Exception handling & multithreading

* Exception handling

Exception- The error does not caught by compiler.

Exception

checked   Unchecked

- Detect during compilation   - detect during execution

↓

ex:-

```
BufferReader. br = new BufferReader (InputstreamReader
                                        (system.in))
        br.readline();
```

- Exception is class in java.

① checked exception-
- The exception that are checked at compilation time. by the java compiler are called checked exceptions.

② - The exception that are checked by JVM are called unchecked exception.

checked exception:

ex:-

```
Public static void main( String args[]) throws
                                    IoException.
{
    BufferReader br = new BufferReader (InputStream
                        Reader (system in))
        br. readline();
}
```

Unchecked exception ex:-

divide by zero:

```
int a=100, b=0, c;
c = a/b;
```

*

$$\begin{tikzcd}
 & \text{java.lang} \arrow[d] & & \\
 & \text{Throwable} \arrow[dl, dash] \arrow[dr, dash] & & \\
\text{Error} & & \text{Exception} \arrow[dl, dash] \arrow[dr, dash] & \\
 & \text{checked} \arrow[d] & & \text{unchecked} \arrow[d] \\
 & \text{At compiletime} & & \text{At execution time}
\end{tikzcd}$$

Syntax:-

```
try
{



}
catch( IOException e)
{                   object
                      exception can be display with
}                         e.printStackTrace();
```

ex:-
```
Class Exp
{
    public static void main( String args[])
    {
        try {
            int arr[]= {10,12};
        S.o.p("open files");
        int n= args.length();
        S.o.p ("n="+n);
        int a = 10/n;
        S.o.p ("a=" +a);
         int c= arr[5];
     }
   } catch ( ArithmeticException ae)
     {
            S.o.p( ae); // ae.printstacktrace();
     }
   finally   // keyword use to end the main() if
                  theire is any error caught/or not-
                                          exception
     {
        S.o.p("class files");
     }
  }
}
catch( ArrayIndexOutOfBound ab)
{
     S.o.p( ab);
}
```

throw

throws- use to throw ~~exptie~~ exception explicitly outside the program.

① Throws-clause-

- If the programmer is not handling runtime exception java compiler may not give any error related to runtime exception.
- But the rule is that the programmer should handled checked exception.

- In this case the programmer can use throws clause to throw the exception explicitly.
ex:-

Throw clause-
- Throw clause is used in the try block for transferring the control to catch block.
ex:-

```
class Sample
{
    public static void ~~main~~ demo( )
    {
        try {
                S.o.p( "try block");
        throw new NullPointerException(
                                   "Exception data");
        },
        catch( NullPointerException ne)
        {
            S.o.p(ne);
        }
    }
class Throw demo
{
    P.S.V.M( )
    {
        Sample demo( ); // function call by
                            class name because
                        demo function is static.
    }
}
```

- Throw clause is use in software testing to test whether program is handling all the exceptions according to the programmer.

$$\begin{tikzcd}
 & \text{* Built in exceptions :-} \arrow[dl] \arrow[dr] & \\
\text{checked} & & \text{unchecked}
\end{tikzcd}$$

① Unchecked exception-
- detected during runtime.

① ArithmaticException
② ArrayIndexOutOfBound
③ ArrayStoreException
④ ClassCastException
⑤ EnumConstantNotPresentException
⑥ IllegalArgumentException.
⑦ IllegalMonitorStateException.
⑧ IllegalStateException
⑨ NegativeArraySizeException
⑨ Nullpointer Exception.
⑩ NumberFormatException.
⑪ SecurityException
⑫ StringIndexOutOfBoundException.
⑬ TypeNotPresent Exception.
⑭ UnsupportedOperationException.

ex:

* Null pointers

```
① class NullPointer
{
      Public static void main( )
      {
         try {
            String a = null;
            s.o.p ( a.charAt(0));
          }
          Catch( NullPointerException ne)
          {
              s.o.p(" string is Null");
          }
    }
}
② class  String Index
   {
      p.s.v.m(   )
      {
          try{
              String  a = "Hello";
              char c =   a.charAt(10));
              s.o.p(c);
          }
        Catch( String Index Out Of Bound  e)
          {
              s.o.p( "string out of bound");
          }
      }
}

③ class  Noformat
  {
          public static void main()
          {
               try{
                   int no = Integer.parseInt ("No");
            s.o.p(no);
        }
        catch(NumberformatException e)
        {
            s.o.p("No Format Exception");
        }
      }
   }
}
```

* User defined exception :-

```
class MyException extends Exception
{
        Private static int accno[] = { 1001, 1002,
                                   1003, 1004, 1005 };
        Private static string name[] = { "abc", "xyz",
                          "pqr", "lmp", "stq" };
        Private static double bal[] = { 1000.0.2,
              2000.05, 3000.06, 4000.01, 5000.08 }
        MyException()
        {
        }
        MyException (str)
        {
            super (str);
        }
}
~~class~~
  Public static void main( string args[])
  {
     try {
      s.o.pl( "Accno : Name : bal");
      for( i=0;  i<5 ; i++)
      {
          s.o.p ( accno[i] + name[i] +bal[i]);

      if( bal [i] < 1000)
      MyException me= new MyException ( "less amount
          throw me;
      }
    }
  }
  catch( MyException me)
  {
      me. printStractionTrace ( );
  }
}
```

* Nestea try :-

```
class NestTry {

    public static void main( String args[])
    ~~try~~ {
        try
        {
            int a= args. length;
            int b = 42/a;
            s.o.p(" a=" + a );
          try {
            if( a==1)
              a= a/a-a;
            if( a==2)
            {
                int c[] = {1};
                int c[42]= 45;
            }
        }
        catch ( ArrayIndexOutOfBoundException e)
        {
            s.o.p (e);
        }
    }
    catch( ArithmeticException e)
    {
        s.o.p( " divide by 0 "+e);
    }
  }
}
```

* Multithreading :-

```
Class current
{
    p.s.v.m ( String args [])
    {
        s.o.p(" current thread execution");
        Thread t= Thread. currentThread();
        s.o.p(t);
        s.o.p( "Name is =" + getName());
    }
}
OUTPUT: Thread [Main, 5, Main]
        Name is= main
```

• Thread is light weight resource.

• Creating thread

$$\begin{tikzcd} & \text{thread} \arrow[dl, dash] \arrow[dr, dash] & \\ \text{extend} & & \text{Implement Runnable} \end{tikzcd}$$

```
Class Myclass implement Runnable
class Myclass extends Thread
{
        public void run()
        {
        }
}
Myclass obj = new Myclass();
Thread t= new Thread (obj);
t.start();
```

```
→
class Mythread extends Thread
{
        public void run()
        {
            for( int i=0 ; i<1000;i++)
            {
                s.o.p(i);
                boolean stop = false;
                if(stop == true)
                  return;
            }
}       }
class Demo
{
      p.s.v.m(      )
    {
      MyThread obj = new MyThread();
      Thread t=new Thread( obj);
            t.start();
    }
}
```

Assignment:
Write a java program that <u>currently</u> <u>implements</u> producer, consumer problem using concept of interthread communication.

```
class Communicate
{
      public static void main( String []args)
    {
        Producer obj1=new Producer();
        Consumer obj2 new consumer();
       Thread t1= new Thread (obj1);
        Thred t2 = new Thread (obj2);
          t2. Start();
          t1. start();
    }
}

class  Producer  extends  Thread
 {
         String Buffer sb;
        boolean dataprodoveq =false;
        Producer()
        {
             sb= new StringBuffer();
        }
       public void run()
      {     synchronised (sb) {
         For( int i=1; i<=10; i++)
         {
             try {
                   sb. append (i+":");
                   Thread. Sleep (100);
                   s.o.p ("Appending");
             } catch( Exception e)
             {
             }
         ~~dataprodover = true;~~
           sb.noitfy()
        }  }
      }
  }
```

Suspend execution of thread till the time reach.

```
class Consumer extends Thread
{
        Producer prod;
        Consumer (Producer prod);
        {
              this.prod = prod;
        }
      Public void run()
      {    synchronized (prod.sb) {
           try {
                  prod.wait();
              while (!= prod.dataprodover)
                Thread. sleep(100);
           } catch ( Exception e)
             {

             }
        s.o.p (prod.Sb);
      }
}
```

- obj. notify() methods realises and object and sends a notification to a waiting thread. that object is available.

- obj. notifyall() is uesful to send notification to all waiting threads at once that object is available.

- obj. wait() method is makes a thread wait for the object till it receives a notification from notify method or notify all method.

* Thread priorities :-

- range : 0 to 10 (min: 0, max: 10) 5 - default priority

- Priority assigned to thread is done by JVM/programmer

- Variable use to find priority :-

```
public static int Min-priority;
                  MAX-priority;
                  NORM-priority;
public final void setpriority(int priorityNO) // set priority to thread.

public final int getpriority()
            t1. getpriority()   // by default we get 5.
```