

# Exceptions

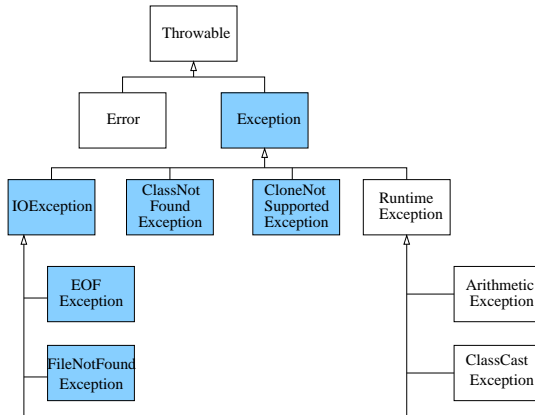
Tim Kovacs  
kovacs@cs.bris.ac.uk

Frederik Vercauteren  
frederik@cs.bris.ac.uk

- **User input** errors: typos in arguments, malformed url's, ...
- **Device** errors: printer turned off, network down, ...
- **Physical limitations**: disk is full, run out of memory, ...
- **Code** errors: invalid array index, division by zero, ...

- **Exception**: object that encapsulates **error information**
  - If error occurs, exception is **thrown**
  - Method that caused error **exits immediately**
  - Exception handling mechanism looks for **exception handler**
  - Handler **catches** exception and deals with it

## Exceptions - Classification



## Throwing Exceptions

- **Unchecked**: derived from **Error** or **RuntimeException**
- **Checked**: all the other exceptions
- Method that can throw exception has to **include this in header**

```
public Image loadImage(String s)
    throws EOFException, MalformedURLException
{
    ...
    throw new EOFException();
}
```
- **Checked** exception must **always** be **declared**
  - If you fail to do this, compiler will complain
  - Not necessary for unchecked exceptions

## Catching Exceptions

- When an exception is thrown, some code has to catch exception

```
try
{
    statement1
    statement2
    statement3
}
catch (ExceptionType1 e)
{
    handler for exception type 1
}
catch (ExceptionType2 e)
{
    handler for exception type 2
}
finally
{
    clean up claimed resources, will always be executed
}
```

## Exceptions - Classification

- **Error**: internal errors and resource exhaustion in JRE
  - Should not throw objects of this type, abnormal conditions
- **Exception**: program should deal with these exceptions
- **RuntimeException**: caused by a programming error
  - "If it's a **RuntimeException**, it's your own fault"
  - Bad cast, out-of-bound array access, null pointer access, ...
- Exceptions that **do not inherit** from **RuntimeException**
  - **IOException**: read past end of file, malformed URL, ...
  - **CloneNotSupportedException**: `clone` not implemented

## Creating Exceptions Classes

- If standard exception classes aren't adequate, **define your own**
  - Could use **static inner class**

```
class FileFormatException extends IOException
{
    public FileFormatException() { }
    public FileFormatException(String gripe)
    {
        super(gripe);
    }
}
```

- Can now declare and throw object of this exception type

```
String readData(BufferedReader in) throws FileFormatException
{
    ...
    if (n < len) throw new FileFormatException();
    ...
}
```

## Catching Exceptions

- If any of the code inside the **try** block throws exception
  - Program **skips remainder** of code in **try block**
  - Program **executes handler** inside first matching **catch** clause
- If none of the exception types matches the thrown exception
  - Method exits immediately
  - Exception **propagates up calling chain**
- If no exception is thrown, then program skips **catch** clauses
- Code in **finally** clause is **always** executed
  - Used to free up resources: close files, dispose of graphics

```
{
  FileReader inFile = null;
  FileWriter outFile = null;
  try
  {
    inFile = new FileReader(source);
    outFile = new FileWriter(destination, true);
    ...
  }
  catch(IOException e)
  {
    System.out.println(e.toString());
  }
  finally
  {
    if (inFile != null) { inFile.close(); }
    if (outFile != null) { outFile.close(); }
  }
}
```

- Don't replace a simple test by a try / catch block
- Example: check if there are more elements in list

- **Do not micromanage** exceptions
  - Don't use a separate try / catch block for each statement
- **Do not squelch** (suppress) exceptions
  - Compiler will complain if you don't declare checked exceptions
- Do not be afraid of **propagating exceptions**
  - Only handle the exception if you know how to deal with it