Basic Java

Unit 6 – Exception Handling

Pratian Technologies (India) Pvt. Ltd.

www.pratian.com

PRATIAN
TECHNOLOGIES

Topics

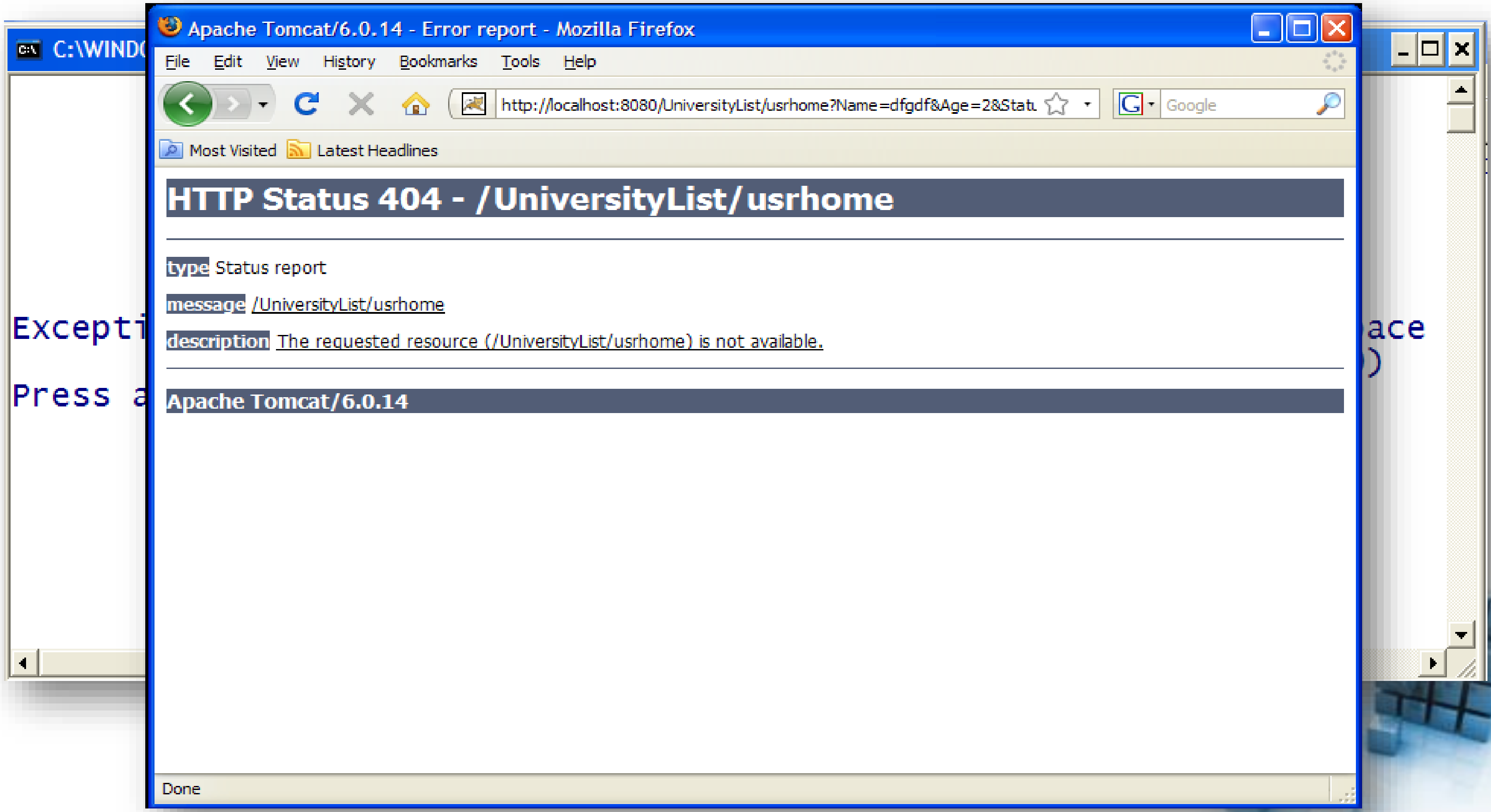
Error Condition

- Our applications should be built to survive ‘error conditions’.
- The program / module must handle the error situation gracefully and not just terminate abruptly

What ‘error ’ are we talking about ?

Error Condition

- An *error* is a condition due to which a program cannot normally continue execution and needs to be abruptly terminated.



C:\WIND
Excepti
Press a
ace
)
Apache Tomcat/6.0.14 - Error report - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/UniversityList/usrhome?Name=dfgdf&Age=2&Statu
Google
Most Visited
Latest Headlines
Done

HTTP Status 404 - /UniversityList/usrhome

type Status report

message /UniversityList/usrhome

description The requested resource (/UniversityList/usrhome) is not available.

Apache Tomcat/6.0.14

Error handling in traditional languages

- Traditional error handling methods include
  - Boolean functions (which return TRUE/FALSE).
  - Integer functions (returns –1 on error).
  - And other return arguments and special values.

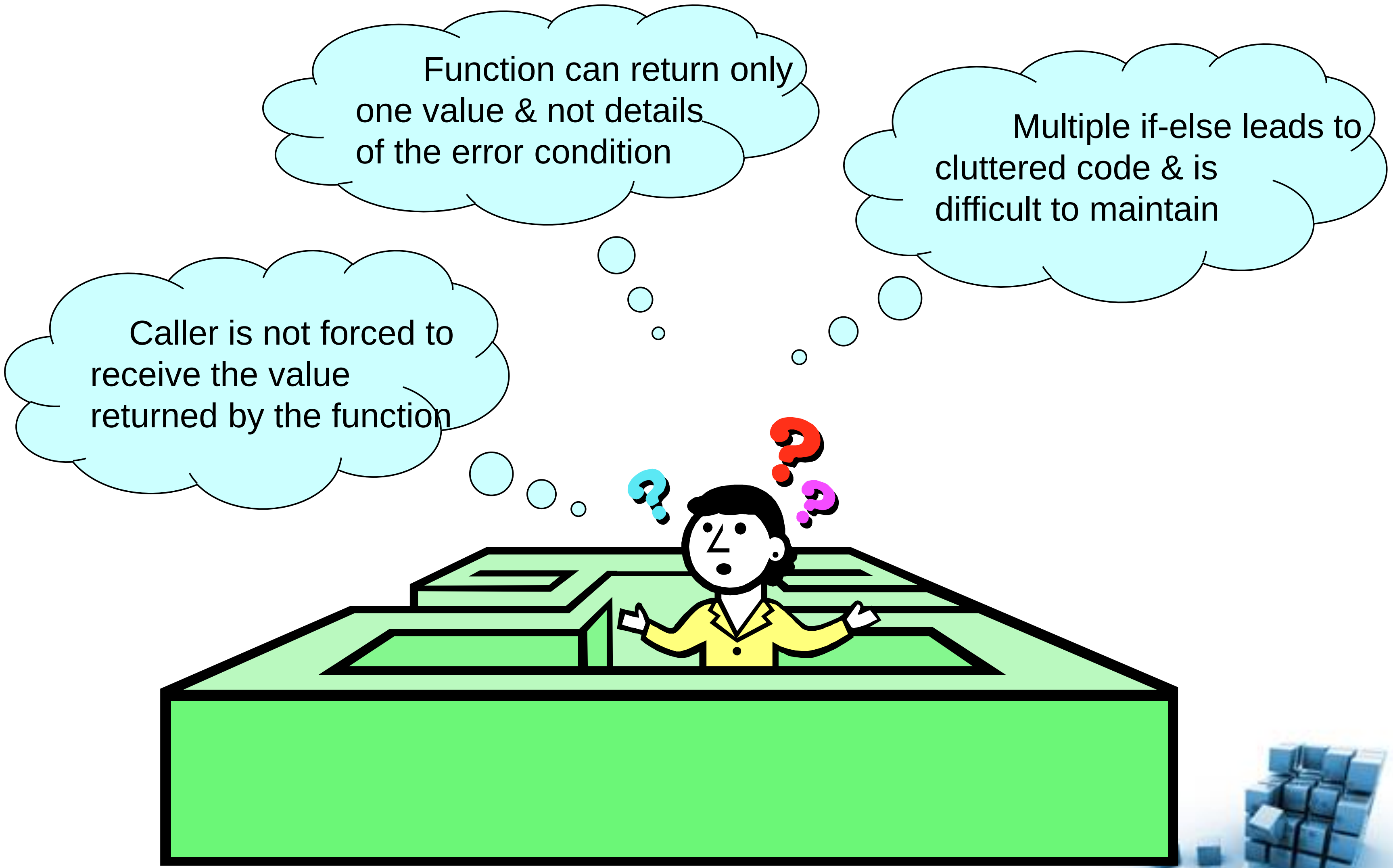
```
int main () {
        int res;
        if (can_fail () == -1) {
                cout << "Something failed!" << endl;
                return 1;
        }
        if(div(10,0,res) == -1) {
                cout << "Division by Zero!" << endl;
               return 2;
        }
              return 0;
 }
```

Problems with traditional approach

Function can return only one value & not details of the error condition

Multiple if-else leads to cluttered code & is difficult to maintain

Caller is not forced to receive the value returned by the function

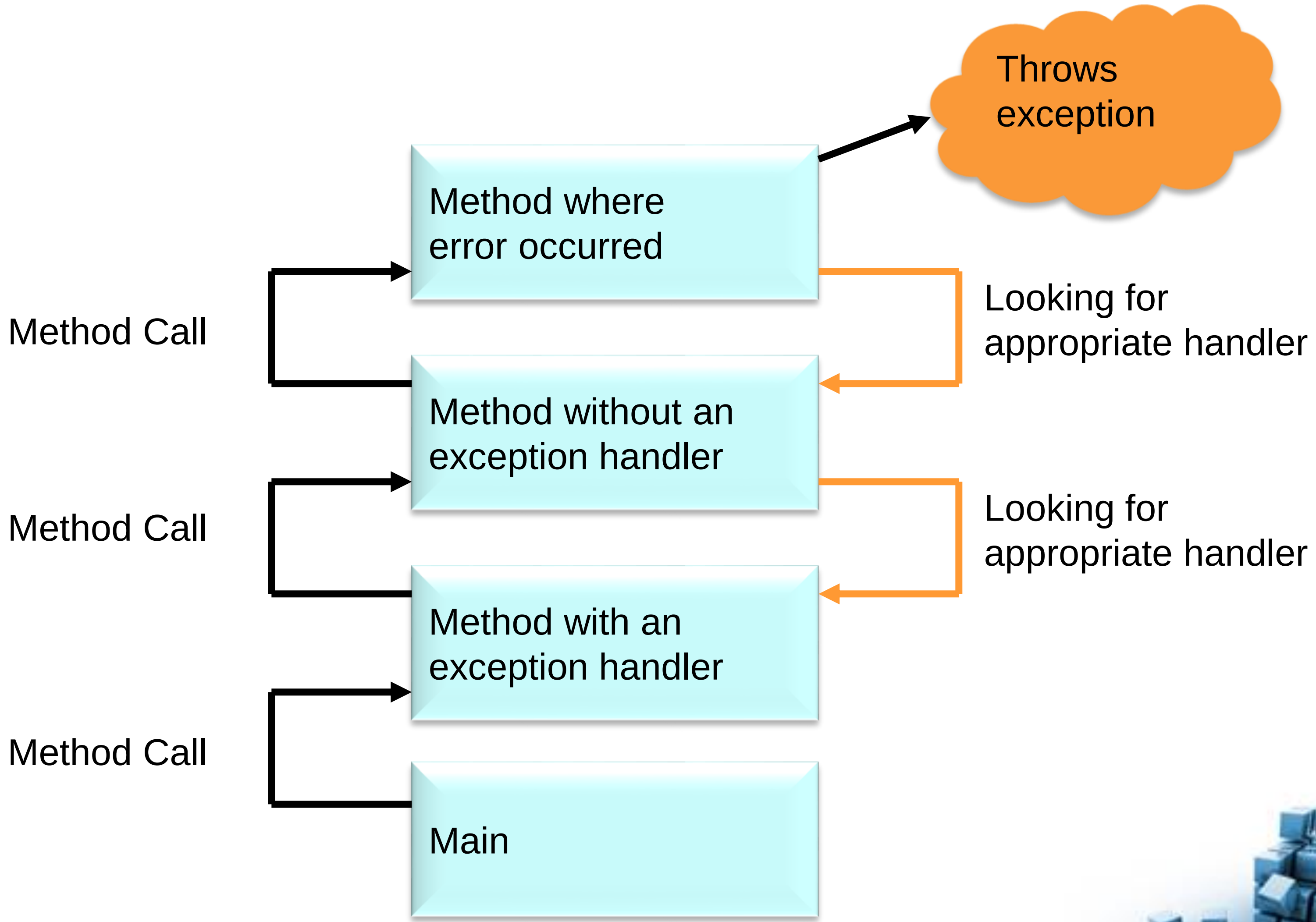


Exception handling in Java

- Java's support for error handling is done through Exceptions.

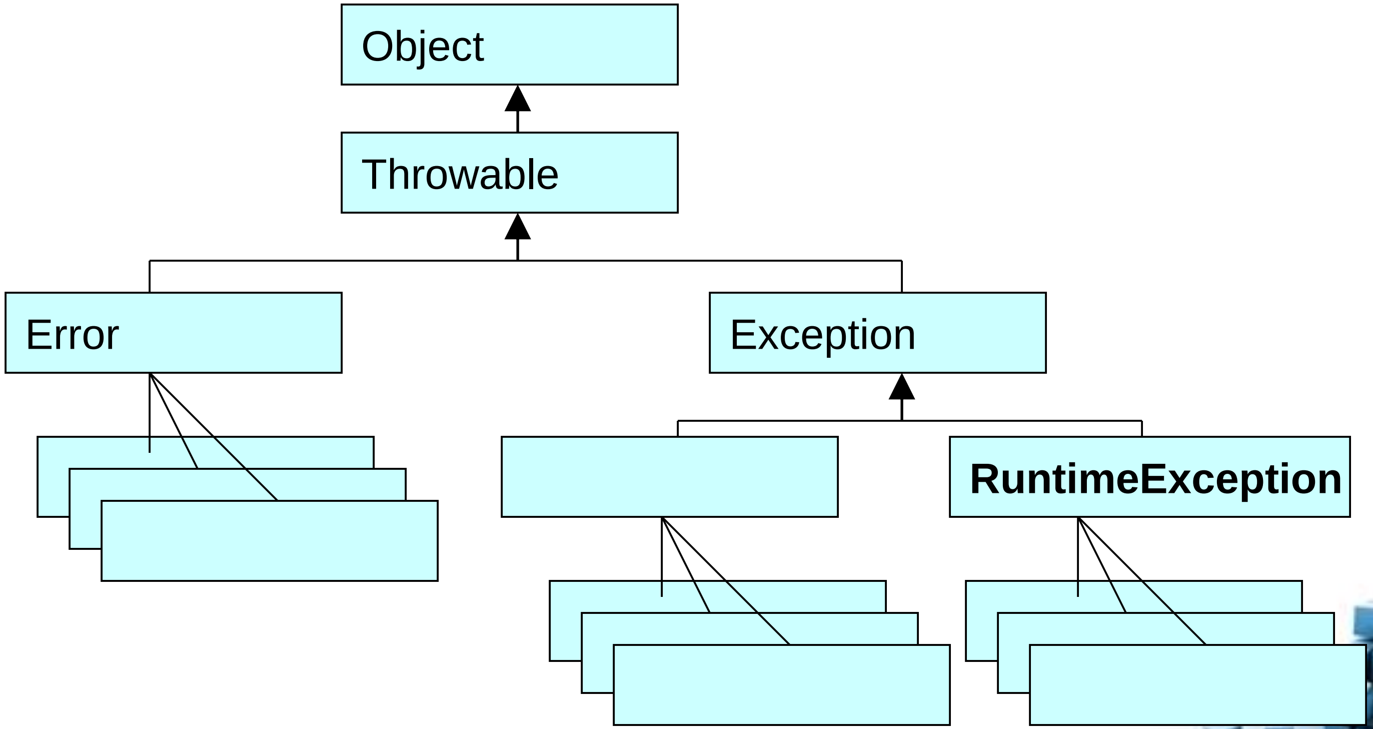
- **What is an Exception?**
  - An *exception* is an event that occurs during the execution of a program that disrupts the normal flow of instructions.
  - In the wake of such an event, the JVM creates an exception object, that contains information about the error, including its type and state of the program when the error occurred.
  - Creating an exception object and notifying the caller of the method is called *throwing an exception*.

Exception handling mechanism


Throws exception
Method where error occurred
Looking for appropriate handler
Method Call
Method without an exception handler
Looking for appropriate handler
Method Call
Method with an exception handler
Method Call
Main

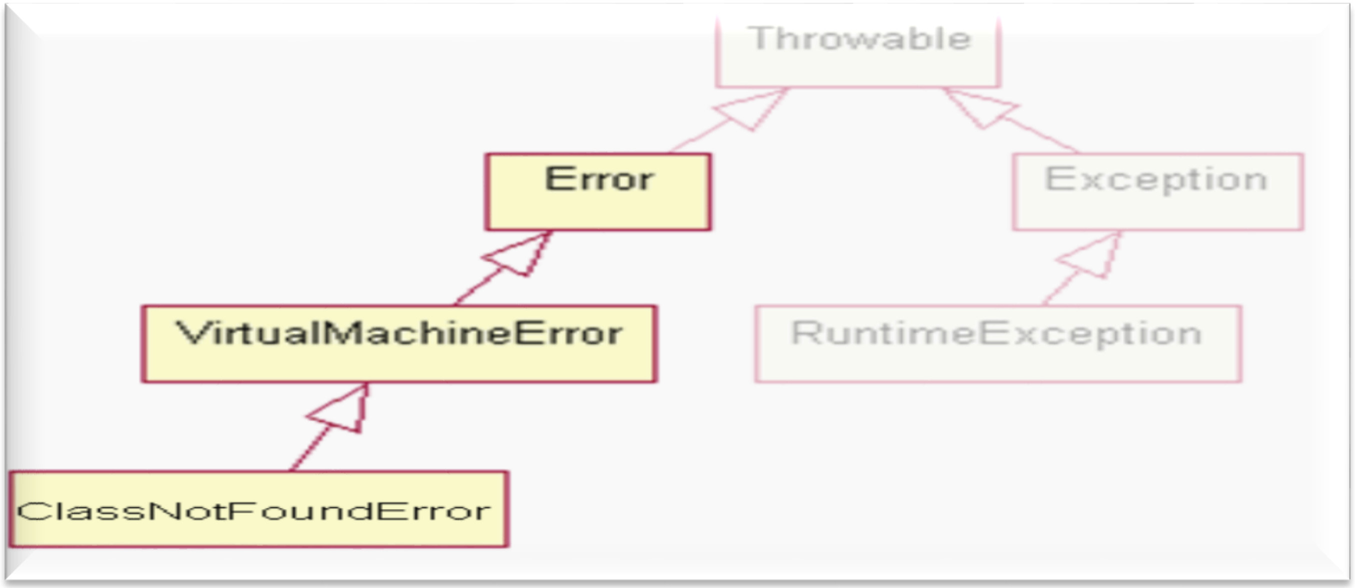
Throwable class

- When an error occurs in a method the caller of the method is notified by throwing an exception object, this object should be of the type class ***Throwable***.
- *Throwable* has two subclasses, *Error* and *Exception*.



PRATIAN TECHNOLOGIES

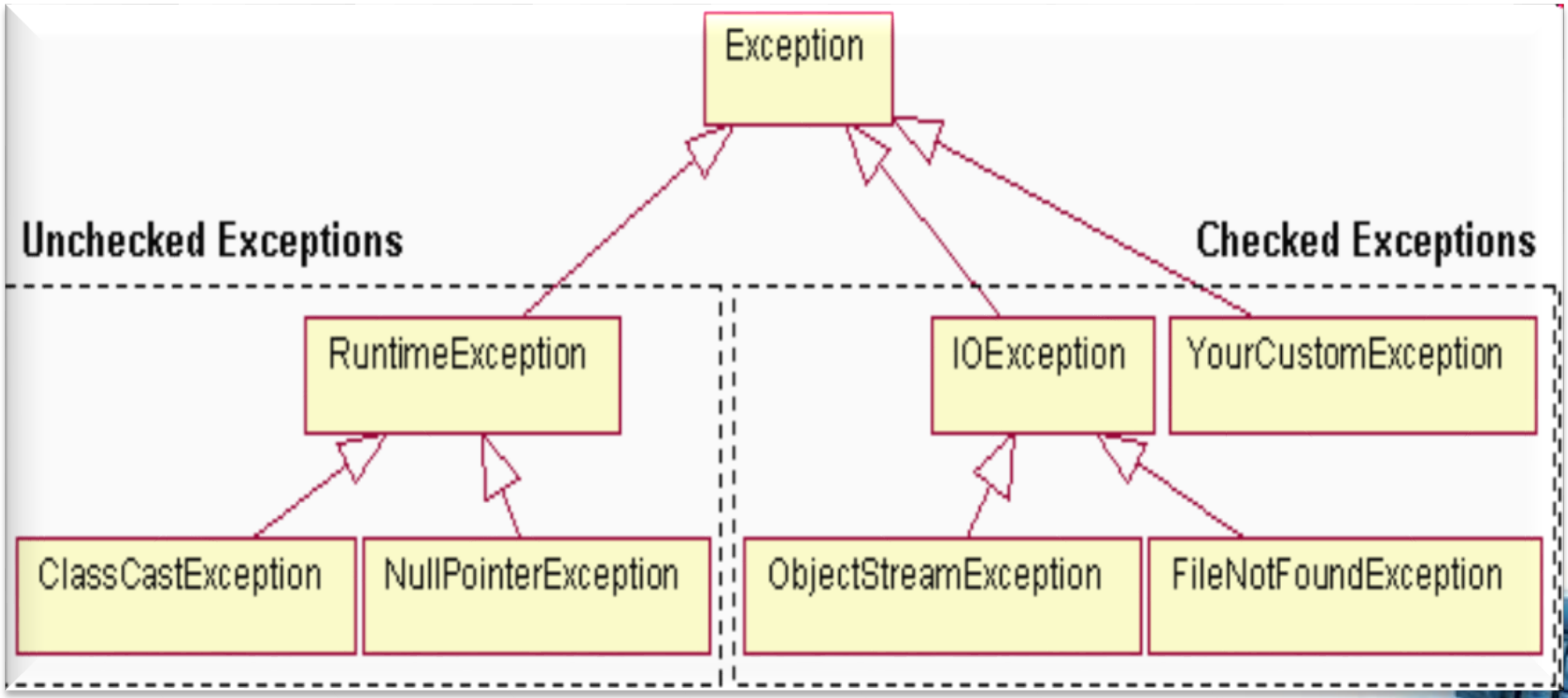
Errors



- Errors are exceptional conditions that are external to the application, and that the application cannot anticipate or recover from
- Errors are not required to be handled by the application.
- For example,
  - OutOfMemoryError
  - NoClassDefFoundError

Types of Exception

- Exceptions are of 2 types
  - **Checked exceptions**
  - **Unchecked or runtime exception**



PRATIAN TECHNOLOGIES

Types of Exception

- **Checked exceptions**
  - These are exceptional conditions that a well written application should anticipate and recover from.
  - Since these exceptions are anticipated and are recoverable conditions, compiler forces these exceptions to be handled.

  e.g.: FileNotFoundException, IOException

  SQLException
- **Unchecked or runtime exception**
  - These are exceptional conditions that usually indicate programming bugs, such as logical errors or improper use of an API.
  - Since these can be fixed programmatically, compiler does not force these exceptions to be handled.

  e.g.: NullPointerException

  ArrayIndexOutOfBoundsException

Exception Handlers

- Java programming language supports, exception handling by providing three exception handler components
  - ***try***
  - ***catch***
  - ***finally***

- The *try*, *catch* and *finally* blocks are used to write an exception handler.

try Block

- Any code that might throw an exception is enclosed within a ***try*** block. It is the first step in constructing an exception handler.

```
try
{
        code          }
                      }   try block
        ...........   }
}
```

try Block Example

```
public int countChars(String fileName)
{
        int total = 0;
        try {
                FileReader r = new FileReader(fileName);
                while( r.ready()) {
                        r.read();
                        total++;
                }
                r.close();
            }


}
```

catch Block

- The ***catch*** block contains code that is executed if and when the exception occurs.
- A ***catch*** block is an exception handler associated with a ***try*** block and handles the type of exception indicated by its argument.
- Every try block is associated with ***zero or more catch block***

```
try
{
}
catch(ExceptionType name)
{
}
catch(ExceptionType name)
{
}
```

catch Block Example

```
public int countChars(String fileName)
{
        int total = 0;
        try {
                FileReader r = new FileReader(fileName);
                while( r.ready()) {
                        r.read();
                        total++;
                 }
                         r.close();
        }
        catch(FileNotFoundException e){
              System.out.println("File named " + fileName + "not found. " +e);
                total = -1;
        }
        catch(IOException e){
           System.out.println(“Unable to perform I/O, please try later”);
               total = -1;
        }
}
```

Exercise

- Write some code that is capable of generating exceptions like ArithmeticException, NumberFormatException, ArrayIndexOutOfBoundsException etc

  Write an exception handler, that is, enclose these line of code in try block and write appropriate catch blocks to catch different exception types.

PRATIAN TECHNOLOGIES

Exception Handler

- It is possible to write a generic catch block to catch any type of exception that might occur

```
public int countChars(String fileName) {
        int total = 0;
        try {
                FileReader r = new FileReader(fileName);
                while( int c = r.read() != -1)
                        total++;
                r.close();
         }
         catch(FileNotFoundException e){
               System.out.println("File nam                    found. " +e);
          }
         catch(IOException e){
            System.out.println(“Unable t                        y later”);
         }
         catch(Exception e){
                System.out.println(“Unable to continue processing, due to some
                                        internal error”);
         }
}
```

Such a generic catch block should be included at the end, after the specific catch blocks

Clean up code

```
public int countChars(String fileName)
{
        int total = 0;
        try {
                FileReader r = new FileReader(fileName);
                while( r.ready()) {
                        r.read();
                        total++;
                 }
                 r.close();
        }
        catch(FileNotFoundException e){
              System.out.println("File named " + fileName + "not found. " +e);
                total = -1;
                 r.close();
        }
        catch(IOException e){
           System.out.println("Unable to perform I/O, please try later");
               total = -1;
                 r.close();
        }
}
```

Developer needs to perform clean up that is, release the file that was opened programmatically

finally Block

- A ***finally*** block is executed irrespective of whether the try block throws an error or not.
- ***finally*** block is guaranteed to be executed and can be used for any clean-up code.
- A try block can be followed up with **zero or more catch** blocks, but **only one *finally*** block.

```
try{
}
catch(ExceptionType name){
}
finally{
}
```

finally Block Example

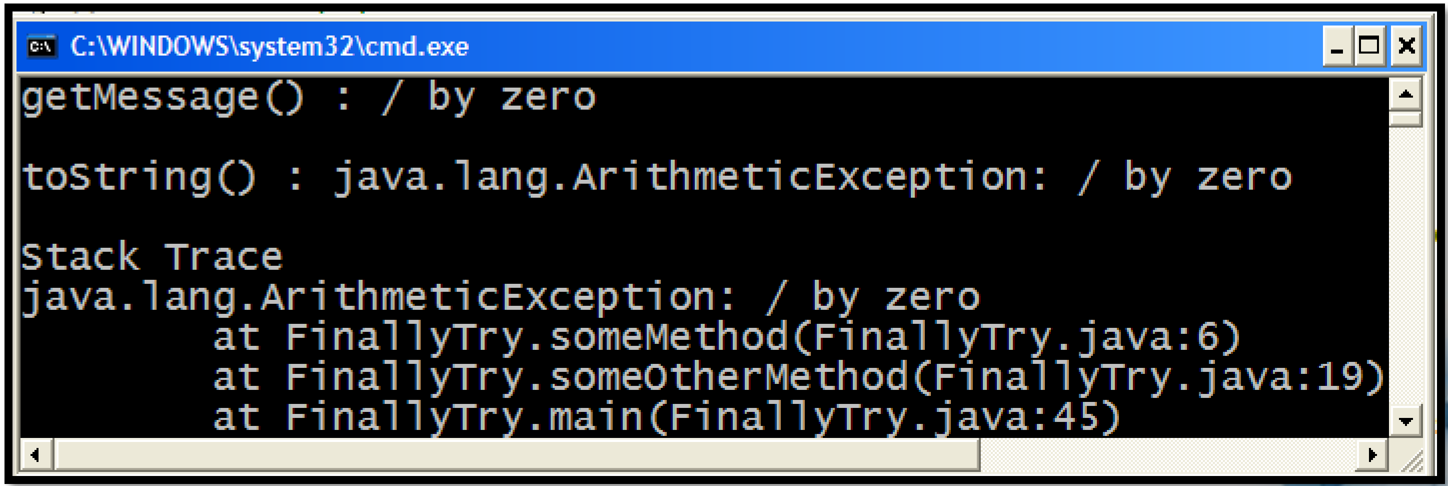
```
public int countChars(String fileName) {
        int total = 0;
        FileReader r = new FileReader(fileName);
        try {
                while( r.ready()) {
                        r.read();
                         total++;
                }
        }
        catch(FileNotFoundException e){
                System.out.println("File named " + fileName + "not found. " + e);
                total = -1;
        }
        catch(IOException e){
              System.out.println("IOException occured "+ "while counting " + e);
               total = -1;
        }
        finally {
                r.close();
         }
}
```

Clean up code goes into finally block

Some Useful Methods

- The Throwable class has some useful methods that can be called on any Exception type
  - String getMessage()
    - Returns the message that was set while the exception object was created, by calling the parameterized constructor

      Exception(String message)
  - String toString
    - Returns a String representation of the exception object
  - void printStackTrace()


C:\WINDOWS\system32\cmd.exe

```
getMessage() : / by zero

toString() : java.lang.ArithmeticException: / by zero

Stack Trace
java.lang.ArithmeticException: / by zero
        at FinallyTry.someMethod(FinallyTry.java:6)
        at FinallyTry.someOtherMethod(FinallyTry.java:19)
        at FinallyTry.main(FinallyTry.java:45)
```

Method specifying it ‘throws’ Exception

```
public void writeList() {
        BufferedWriter out = new BufferedWriter(new
                                                FileWriter(“outputFile.txt”));
        for(int i=0 ; i<SIZE ; ++i)
                out.write(vector.elementAt(i));
        out.close();
}
```

- The writeList() method includes method calls that throw FileNotFoundException, IOException
- The method however need not catch the exception and thereby allow a method further up the call stack to handle it.
- In that scenario, the method has to specify the exception as being thrown by the method.

The ‘throws’ clause

- A method can specify to throw an exception by adding a ***throws*** clause to the method declaration.
- The throws clause comprises the ***throws*** keyword followed by a comma-separated list of all the exceptions thrown by that method.
- The clause goes after the method name and argument list and before the brace that defines the scope of the method.

```
public void writeList() throws FileNotFoundException,
                                        IOException  {

          ……………………

}
```

User defined Exceptions

- If an exception cannot be represented by those in the Java platform, a user can define his own exception.
- The user defined exception class should be a subclass of Exception or any of its sub types.

```
public class
ProductNotFoundException
extends Exception
{
}
```

```
public Product getProduct(int
prodId) throws
ProductNotFoundException {
        .........
        .........
}
```

‘Throwing’ Exceptions

- Use the ‘throw’ clause to throw the user defined exception object
- The ***throw*** statement requires a throwable object as argument. Throwable objects are instances of any subclass of the ***Throwable*** class.
- ***throw*** causes the method to terminate and returns an exception object to the caller.

```
public Product getProduct(int prodId) throws
ProductNotFoundException {
        // Retrieve product info from the database
        // select * from Product where prod_id = prodId
        if(!found)
                throw new ProductNotFoundException();
        return product;
}
```

Question time

Please try to limit the questions to the topics discussed during the session. Thank you.