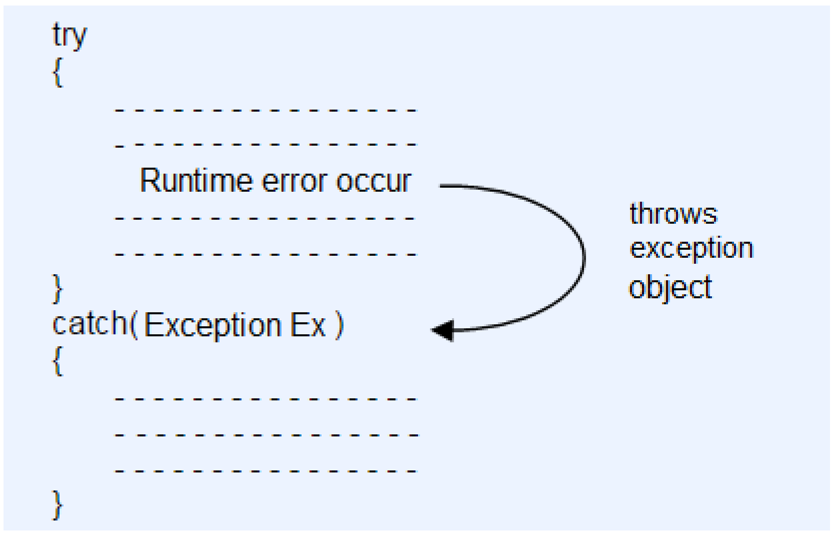
exception and error

Errors are the conditions which cannot get recovered by any handling techniques. It surely cause termination of the program abnormally.

Exceptions are the conditions that occur at runtime and may cause the termination of program. But they are recoverable using try, catch and throw keywords. Exceptions are divided into two catagories : checked and unchecked exceptions.

Checked exceptions like IOException known to the compiler at compile time while unchecked exceptions like ArrayIndexOutOfBoundException known to the compiler at runtime

general form of exception handling block.



```
try
{
     - - - - - - - - - - - - - - - -
     - - - - - - - - - - - - - - - -
       Runtime error occur
     - - - - - - - - - - - - - - - -
     - - - - - - - - - - - - - - - -
}
catch( Exception Ex )
{
     - - - - - - - - - - - - - - - -
     - - - - - - - - - - - - - - - -
     - - - - - - - - - - - - - - - -
}
```

```
public class ExceptionEg {

   public static void main(String[] args)
   {
      int a = 5, b = 0;

      // Attempting to divide by zero
      try {
         int c = a / b;
      }
      catch (ArithmeticException e) {
         e.printStackTrace();
      }
   }
}
```

Different keywords used by Java to handle exception.

Java provides five keywords to support exception handling.

- **Try :** The try block contain statements which may generate exceptions.
- **Catch :**The catch block defines the action to be taken, when an exception occur.
- **Throw :** When an exception occur in try block, it is thrown to the catch block using throw keyword.
- **Throws :** Throws keyword is used in situation, when we need a method to throw an exception.
- **Finally :** If exception occur or not, finally block will always execute.

built in exceptions in Java. (System defined)

Built-in exceptions are the exceptions which are available in Java libraries. These exceptions are suitable to explain certain error situations. Below is the list of important built-in exceptions in Java.

Examples of Built-in Exception:

1. **Arithmetic exception :** It is thrown when an exceptional condition has occurred in an arithmetic operation.

```
// Java program to demonstrate
// ArithmeticException
class ArithmeticException_Demo {
public static void main(String args[])
    {
        try {
            int a = 30, b = 0;
            int c = a / b; // cannot divide by zero
            System.out.println("Result = " + c);
        }
        catch (ArithmeticException e) {
            System.out.println("Can't divide a number by 0");
        }
    }
}
```

checked and unchecked exceptions.

Checked: are the exceptions that are checked at compile time. If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using *throws* keyword.

Unchecked are the exceptions that are not checked at compiled time. It is up to the programmers to be civilized, and specify or catch the exceptions.
In Java exceptions under *Error* and *RuntimeException* classes are unchecked exceptions, everything else under throwable is checked.

user-defined exceptions.

In java we have already defined, exception classes such as ArithmeticException, NullPointerException etc. These exceptions are already set to trigger on pre-defined conditions such as when you divide a number by zero it triggers ArithmeticException, In the last tutorial we learnt how to throw these exceptions explicitly based on your conditions using throw keyword.

In java we can create our own exception class and throw that exception using throw keyword. These exceptions are known as **user-defined** or **custom** exceptions..

nested try.

The try block within a try block is known as nested try block in java.

Sometimes a situation may arise where a part of a block may cause one error and the entire block itself may cause another error. In such cases, exception handlers have to be nested.

Syntax:

```
try
{
   statement 1;
   statement 2;
   try
   {
      statement 1;
      statement 2;
   }
   catch(Exception e)
   {
   }
}
catch(Exception e)
{
}
```

Example:

```
class Excep6{
 public static void main(String args[]){
  try{
    try{
     System.out.println("going to divide");
     int b =39/0;
    }catch(ArithmeticException e){System.out.println(e);}

    try{
    int a[]=new int[5];
    a[5]=4;
    }catch(ArrayIndexOutOfBoundsException e){System.out.println(e);}

    System.out.println("other statement);
  }catch(Exception e){System.out.println("handeled");}

  System.out.println("normal flow..");
 }
}
```

multiple catch statements

A **single try statement** can have multiple catch statements. Execution of particular catch block depends on the type of exception object thrown by the method. If the exception object is of ArithmeticException class, catch block with ArithmeticException will get execute. If no corresponding catch block found, run-time error will occur.

throw and throws.

The throw keyword in Java is used to explicitly throw an exception from a method or any block of code. We can throw either checked or unchecked exception. The throw keyword is mainly used to throw custom exceptions.

Syntax:

throw *Instance*
Example:
throw new ArithmeticException("/ by zero");

throws is a keyword in Java which is used in the signature of method to indicate that this method might throw one of the listed type exceptions. The caller to these methods has to handle the exception using a try-catch block.

Syntax:
type method_name(parameters) throws exception_list
exception_list is a comma separated list of all the
exceptions which a method might throw.

Finally keyword

Java finally block is a block that is used *to execute important code* such as closing connection, stream etc.

Java finally block is always executed whether exception is handled or not.

Java finally block follows try or catch block.

- Finally block in java can be used to put "cleanup" code such as closing a file, closing connection etc.