE: can't reassign a value



4. untill 1.6 version try should compulsorily be followed by either catch or finally, but from

```
// 1.7 version we can take only take try with resources without cath or finally.
try(R){
```

```
//valid
}
```

5. Advantage of try with resources concept is finally block will become dummy because we are not required to close resources explicitly.

6. try with resource nesting is also possible.

```
try(R1){
    try(R2){
    try(R3){
    }
    }
    }
```

MultiCatchBlock

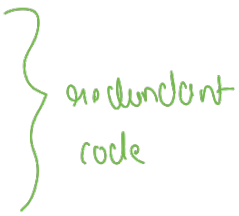
==========

Till jdk1.6, eventhough we have multiple exception having same handling code we have to write a seperate catch block for every exceptions, it increases the length of the code and reviews readability.

logic

====

```
try{
    ....
    ....
    ....
    ....
   }catch(ArithmeticException ae){
   ae.printStackTrace();
   }catch(NullPointerExcepion ne){
   ne.printStackTrace();
   }catch(ClassCastException ce){
   System.out.println(ce.getMessage());
   }catch(IOException ie){
   System.out.println(ie.getMessage());
   }
```



To overcome this problem SUNMS has introduced "Multi catch block" concept in 1.7 version

```
try{
    ....
    ....
    ....
    ....
   }catch(ArithmeticException |NullPointerException e){
   e.printStackTrace();
   }catch(ClassCastException |IOException e){
   e.printStackTrace();
```

→ In multicatch block,there should not be any relation b/w exception types(either child to parent or parent to child or same type) it would result in compile time error.

```
eg:: try{
}catch( ArithmeticExeption | Exception e){
e.printStackTrace();
}
```

Output:CompileTime Error

throw =>handle the exception using catch block and throw it back the exception object to the caller.

throws => method signature and commonly used if the exception is "CheckedException".

CheckedException => compiler will check for the handling code only then compilatin is successfull.

eg: IOException,SQLException,..... are all checkedexceptions.

UnCheckedException => compiler will not check for the handling code,but jvm will come into picture and possiblity of "succesful" or "abnormal" termination.

eg: RunTimeException and its child classes

,Error and its child classes are all "UncheckedException".

Rules of Overriding when exception is involved

==========

While Overriding if the child class method throws any checked exception compulsorily the parent class method should throw the same checked exception or its parent otherwise we will get Compile Time Error.

There are no restrictions on UncheckedException.

eg#1.

See only child to parent.

```
class Parent{
    public void methodOne();
    }
    class Child extends Parent{
    public void methodOne() throws Exception{}
    }
    error: methodOne() in Child cannot override methodOne() in Parent
     public void methodOne() throws Exception{}
    overridden method does not throw Exception
```

as parent doesn't throws exception.

Rules w.r.t Overriding

==========

```
parent: public void methodOne() throws Exception{}
child : public void methodOne()
output: valid

parent: public void methodOne(){}
child : public void methodOne() throws Exception{}
output: invalid

parent: public void methodOne()throws Exception{}
child : public void methodOne()throws Exception{}
output: valid

parent: public void methodOne()throws IOException{}
child : public void methodOne()throws IOException{}
output: valid

parent: public void methodOne()throws IOException{}
child : public void methodOne()throws FileNotFoundException,EOFException{}
output: valid

parent: public void methodOne()throws IOException{}
child : public void methodOne()throws FileNotFoundException,InterruptedException{}
output: invalid

parent: public void methodOne()throws IOException{}
child : public void methodOne()throws FileNotFoundException,ArithmeticException{}
output: valid

parent: public void methodOne()
child : public void methodOne()throws
ArithmeticException,NullPointerException,RuntimeException{}
output: valid

parent: public void methodOne()throws IOException{}
child : public void methodOne()throws Exception{}
output: invalid

parent: public void methodOne()throws Throwable{}
child : public void methodOne()throws IOException{}
output: valid
```

Instanceof

==========

1. We can use the instanceof operator to check whether the given an object is particular type or not.

r instanceof X

r => reference

X => class/interfaceName

eg:

```
ArrayList al =new Arraylist();//inbuilt object where we can keep any type of
other objects
al.add(new Student());//0th position
al.add(new Cricketer());//1st position
al.add(new Customer());//2nd position
Object o=l.get(0); // l is an arraylist object
if(o instanceof Student) {
Student s=(Student)o ;
//perform student specific operation
}
elseif(o instanceof Customer) {
Customer c=(Customer)o;//perform Customer specific operations
 }
```

eg#2.

```
Thread t = new Thread( );
System.out.println(t instanceof Thread);//true
System.out.println(t instanceof Object);//true
System.out.println(t instanceof Runnable); //true
```

Ex :

public class Thread extends Object implements Runnable {}

=> To use instanceof operator compulsory there should be some relation between argument types

(either child to parent Or parent to child Or same type) Otherwise we will get compile time error saying inconvertible types.

eg: String s= new String("sachin");

System.out.println(s instanceof Thread);//CE

Thread t=new Thread();

System.out.println(t instanceof String);//CE // no rel^n b/w classes that's why CE

=> Whenever we are checking the parent object is child type or not by using instanceof operator that we get false.

Object o=new Object();

System.out.println(o instanceof String); //false

Object o=new String("ashok");

System.out.println(o instanceof String); //true

=> For any class or interface X null instanceof X is always returns false

```
System.out.println(null instanceof X); //false
public class Test {
public static void main(String[] args) {
Object t = new Thread();
System.out.println(t instanceof Object);//true
```

compile time : object
Runtime : Thread

```
System.out.println(t instanceof Thread);//true
System.out.println(t instanceof Runnable);//true
System.out.println(t instanceof String);//false
System.out.println(null instanceof Object);//false
}
}
```

always keep type.

always keep reference type.

isInstance()

==========

Difference between instanceof and isInstance() :

instanceof

=========

instanceof an operator which can be used to check whether the given object is particular type or not We know at the type at beginning it is available.

eg: String s = new String("sachin");
System.out.println(s instanceof Object);//true
//If we know the type at the beginning only.

isInstance()

isInstance() is a method , present in class Class , we can use isInstance() method to checked whether the given object is particular type or not We don't know at the type at beginning it is available Dynamically at Runtime.

```
class Test {
    public static void main(String[] args) {
    Test t = new Test( ) ;
    System.out.println(Class.forName(args[0]).isInstance(t));////arg[0] --- We don't know the type
at beginning
    }
    }
    java Test Test //true
    java Test String //false
    java Test Object //true
```