Exception Handling

Types of errors

- Compilation time error

  Run time errors :

-Due to incorrect input of end user or

-Due to incorrect logic of programmer

- General errors :

Traditional Error handling

- By using if---- else conditional statement we can write error handlers

```
 if(condition)
{
}
else
{ if(condition)
      {
      }
}
```

What is an Exception ?

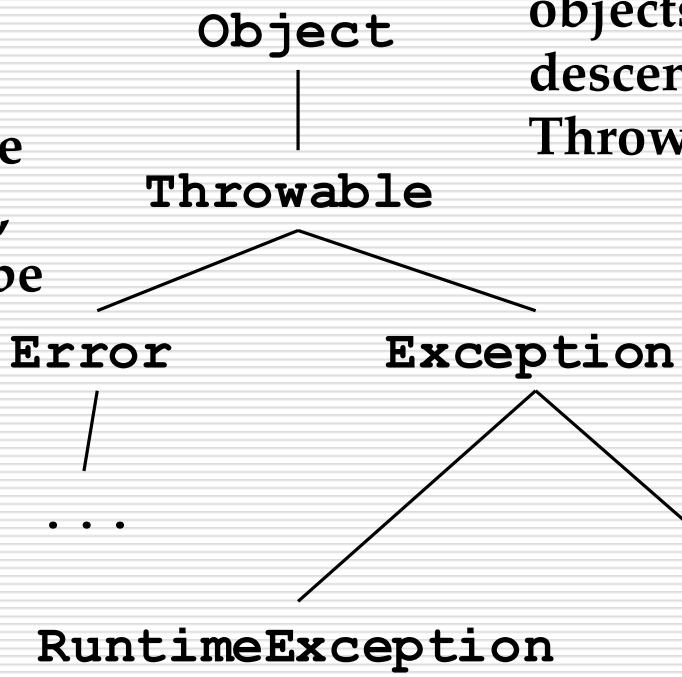
- Once code is compiled and running, it will have to face the real world of erroneous input, inexistent files, hardware failure… Such problems are commonly known as *runtime errors*, which are likely to cause the program to abort.

- It is therefore important to anticipate such problems and handle them correctly, by avoiding loss of data or premature termination, and by notifying the user.

Exception Hierarchy

Classes derived from Error are used for serious, system-generated errors, like OutOfMemoryError, that usually cannot be recovered from

Java will only throw objects of a class descended from Throwable



Classes derived from RuntimeException are used for ordinary system-generated errors, like ArithmeticException

Classes derived from Exception are used for ordinary errors that a program might want to catch and recover from

The Exception Class Hierarchy

- Java distinguishes between two types of error conditions :
  - Errors represents internal errors of the Java run-time system. You won't throw any of those, and you can't do much about them when they happen.
  - Exceptions represent errors in the Java application program – written by you.
    - Because the error is in your program, you are expected to handle the exceptions.
      - Try to recover, if possible
      - Minimally, enact a safe and informative shutdown.

The Exception Class Hierarchy

1. RuntimeException : (unchecked exception)
   - Usually (but not only) errors resulting from programming mistakes,
     - ArrayIndexOutOfBoundsException,
     - NullPointerException
   - Do not need to be advertised or caught.
   - It's your fault : Could have prevented it with an if () clause.

2. Checked exceptions:
   - File operations
   - Must be advertised and caught

Some Predefined Exceptions

Java Exception	Code to Cause It
NullPointerException	String s = null; s.length();

ArithmeticException	int a = 3;
	int b = 0;
	int q = a/b;

ArrayIndexOutOfBoundsException	int[] a = new int[10]; a[10];
ClassCastException	Object x = new Integer(1); String s = (String) x;
StringIndexOutOfBoundsException	String s = "Hello"; s.charAt(5);

Try– catch Block

- Try block consists of a set of executable statements that can possibly throw exception while executing them.
- Try block is followed by one or more catch blocks where exception thrown in the try block is caught.
- When nested try blocks are used, inner try block is executed first and any exception thrown in that block is caught in the following catch blocks.

Example

```
public class Test {
  public static void main(String[] args) {
    try {
      int i = Integer.parseInt(args[0]);
      int j = Integer.parseInt(args[1]);
      System.out.println(i/j);
    }
    catch (ArithmeticException a) {
      System.out.println("You're dividing by zero!");
    }
  }
}
```

Throw From Called Method

- The try statement gets a chance to catch exceptions thrown while the try part runs
- That includes exceptions thrown by methods called from the try part

Example

```
void f() {
  try {
    g();
  }
  catch (ArithmeticException a) {
    …
  }
}
```

- If g throws an ArithmeticException, that it does not catch, f will get it
- In general, the throw and the catch can be separated by any number of method invocations

Multiple catch Parts

- To catch more than one kind of exception, a catch part can specify some general superclass like RuntimeException

- But usually, to handle different kinds of exceptions differently, you use multiple catch parts

Available Methods

- public void printStackTrace()

  java.lang.ArithmeticException: / by zero
  at DemoClass.calc(DemoClass.java:15)
  at DemoClass.almostThere(DemoClass.java:9)
  at DemoClass.main(DemoClass.java:5)
- public String getMessage()

  e.g. / by zero
- public String toString()

  e.g. java.lang.ArithmeticException: / by zero

Example

```
public static void main(String[] args) {
  try {
    int i = Integer.parseInt(args[0]);
    int j = Integer.parseInt(args[1]);
    System.out.println(i/j);
  }
  catch (ArithmeticException a) {
    System.out.println("You're dividing by zero!");
  }
  catch (ArrayIndexOutOfBoundsException a) {
    System.out.println("Requires two parameters.");
  }
}
```

Example

```
public static void main(String[] args) {
try {
  int i = Integer.parseInt(args[0]);
  int j = Integer.parseInt(args[1]);
  System.out.println(i/j);
}
catch (ArithmeticException a) {
  System.out.println("You're dividing by zero!");
}
catch (ArrayIndexOutOfBoundsException a) {
  System.out.println("Requires two parameters.");
}
catch (RuntimeException a) {
  System.out.println("Runtime exception.");
}
}
```

Overlapping Catch Parts

- If an exception from the try part matches more than one of the catch parts, only the first matching catch part is executed
- A common pattern: catch parts for specific cases first, and a more general one at the end
- Note that Java does not allow unreachable catch parts, or unreachable code in general

The throw Statement

<throw-statement> ::= throw <expression> ;

- Most exceptions are thrown automatically by the language system
- Sometimes you want to throw your own
- The <expression> is a reference to a throwable object—usually, a new one:

```
throw new NullPointerException();
```

The Throws Clause

```
void z() throws SomeException
{
throw new SomeException ("You have run out of gas.", 19);
}
```

- A throws clause lists one or more throwable classes separated by commas
- This one always throws, but in general, the throws clause means might throw
- So any caller of z must catch SomeException, or place it in its own throws clause (propagate exception)

Finally Block

- Finally The finally block will execute whether or not an exception is thrown.

```
class FinallyDemo {
static void procA() {
try {
System.out.println("inside procA");
throw new RuntimeException("demo");
} finally {
System.out.println("procA's finally");
}
}
```

Propagating Exception

```
import java.io.*;
class DemoThrow
{       public static void main(String args[]){
    DemoThrow my = new DemoThrow();
}
  public DemoThrow() {     checkValue();     }
  public void  checkValue (){
    try      {
      makeCalc(0);
        }
    catch(ArithmeticException a)
    {              System.out.println(a);                  }  catch(IOException io)
                                                                        {
                                                                  System.out.println(io);
                                                                        }
  }
```

Custom Exception

```
class MyException extends Exception
{       MyException(int a)
        {  d=a;   }
        public String toString(){
                return "MyException "+d;        }
}
```

Custom Exception

```
class Throwdemo
{  int size,array[];
    Throwdemo(int s)
      { size=s;
        try  {
            checkValue();
            }
        catch(MyException e)
         {            System.out.println(e);
          }
}
```

Custom Exception

```
void checkValue() throws MyException
{
if( v<0)
            throw new MyException();
                    array= new int[3];
                    for(int i=0;i<3;i++)
                    array[i]=i+1;}

public static void main(String args[])
{
  new Throwdemo(Integer.parseInt(args[0]));
}
}
```

Assignments

Accept employee details like name,age,email.

If employee age is less than 25, then thow an exception AgeException.