Exception Handling

Exception: when unwanted, unexpected event that occurs during the execution of a program and it interrupts the normal flow of program is called exception.

- Handling such a exception and bringing program to normal flow of execution is called exception Handling.
- All the exceptions occurs at runtime while syntax errors occur at compile time.

- Some common problems which may cause exception

(1) creating array objed with negative size.
(2) Accessing index of array which is not available
(3) dividing an integer value with zero
(4) invoking instance members with null reference.
(5) opening a file which is not on specified location.
(6) Reading/writing file without access mode provided.

Example:

```
class ExceptionExample {

    psvm(-)
    {
        SOP ("main started");
        SOP ("execution started");
        SOP ("divide a number by zero :" + (10/0));
        SOP ("execution continue....");
        SOP ("main End");

    }
}
```

O/P ↵

Exception in Thread "main" : java.lang.AE : / zero
at ExceptionExample.main()

Name of Exception
description
stack trace.

Default Exception:

When no exception Handling mechanism written Program will create exception object and handover that exception to JVM.

Exception object consist of
1. Name of exception
2. discription of exception
3. location of exception (stack trace)

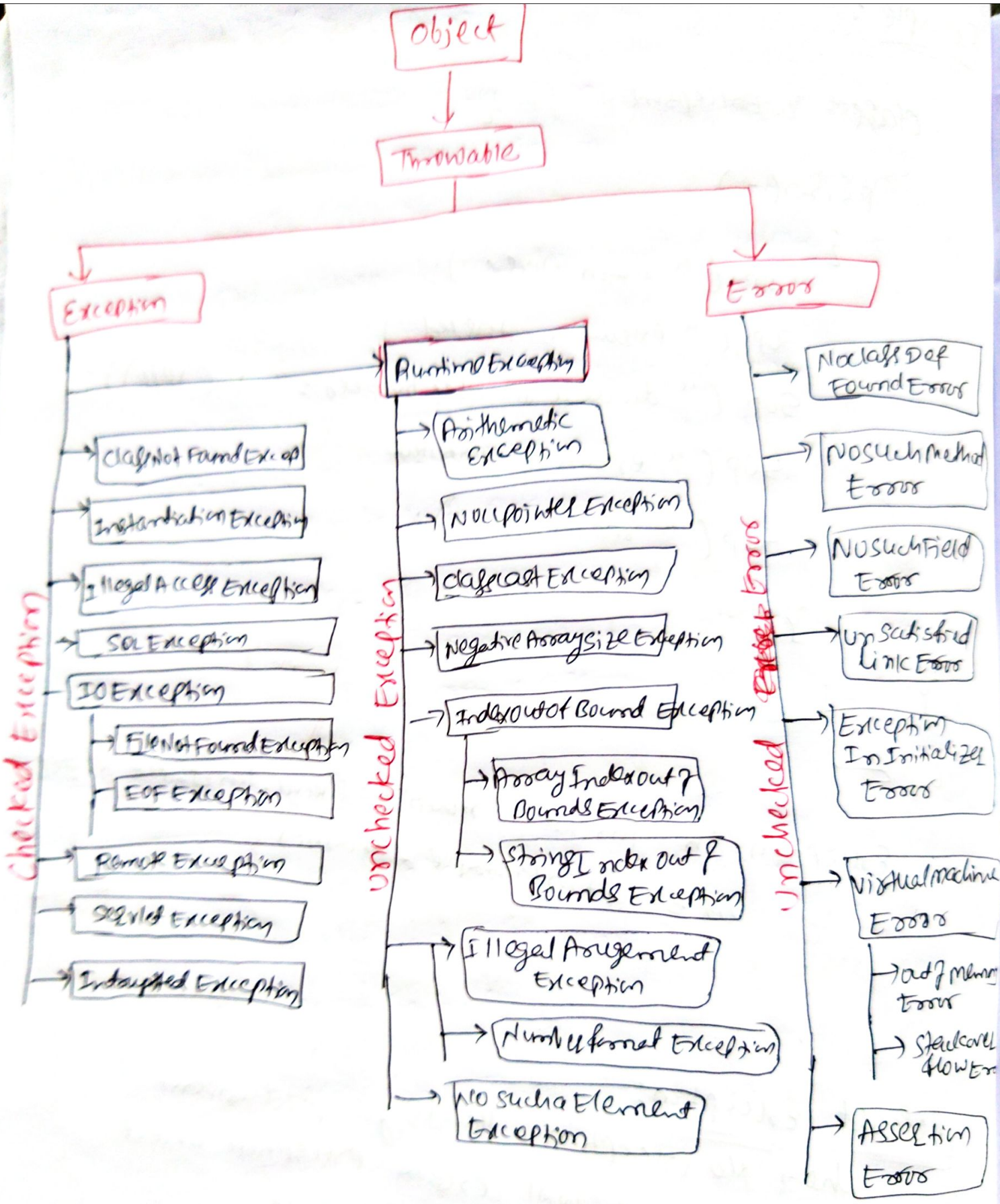


Fig. class structure of Exception in Java

* Exception: Exceptions are caused by our programs and are recoverable.
  . exception classes are subclass of java.lang.Exception class

* Error: . Error occur due to lack of system resource
  . Error are non-recoverable
  . Error are always unchecked
  . Error classes are subclasses of java.lang.Error class

* Types of Exception:
  (1) Checked Exception
  (2) Unchecked Exception
  (3) Partially checked Exception
  (4) Fully checked Exception.

(1) Checked Exception (Compile time Exception): Exception that are checked by compiler at compile time for smooth execution of program are called checked exception.

Example: FileNotFound, SQLException, IOExcepting EOFException, Interupted Excepting

(1) checked exception are handled using try and catch
(2) checked Exception are Handled using throws

(2) Unchecked Exception:

- Runtime Exception.
- Exception that are not checked during compile time by compiler, it's responsibility of Programmers to Handle and code to Handling those.
- Extends java.lang.RuntimeException.
- java.lang.RuntimeException and its subclasses are unchecked exception
- Example : • RuntimeException
  • RuntimeException all subclasses
  • Error class and its subclasses.

(3) Partially checked:

A checked exception is said to be Partially checked iff some of its child class are unchecked.

Ex: Exception, Throwable

(4) Fully checked:

A checked exception is said to be fully checked iff it's childclasses are checked only.

Ex: IOException, InterruptedException.

Note: whether exception is checked or unchecked it always occurs at runtime only.

Methods from Throwable class

Methods

(1) Public void String getmessage();
- Access the message avalible with Exception.

(2) Public Throwable getcause();
- Access the reason of the Exception if avalible.

(3) Public void PrintStackTrace();
- Print the stack trace to the default output (console)

(4) public void PrintStackTrace(PrintStream Ps);
- Print stack trace on File (PrintStream)

(5) Public void PrintStackTrace(PrintWriter Ps);
- Print stack trace on file (PrintWriter)

(6) Public void initCause(Throwable cause);
- Initialize the reason of the exception.

Note:
* Throwable is the superclass for Exceptim classes and Error classes. So all Exceptim can access these methods.

Handling Exception In program

We can handle the exception using following ways

1. try and catch Block
2. throws keyword.

1. Handling Exception using try-catch Blocks

we have to put risky code inside the try block and corresponding handling code we have to put inside the catch Block.

```
try {
    //Risky code
}
catch (xxxx e)
{
    //handling code
}
```

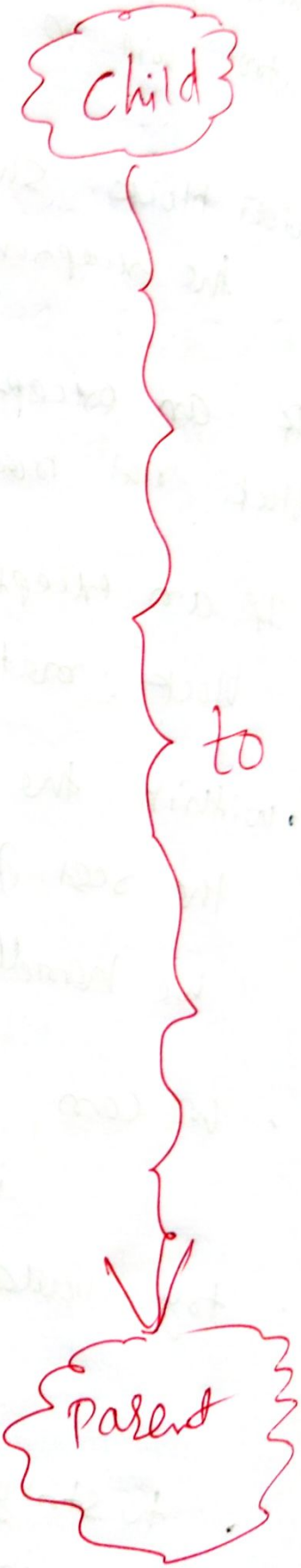
Example

```
m1()
{
  try {
      sop(20/0);
      }
  catch (AE e)
  {
  e.printStackTrace();
  }
}
```

- Try Block should be followed by zero or more catch Blocks
- when exception is raised by try Block statements then control will be transferred to the corresponding catch Block.
- catch block should contain the statements to handle the exception raised by the try Block
- If an exception raised & corresponding catch block not matched Abnormal Termination
- If an exception raised & corresponding catch block matched normal Termination.
- within the try block if any where exception raised the rest of try Block won't be executed Eventhough we handled that, so keep only risky code in try.
- we can Nest (try-catch) Blocks in try also in catch Block.
- try should contain immediate catch block followed no intermediate code.
- to string() method is used to Bind the stack information of Exception and also getmessage(), Printstacktrace()

* try with multiple catch

```
try {
    ≡≡≡
}
catch (AE e)
    {
    ≡≡≡
    }
catch (FileNotFoundException e)
    {
    ≡≡≡
    }
catch (NullPointerException e)
    {
    ≡≡≡
    }
Catch (Exception e)
    {
    ≡≡≡
    }
```



Catching multiple Exception Types

- New feature in Java 7
- using this single catch block can handle more than one type of exception.
- It can reduce code duplication.
- when you writing multiple exception in one catch block then
  - * multiple exception should be unique
  - * multiple exception should not have any inheritance relationship.

Example:

```
try {
    ===
}
Catch (ArrayIndexoutOFBoundException | NumberFormat
       Exception | ArithmeticException e)
{
    ===
}
```

Finally Block

- used to write cleanup code
- Ex: IOstream close
  db connection close
- executes always irrespective of whether exception raised or not or handled or not
- It is not recommended to define clean up code other than finally Block

```
try {
    // risky code
}
catch (exception e)
{
    // Handling code
}
finally {
    // cleanup code
}
```

- finally not get executes when JVM shutdown, System.exit(0);
- if there is any return statement present in try or catch before return the finally will execute then return.

• only one finally Block for one try Block.

Example:

```
SOP("main started");
try {
    int res = 10/0;
    SOP(res);
}
catch (NumberFormatException e)
{
    SOP("inside catch");
}
finally {
    SOP("finally block")
}
SOP("main completed");
```

• Note

(1) try with finally OK

(2) try without finally, without catch Not OK

(3) catch without try Not OK

(4) finally without try Not OK

(5) intermediate code between try and catch Not OK

(6) catch Blocks should follow class relationship from child to Parent

(7) only one finally Block for one try Block