Example

- InSufficientFundsException
- TooYoungException
- TooOldException

Eg#1

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

public class CustomizedExceptionDemo{
 public static void main(String[] args){
 int age=Integer.parseInt(args[0]);
 if(age>60){
     throw new TooYoungException("please wait some more time.... u
will get best match");
        }
 else if(age<18){
  throw new TooOldException("u r age already crossed....no chance of
getting married");
 }
 else{
  System.out.println("you will get match details soon by email");
 }
  }
}
```

Output:

java CustomizedExceptionDemo 61

Exception in thread "main" TooYoungException: please wait some more time....
u will get best match at CustomizedExceptionDemo.main(CustomizedExceptionDemo.java:21)

java CustomizedExceptionDemo 27

You will get match details soon by email

java CustomizedExceptionDemo 9

Exception in thread "main" TooOldException: u r age already crossed....no chance of getting married at CustomizedExceptionDemo.main (CustomizedExceptionDemo.java:25)

Note:

It is highly recommended to maintain our customized exceptions as unchecked by extending **RuntimeException.**

throws statement

In our program if there is a chance of raising a checked exception then we should handle either by **try catch** or by **throws** keyword otherwise the code won't compile.

Eg#1

```
import java.io.*;
class Test3{
 public static void main(String... args){
  PrintWriter pw=new PrintWriter("abc.txt");
  pw.println("Hello world");
 }
}
```

CE: unreported exception java.io.FileNotFoundException; must be caught or declared to be thrown

Eg#2

```
class Test3{
 public static void main(String... args){
  Thread.sleep(3000);
 }
}
```

CE: unreported exception java.lang.InterruptedException; must be caught or declared to be thrown

We can handle this compile time error by using the following 2 ways

1. using try catch
2. using throws keyword

1. using try catch

```
class Test3{
 public static void main(String... args){
  try{
   Thread.sleep(5000);
  }catch(InterruptedException ie){}
 }
}
```

Output:: compiles and successfully runs.

2. using throws keyword

```
class Test{
 public static void main(String... args) throws InterruptedException{
  Thread.sleep(5000);
 }
}
```

Output:: **compiles and successfully runs.**

- we can use throws keyword to delegate the responsibility of exception handling to the caller method.
- The caller method is responsible for handling the exception.

Note

- Hence the main objective of the "throws" keyword is to delegate the responsibility of exception handling to the caller method.
- throws keyword required only for checked exception. usage of throws keyword for unchecked exceptions there is no use.
- "throws" keyword required only to convince compiler.Usage of throws keyword does not prevent abnormal termination of the program.
- Hence recommended to use try-catch over throws keyword.

Eg#1

```
class Test{
 public static void main(String... args) throws InterruptedException{
  doWork();
 }
 public static void doWork() throws InterruptedException{
  doMoreWork();
 }
 public static void doMoreWork() throws InterruptedException{
  Thread.sleep(5000);
 }
}
```

In the above code, if we remove any of the throws keyword it would result in "CompileTimeError".

Case studies of Throwable

Case 1:

- we can use throws keyword only for Throwable types otherwise we will get a compile time error.

```
class Test3{
 public static void main(String... args)throws Test3{
 }
}
```

Output:: **Compile Time Error,Test3 cannot be Throwable**

```
class Test3 extends RuntimeException{
 public static void main(String... args)throws Test3{
 }
}
```

Output:: **Compiles and run successfully**

Case 2:

```
public class Test3 {
 public static void main(String... args) {
  throw new Exception();
 }
}
```

Output::
Compile Time Error
unreported Exception must be caught or declared to be thrown

```
public class Test3 {
 public static void main(String... args) {
  throw new Error();
 }
}
```

Output::
RunTimeException
Exception in thread "main" java.lang.Error at Test3.main(Test3.java:4)

Case 3:

- In our program within the try block,if there is no chance of raising an exception then we can't write a catch block for that exception, otherwise we will get a Compile Time Error saying "exception XXX is never thrown in the body of the corresponding try statement".
- But this rule is applicable only for fully checked exceptions only.

Eg#1

```
public class Test3
{
 public static void main(String... args) {
  try
  {
   System.out.println("hiee");
  }
  catch (Exception e)
  {
  }
 }
}
```

Output:: hiee

Eg#2

```
public class Test3
{
 public static void main(String... args) {
  try
  {
   System.out.println("hiee");
  }
  catch (ArithmeticException e)
  {
  }
 }
}
```

Output:: hiee

Eg#3

```
public class Test3
{
 public static void main(String... args) {
  try
  {
   System.out.println("hiee");
  }
  catch (java.io.FileNotFoundException e)
  {
  }
 }
}
```

Output:: Compile time error(fully checked Exception)

Eg#4

```
public class Test3
{
 public static void main(String... args) {
  try
  {
   System.out.println("hiee");
  }
  catch (InterruptedException e)
  {
  }
 }
}
```

Output:: Compile time error(fully checked Exception)
exception InterruptedException is never thrown in the body of the corresponding try statement.

Eg#5

```
public class Test3
{
 public static void main(String... args) {
  try
  {
   System.out.println("hiee");
  }
  catch (Error e)
  {
  }
 }
}
```

Output:: hiee

Case 4:
we can use throws keyword only for constructors and methods but not for classes.

Eg#1

```
class Test throws Exception    //invalid
{
 Test() throws Exception{ //valid

 }
 methodOne() throws Exception{ //valid

 }
}
```

Note:
Exception handling keywords summary

- try => maintain risky code
- catch=> maintain handling code
- finally=> maintain cleanup code
- throw => To hanover the created exception object to JVM manually
- throws=> To delegate the Exception object from called method to caller method.

Various compile time errors in ExceptionHandling

1. Exception XXXX is already caught
2. Unreported Exception XXXX must be caught or declared to be thrown.
3. Exception XXXX is never thrown in the body of the corresponding try statement.
4. try without catch,finally
5. catch without try
6. finally without try
7. incompatible types : found : xxxx / required : Throwable
8. unreachable code