```
/*

Exception: An unwanted unexpected event that disturb normal flow of the program is called
exception
        --> It is highly recommended to handle exceptions.
        --> The main objective of exception hanlding is normal termination of the program.

Exception Handling:
        --> Exception handling does not mean repairing an exception.
        --> We have to define alternative way to conitune rest of the program normally.
        --> This way of "Defining alternative" is nothing but exception handling.

--> Throwable acts as a root for exception hierarchy.
--> Throwable class contains the two child classes: Exception and Error

Exception: Most of the cases exceptions are caused by our program and these are recoverable
.
Error: Most of the cases errors are not caused by our programs these are due to lack of
system resources and these are not recoverable.

--> There are two types of Exceptions: Checked and Unchecked Exceptions

Note: Whether exeption is checked or unchecked compulsory it should occur at runtime only.
        --> There is no chance of occuring any exception at compile time.

Examples of Exceptions:
    (1) RuntimeException
            - ArithmeticException
            - NullPointerException
            - IndexOutOfBoundsException
            - IllegalArgumentException
            - ClassCastException
            - IllegalStateException
    (2) IOException
            - EOFException
            - FileNotFoundException
    (3) SQLException
    (4) ServletException
                            and Many more.........


Exception Handling by try catch:

    --> It is highly recommended to handle exceptions.
    --> In our program the code which may cause an exception is called risky code we have
to place risky code inside try block
    --> and the corresponding handling code inside catch block.


Note:
    1. Within the try block if anywhere an exception raised then rest of try block won't be
 executed even though we handled that exception.
        -> Hence we have to place only risky code inside try and length of the try block
should be as less as possible.

 */

package ExceptionHandling;
```

File - /Users/cm_mac/LJIET/2023/Java 2 LJU 2023/JAVA2Notes/src/ExceptionHandling/EH1.java

```java
public class EH1 {
    public static void main(String[] args) {
        System.out.println("Hello");
        // Without try catch
        System.out.println(5 / 0);

        /*
         * with try catch
         *
         * try{
         * System.out.println(5 / 0);
         * } catch (ArithmeticException e) {
         * System.out.println(10 / 2);
         * }
         *
         */

        System.out.println("World");
    }

}
```

```java
/*

Various Methods to print Exception Information:

--> Throwable class defines the following methods to print exception information to the
console.

1. printStackTrace():
    --> This method prints exception information in the following format:
        Name of the exception: Description of the exception
        Stack trace

2. toString():
    --> This method prints exception information in the following format.
        Name of the exception: Description of the exception

3. getMessage():
    --> This method returns only description of the exception
        Description

NOTE: Default exception handler internally uses printStackTrace() method to print exception
 information to the console.

 */

package ExceptionHandling;

public class EH2 {
    public static void main(String[] args) {
        int a[] = { 1, 2, 3, 4 };

        try{
            for (int i=1; i<=4;i++){
                System.out.println(a[i]);
            }
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Output of printStackTrace Method: ");
            e.printStackTrace();

            System.out.println("Output of toString Method: ");
            // System.out.println(e.toString());
            System.out.println(e);

            System.out.println("Output of getMessage Method: ");
            System.out.println(e.getMessage());

        }
    }
}
```

```java
/*

try with multiple catch blocks:

--> The way of handling an exception is varied from exception to exception
    hence for every exception raise a separate catch block that is try with multiple catch
blocks is possible and
    recommended to use.

 */
package ExceptionHandling;

public class EH3 {
    public static void main(String[] args) {
        try {

            //int c = 5 / 0;

            int a[] = { 1, 2, 3, 4 };
            for (int i = 1; i <= 4; i++) {
                System.out.println(a[i]);
            }
            // int c = 5 / 0;

        } catch (ArithmeticException e) {
            System.out.println(e);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println(e);
        }
        System.out.println("Last statement Executed");
    }
}
```

```java
/*

try with multiple catch blocks:

--> If try with multiple catch blocks presents then order of catch blocks is very important
.
--> It should be from child to parent.
--> By mistake if we are taking from parent to child then we will get compile time error (C
.E.) saying
    "exception xxxx has already been caught".

 */

package ExceptionHandling;

public class EH4 {
    public static void main(String[] args) {
        try {

            // int c = 5 / 0;

            int a[] = { 1, 2, 3, 4 };
            for (int i = 1; i <= 4; i++) {
                System.out.println(a[i]);
            }
            // int c = 5 / 0;

        } catch (Exception e) {
            System.out.println(e);
        }
        // catch (ArrayIndexOutOfBoundsException e) { // Here we used child after parent so
, C.E: Unreachable catch block
        //                                           // for ArrayIndexOutOfBoundsException
. It is already handled by the
        //                                           // catch block for Exception
        //     System.out.println(e);
        // }
        System.out.println("Last statement Executed");
    }
}
```

```
/*

Finally Block:

    --> It is never recommended to take clean up code inside try block because there is
        no guarantee for the execution of every statement inside a try block.
    --> It is never recommended to place clean up code inside catch block because
        if there is no exception then catch block won't be executed.
    --> We require some place to maintain clean up code which should be executed always
        irrespective of whether execution raised or not raised and whether handled or not
        handled such type of place is nothing but "finally block".
    --> Hence, Main objective of finally block is to maintain cleaup code.

    Syntax:

        try{
            // risky code
        } catch (X e){
            // handling code
        } finally {
            // cleanup code
        }
 */

package ExceptionHandling;

public class EH5 {
    public static void main(String[] args) {

        try {
            System.out.println("Try block executed");
            //System.out.println(10/0);
        } catch (ArithmeticException e) {
            System.out.println("Catch block executed");
            System.out.println(e);
        } finally {
            System.out.println("Finally block executed");
        }

    }
}
```

```java
/*

Finally Block:

    --> Even though return present in try or catch blocks, first finally will be executed.
    --> And after that only return statement will be considered.
    --> "finally" dominates "return statement".

 */

package ExceptionHandling;

public class EH6 {
    public static void main(String[] args) {
        try{
            System.out.println("try block executed");
            return;
        } catch (ArithmeticException e) {
            System.out.println("catch block executed");
        } finally {
            System.out.println("finally block executed");
        }

    }
}
```

```java
/*

Finally Block:

    --> If return statment is present in try, catch and finally block  then finally block return statement will be considered.
    --> There is only one situation where the finally block won't be executed, i.e., whenever we are using System.exit(0) method.

 */

package ExceptionHandling;

public class EH7 {
    public static void main(String[] args) {
        System.out.println(m1());
    }

    public static int m1() {
        try{
            System.out.println(5/0);
            return 111;
        } catch (ArithmeticException e) {
            return 222;
        } finally {
            System.out.println("finally executed");
            return 333;
        }
    }
}
```

```java
/*

throw Statement:

    --> To create exception object explicitly, one can use throw statement.
    --> By using throw statement, we can hand over exception to the JVM manually.
    --> The result of following 2 examples will be exactly the same.

 */

package ExceptionHandling;

/*
    In this case creation of ArithmeticException object and handover to the JVM will be
performed
    automatically by the main() method.

 */

public class EH8 {
    public static void main(String[] args) {
        System.out.println(5 / 0);
    }
}

/*
    In this case we are creating exception object explicitly and handover to the JVM
manually.
 */

class EH8_1 {
    public static void main(String[] args) {
        throw new ArithmeticException("/ by Zero by CVM");
    }
}

/*
    CASE 1:
        If Exception object refers null then we will get NullPointerException.
*/

class EH8_2_C1 {

    static ArithmeticException e = new ArithmeticException(); // R.E: ArithmeticException

    public static void main(String[] args) {
        throw e;
    }
}

class EH8_3_C1 {

    static ArithmeticException e; // R.E.: NullPointerException

    public static void main(String[] args) {
        throw e;
    }
}
```

```java
/*
    CASE 2:
        After throw statement we can't take any statement directly otherwise we will get C.
E. saying
        "Unreachable code"
 */


class EH8_4_C2 {
    public static void main(String[] args) {
        System.out.println(5 / 0); // R.E. ArithmeticException
        System.out.println("Statement 1");
    }
}

class EH8_5_C2 {
    public static void main(String[] args) {
        throw new ArithmeticException("/ by zero by CVM");
        // System.out.println("Statement 1"); // C.E. Unreachable code
    }
}

/*
    CASE 3:
        We can use throw keyword only for Throwable types otherwise we will get C.E. saying
        "incompatible types"
 */

class EH8_6_C3 {
    public static void main(String[] args) {
        // throw new EH8_6_C3(); // C.E.
        // Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        // No exception of type EH8_6_C3 can be thrown; an exception type must be a
subclass of Throwable
    }
}

class EH8_7_C3 extends RuntimeException {
    public static void main(String[] args) {
        throw new EH8_7_C3(); // R.E.: Exception in thread "main" ExceptionHandling.
EH8_7_C3
                              // at ExceptionHandling.EH8_7_C3.main(EH8.java:96)
    }
}

class EH8_8_C3 extends Throwable {
    public static void main(String[] args) {
        // throw new EH8_8_C3(); // C.E.: Unhandled Exception type EH8_8_C3
    }
}

class EH8_9_C3 extends Exception {
    public static void main(String[] args) {
        // throw new EH8_9_C3(); // C.E.: Unhandled Exception type EH8_9_C3
    }
}
```

```java
/*
    throws Statement:
        --> In program if there is chance of raising checked exception then, compulsory we
should handle it,
        --> Either by try.... catch or by throws keyword,
        --> Otherwise we will get C.E.


 */

package ExceptionHandling;


public class EH9 {
    public static void main(String[] args) {
        // Thread.sleep(3000); // C.E.: Unhandled exception type InterruptedException
    }
}

/*
    We can handle above C.E. by using following 2 ways.
        1. try...catch
        2. throws keyword
 */

class EH9_1 {
    public static void main(String[] args) {
        try {
            Thread.sleep(3000);
            System.out.println("Statement 1");
        } catch (InterruptedException e) {
            System.out.println(e);
        }
    }
}

class EH9_2 {
    public static void main(String[] args) throws InterruptedException {
        Thread.sleep(3000);
        System.out.println("Statement 1");
    }
}

/*

1. Main objective of "throws keyword" is to delicate the responsibility of exception
handling to the caller method.
2. "throws" keyword required only for checked exceptions.
3. "throws" keyword required only to convience compiler.
4. Usage of "throws" keyword does not prevent abrnormal termination of the program.
5. We can not use "throws keyword with classes and interfaces"

*/

class EH9_3 {
    public static void main(String[] args) throws InterruptedException {
        m1();
    }
```

```java
    static void m1() throws InterruptedException {
        m2();
    }

    static void m2() throws InterruptedException {
        m3();
    }

    static void m3() throws InterruptedException {
        Thread.sleep(3000);
        System.out.println("Statement 1");
    }
}

// In above program, if we are removing at least one throws keyword then the program will
give C.E.

/*
    CASE 1:
        We can use throws keyword only for Throwable types otherwise we will get compile
time error
 */

class EH9_4_C1{
    public static void main(String[] args) /* throws EH9_4_C1 */ {
        // C.E.: No exception of type EH9_4_C1 can be thrown; an exception type must be
        // a subclass of Throwable
    }
 }

 class EH9_5_C1 extends Throwable /*(Exception, RuntimeException also possible)*/ {
    public static void main(String[] args) throws EH9_5_C1 {

    }
 }

 /*
    CASE 2:
  */

 class EH9_6_C2 {
    public static void main(String[] args) /* throws Exception */  {
        // throw new Exception(); // C.E.: Because Exception is partially checked class,
hence requires Handling
        // throw new Throwable(); // C.E.: Because Throwable is partially checked class,
hence requires Handling

        // Solution is main method throws Exception.
    }
}

class EH9_7_C2 {
    public static void main(String[] args) {
        throw new Error(); // R.E.
    }
}

/*
    CASE 3:
```

```
        In our program, if there is no chance of rising an exception then we can' right
catch block for that exception
        otherwise we will get C.E. saying exception xxxx is never thrown in body of
corresponding try statement.
        --> This rule is applicable only for "fully checked exception".
 */

class EH9_8_C3 {
    public static void main(String[] args) {
        try{
            System.out.println("Statement 1");
        }catch(Exception e){ // Exception is partially checked
            System.out.println(e);
        }
    }
 }

 class EH9_9_C3 {
     public static void main(String[] args) {
         try {
             System.out.println("Statement 1");
         } catch (ArithmeticException e) { // ArithmeticException is unchecked exception
             System.out.println(e);
         }
     }
 }

 class EH9_10_C3 {
     public static void main(String[] args) {
        //  try {
        //      System.out.println("Statement 1");
        //  } catch (java.io.IOException e) { // C.E.:
        //      System.out.println(e);
        //  }
     }
 }
 // IOException is fully checked, Hence C.E.
 // Unreachable catch block for IOException. This exception is never thrown from the try
statement body

 class EH9_11_C3 {
     public static void main(String[] args) {
    //      try {
    //          System.out.println("Statement 1");
    //      } catch (InterruptedException e) { // C.E.
    //          System.out.println(e);
    //      }
    }
 }


 // InterruptedException is fully checked, Hence C.E.
 // Unreachable catch block for InterruptedException. This exception is never thrown from
the try statement body

 class EH9_12_C3 {
     public static void main(String[] args) {
         try {
             System.out.println("Statement 1");
```

```
        } catch (Error e) { // Error is unchecked.
            System.out.println(e);
        }
    }
}
```

```
/*
    Customized Exceptions (User defined Exceptions):

        --> We can create our own exception to meet our programming requirements.
        --> Such type of exceptions are called customized exceptions.

Note: It is highly recommended to maintain our customized exceptions as unchecked by
extending RuntimeException.
    --> We can catch any Throwable type including Errors also.

 */

class TooYoungException extends RuntimeException {
    TooYoungException(String s) {
        super(s);
    }
}

class TooOldException extends RuntimeException {
    TooOldException(String s) {
        super(s);
    }
}

class MarriageB {
    public static void main(String[] args) {
        int age = 0;
        try{
            age = Integer.parseInt(args[0]);
            if (age > 60) {
                throw new TooYoungException("Please wait some more time......You will get
best match");
            } else if (age < 18) {
                throw new TooOldException("Your age already crossed...No chance of getting
Married");
            } else {
                System.out.println("You will get match details soon by mail");
            }
        }
        catch (ArrayIndexOutOfBoundsException e) {
            System.out.println(e);
        }
    }
}
```