Date
Page

Errors & Exception in Java

~~It~~ No matter how smart we are, errors are our constant comparisons. With practice, we keep getting better at finding & correcting them. There are 3 types of errors in Java.

1) Syntax Errors

2) Logical Errors

3) Runtime errors - also called Exceptions.

Syntax Errors
When compiler finds something ~~wr~~ wrong with our program, it throws a syntax error.

```
int a = 9        // No semi-colon, syntax error
a = a + 3;
d = 4            // Variable not declared
```

~~A~~ A logical Error or a bug ~~occurs~~ occurs when a program compiles & round but does ~~the~~ the wrong thing.
→ Message delivered wrongly
→ Wrong time of chats being displayed

• Incorrect redirects!

Runtime Errors

Java may sometimes call an error while the ~~p~~ program is running. These are also called exceptions!

These are encountered due to ~~x~~ circumstances like bad input and (or) resource constraints.
E.g. user supplies 's'+8 to a program which adds 2 numbers

Syntax errors & logical errors are encountered ~~[illegible]~~ users.

Exceptions in Java

~~An excep,~~
An exception is a event ~~th~~ that occurs when a program is executed disrupting the normal flow of instructions.

There are 2 types of exceptions in Java:
) Checked Exception → Compile time except. (Handled by Comp
) Unchecked Exception → Run-time exceptions

Commonly Occuring ~~Ex~~ Exceptions
Following are the few commonly occ. exceptions in Java:

1) Null pointer exception
2) Arithmetic exception
3) Array Index Out Of Bound Exception
4) IllegalArgument Exception
5) NumberFormat Exception

<u>try-catch</u> <u>block</u> <u>in</u> <u>Java</u>

In Java, ~~ex~~ exceptions are managed using try-catch block

Syntax:

```
        try {
            // code to try
          }


        catch(Exception e) {
            // code of exception

          }
```

Handling specific exceptions
In Java, we can handle specific exceptions by typing multiple catch blocks.

```
try{
    //code
    }
~~catch(IO E~~
catch(IOException e)
 {                  //Handles all IO
   // code              exceptions
  }
catch(ArithmeticException e)
  {          //Handles all arithmetic
 // Code          exceptions
   }
catch(Exception e){
     // code    //Handles all other
    }                exceptions
```

Nested Try-Catch block:

```
try{
    try{

        }
    catch(Ex. e){

             }
   }
catch(Ex. e){

        }
```

The Exception class in Java

We can write our custom Exceptions using the Exception class in Java

```
    public class MyException extends
                                Exception
       {
         // Over-ridden methods
       }
```

The Exception class has the foll. imp. methods:

1) String toString() → executed when sout(e) is ran

2) void printStackTrace() → prints Stack Trace

3) String getMessage() → prints the Exception message

E.g.
```
import java.util.Scanner;

class MyException extends Exception{
    @Override
    public String toString(){
        return "I am toString()";
    }
@Override
  public string getMessage(){
    return "I am getMessage()";
      }
}

class MaxAgeException extends Exception{
   @Override
public string tostring(){
   return "Age cannot be greater than 125";
      }

  @Override
~~pub~~ public string getMessage(){
   return "Make sure the value of age
              entered is correct";
     }
}
```

Main

```
~~pub~~ public class ~~cwh~~ {
    p  s  v  m( ) {
    int a;
Scanner sc= new Scanner(System.in);
 a = sc.nextInt();
   ~~if(~~ if (a<9) {
         try{
   throw new Arithmetic Exception("This is an
                                exception");
         }
catch(Exception e){
   sout(e.getMessage ());
   sout(e.tostring ());
e.printStackTrace();
  sout("Finished");
  }
  sout("Yes Finished");

 }
}
}
```

O/P
```
4
This is an exception
java.lang.ArithmeticException: This is an exception
 "        "        "        "
at Main.main(Main.java:33)
Finished
Yes Finished
```

Throw Keyword

The throw keyword is used to throw an exception explicitly by the programmer.

Syntax:

```
throw new MyException("Exception throw");
```

Throws Keyword

The Java throws keyword is used to declare an exception. This gives an information to the programmer that there might be an exception so it's better to be prepared with a try catch block!

Syntax:

```
public void calculate(int a, int b) throws
                    IOException{

        // code

        }
```

E.g.

```
class NegativeRadiusException extends
               Exception{
  @Override
public String ~~to~~ toString (){
     return "Radius cannot be negative";

     }
  @Override

public String getMessage(){

     return "Radius cannot be negative";

     }
}
```

```
~~pub~~ public class Main {

public static double area(int r) throws
        NegativeRadiusException {

    if(r<0) {
        throw new NegativeRadiusException();
    }
double result = Math.PI * r * r;
    return result;
}


~~pub~~ public static int divide(int a, int b)
        throws ArithmeticException{

    ~~s~~ ~~sut~~ return a/b;
    }

p s   v m(      ) {

    try {

        int c = divide(6, 0);
        sout("Result of Div. " + c);

    double ar = area(6);
    sout("Area = " + ar);

    }

  catch(Exception e){
        sout("Exception");
        }

} } }
```

~~Ex~~ o/p 1:
lines x & y not commented
Exception.

o/p 2:
lines x & y commented:

113.09733...

Finally block in Java

Finally block contains the code which is always executed whether the exception is handled or not
It is used to execute code containing instructions to release the system resources. close a connection etc.

```
Ex public class finally{

  public static int greet(){
      try{
          int a=50;
          int b=10;
          int c=a/b;
          return c;
      }
    catch(Exception e){
          sout(e);
      }
finally {
    sout("Cleaning up resources... this is
              the end of this function");
    }
  return -1;
}

p   s   v   m  (         ) {

  int k = greet();
   sout(k);
  int a = 7;
  int b = 9;
  while(true){
     try {
        sout(a/b);
        }
     catch(Exception e){
       sout(e);
          break;
         }
finally {
sout("I am finally value of b=  " +b);
  }
  b--;

 try {
    sout(50/3);
   }
 finally{
   ~~sout~~ sout("Yes");
        }
     }
 }
```