## Topics

- Use of finally clause in Exception Handling
- Some Important Facts About Exceptions

## Use of finally clause (statement)

- finally block in general used to perform certain house keeping operations such as closing files or releasing system resources.
- finally block may be added immediately after try block or after the last catch block.
- finally block when present is guaranteed to execute regardless of whether an exception is thrown or not.
- If required , finally block can be used to handle any exception generated within a try block.

# finally clause Syntax

```
// Immediately After try() block
try
{
        ……………………..
        ……………………..
} // End of try
finally
{
        ……………………..
        ……………………..
} // End of finally
catch(Exception-Type-1 e) { … }
catch(Exception-Type-2 e) { … }
                  …
catch(Exception-Type-N e) { … }
```

```
// After the last catch() block
try
{
        ……………………..
        ……………………..
} // End of try
catch(Exception-Type-1 e) { … }
catch(Exception-Type-2 e) { … }
                  …
catch(Exception-Type-N e) { … }
finally
{
        ……………………..
        ……………………..
} // End of finally
```

# finally clause Example

```
// File Name ExceptionDemo.java
class ExampleFinallyClause
{
            public static void main(String args[])
            {
                        int a=10;
                        int b = 20;
                        try // Outer Try
                        {
                                    int b1=Integer.parseInt(args[0]);
                                    int x = a/(a-b1);
                                    try          // Inner try
                                    {
                                                int y = b/(b-b1);
                                    } // End of Inner try
                                    finally
                                    {
                                    System.out.println("Inner Block executed");
                                    } // End of finally clause
                        } // End of Outer try
                        finally
                        {
                                    System.out.println("Outer Block executed");
                        } // End of finally clause
            } // End of main() Method
} // End of class
```

## finally clause Example …

**java ExampleFinallyClause**

```
Outer Block executed
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0 [Partial Output
Shown]
```

**java ExampleFinallyClause 45**

```
Inner Block executed
Outer Block executed
```

**java ExampleFinallyClause 10**

```
Outer Block executed
Exception in thread "main" java.lang.ArithmeticException: / by zero [Partial Output Shown]
```

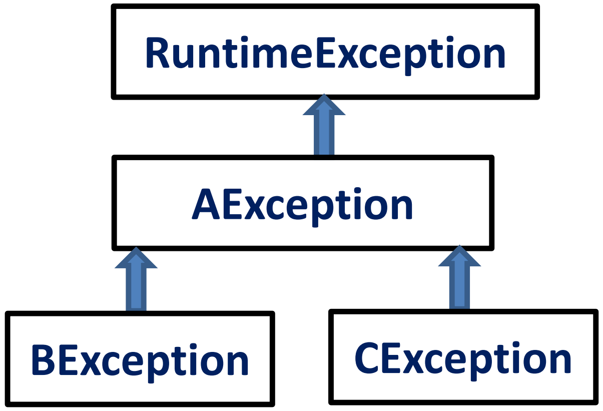
**java ExampleFinallyClause 20**

```
Inner Block executed
Outer Block executed
Exception in thread "main" java.lang.ArithmeticException: / by zero [Partial Output Shown]
```

# Some Important Facts About Exceptions

- Fact I : A super class exception type can catch all sub class exceptions. So, while writing catch blocks , catch sub class exceptions first and then super class exceptions

```
class AException extends RuntimeException           {          }
class BException extends AException                 {          }
class CException extends AException                 {          }
class Demo
{
        public static void main(String args[])
        {
                try
                {
                        int a=10;
                }
                catch(AException e) {          }
                catch(BException e) {          }
                catch(CException e) {          }
        } // End of method
} // End of class
```



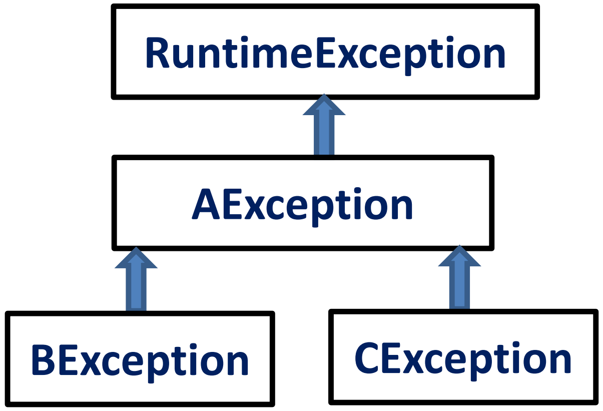
Compile Time Errors

```
exception BException has already been caught
catch(BException e) {}
^
exception CException has already been caught
catch(CException e) {}
2 Errors
```

# Some Important Facts About Exceptions

- Fact I : A super class exception type can catch all sub class exceptions. So, while writing catch blocks , catch sub class exceptions first and then super class exceptions

```
class AException extends RuntimeException        {          }
class BException extends AException              {          }
class CException extends AException              {          }
class Demo
{
        public static void main(String args[])
        {
                try
                {
                        int a=10;
                }
                catch(BException e) {          }
                catch(CException e) {          }
                 catch(AException e) {          }
        } // End of method
} // End of class
```



**NO ERROR**

# Some Important Facts About Exceptions

- Fact II : An overridden method in sub-class can not throw an exception broader and stronger than the method of the super class

```
// File Name : ExceptionDemo.java
import java.io.*;
class A
{
        public void display() throws IOException
        {
        }
}
class B extends A
{
        public void display() throws Exception
        {
        }
}
```

**Compile-Time Error**

```
ExceptionDemo.java:11:
display() in B cannot
override display() in A;
overridden method does
not throw
java.lang.Exception
        public void display()
throws Exception
                      ^
1 error
```

## Some Important Facts About Exceptions

- Fact II : An overridden method in sub-class can not throw an exception broader and stronger than the method of the super class

```
// File Name : ExceptionDemo.java
import java.io.*;
class A
{
        public void display() throws RuntimeException
        {
        }
}
class B extends A
{
        public void display() throws IOException
        {
        }
}
```

**Compile-Time Error**

```
ExceptionDemo.java:11: display() in B cannot override display() in A;
overridden method does not throw java.io.IOException
        public void display() throws IOException
                    ^

1 error
```

innovate achieve lead

# Thank You