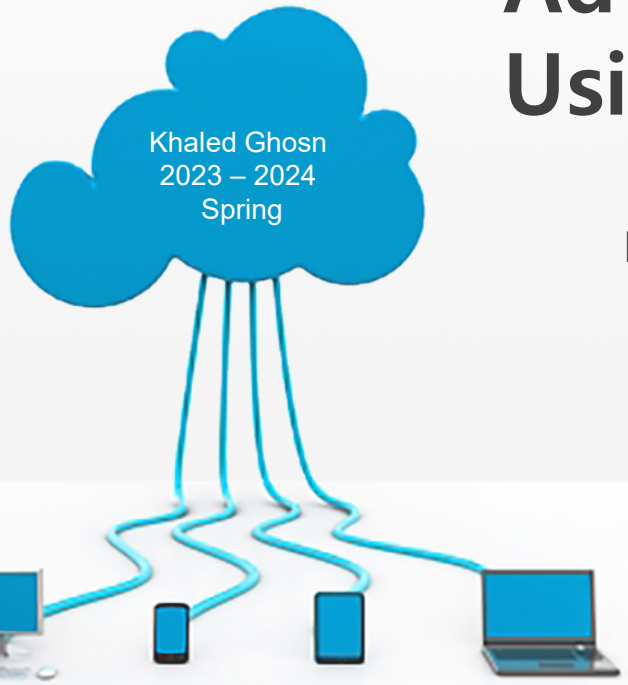
ARTS, SCIENCES & TECHNOLOGY UNIVERSITY IN LEBANON

AUL

CSC 320

Advanced Programming Using Java

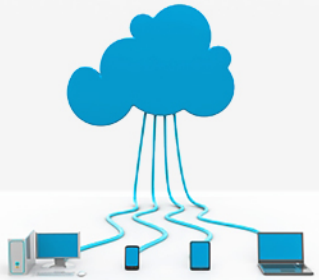


Khaled Ghosn
2023 – 2024
Spring

EXCEPTION HANDLING

Exception Handling

Exception handling enables a program to deal with exceptional situations and continue its normal execution.



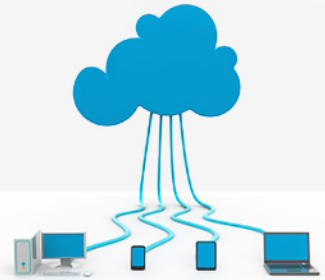
Exception Handling

Errors & Exception Types

a) **Compile Time Errors**
- E.g. syntax errors, static type mismatch, unresolved names, missing imports…
- Detected by the developer
- Program won't run before they're fixed

b) **Logical Errors**
- Happen at runtime without halting the program
- Undetectable by the compiler (otherwise program wouldn't run in the first place)
- Undetectable by the developer (otherwise he/she would have avoided it)
- E.g.
```
double tip = 10;
double subtotal = 25;
double total = subtotal * tip / 100;
```

c) **Runtime Errors**

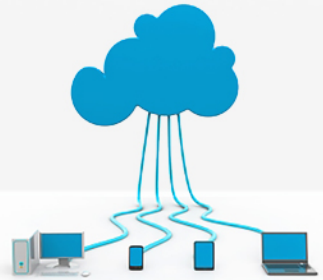


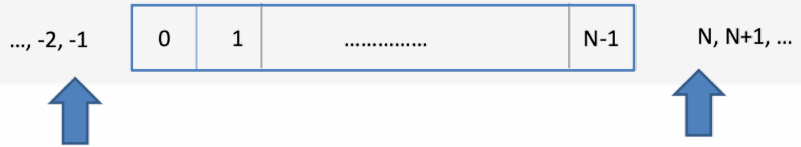
Exception Handling

Runtime Exceptions

c) Runtime Errors

- Happen at runtime (obviously!) causing the program to halt
- Undetectable by the compiler, but should be detected by the developer
- e.g.
  - division by 0 (ArithmeticException)
  - navigating inside a null object (NullPointerException)
  - Scanning type mismatch (InputMismatchException)
  - Traversing array outside its bounds (ArrayIndexOutOfBoundsException)
  - Passing illegal arguments to a method (IllegalArgumentException)



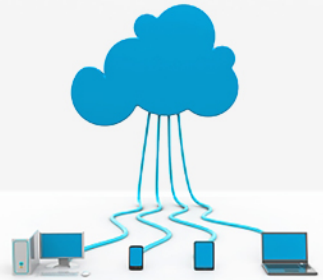


Exception Handling

The Exception Object

- In Java, runtime errors are thrown as exceptions.
- An exception is an object that represents an error or a condition that prevents execution from proceeding normally.
- We are responsible for handling exceptions so that the program can continue to run normally or terminate gracefully.

```
try {
  Code to run;
  A statement or a method that may throw an exception;
  More code to run;
}
catch (type ex) {
  Code to process the exception;
}
```

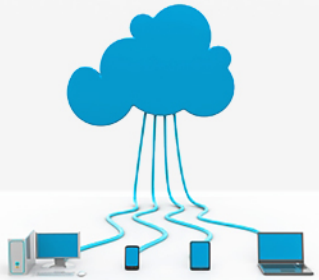


Exception Handling

Advantage of Using Exceptions Handling

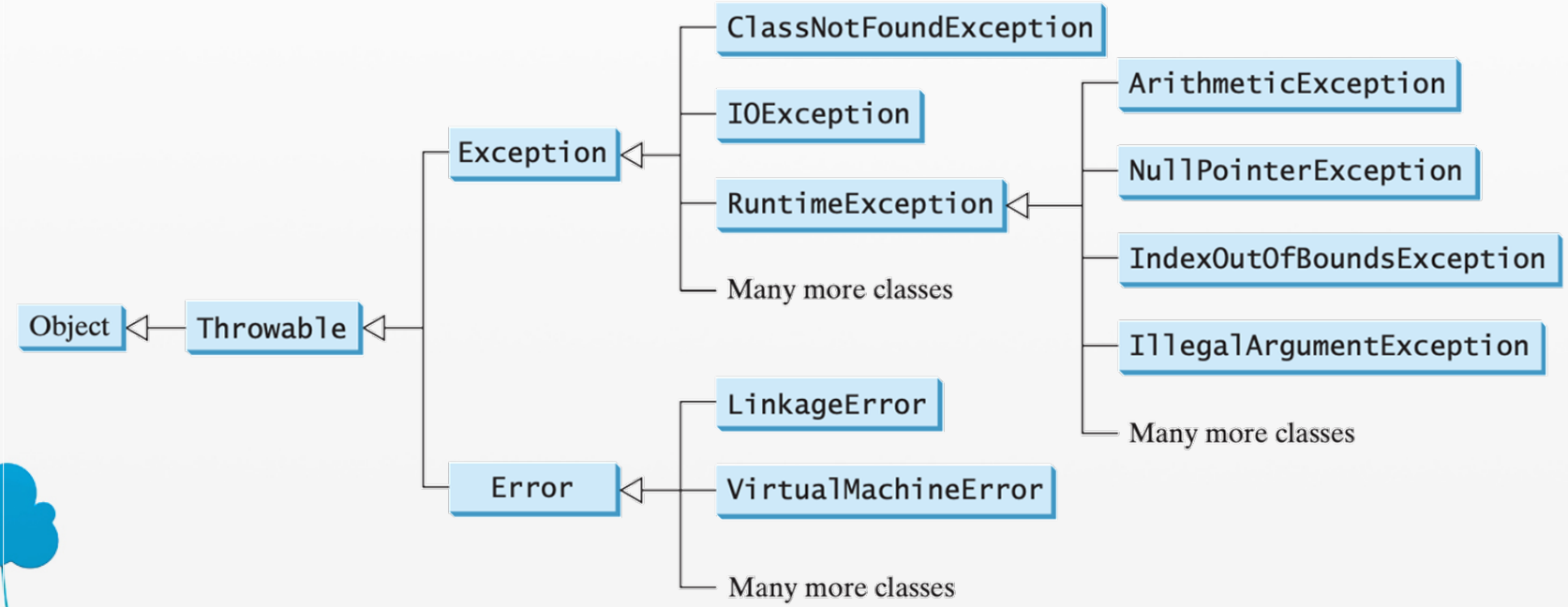
- Exception Handling enables a **method** to **throw** an exception to its caller, enabling the *caller* to *handle* the exception.

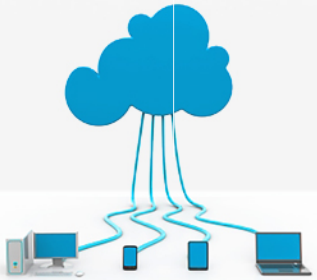
- So, the key benefit is separating the **detection of an error** (done in a *called* method) from the (done in the *calling* method). **handling of an error**.



Exception Handling

Exception Types

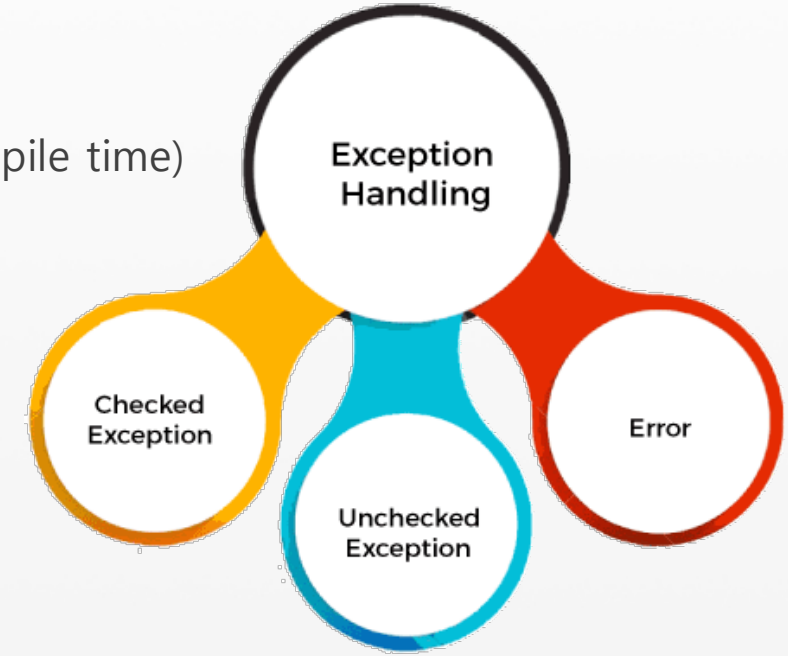


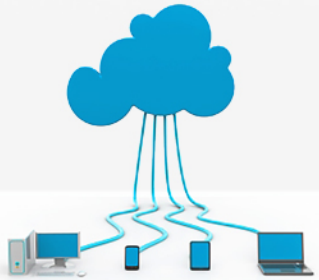


Exception Handling

Exception Types

- **Checked Exceptions** (at compile time)
- **Unchecked Exceptions**
- **Errors** (irrecoverable)





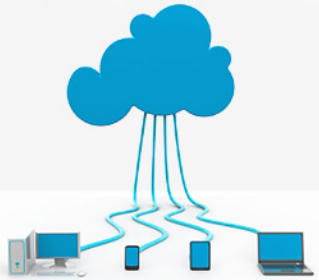
Exception Handling

Checked Exceptions vs. Unchecked Exceptions Types

- RuntimeException, Error and their subclasses are known as **unchecked** exceptions.

- All other exceptions are known as **checked** exceptions,

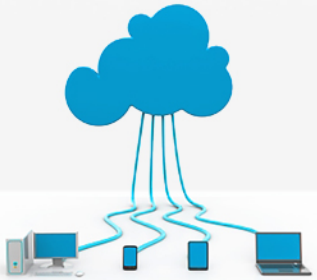
  Meaning that the compiler forces the programmer to check and deal with the exceptions either in a try-catch block or declare the exception in the method header.



Exception Handling

Unchecked Exceptions

- In most cases, unchecked exceptions reflect programming logic errors that are not recoverable, e.g.
  - a **NullPointerException** is thrown if you access an object through a reference variable before an object is assigned to it;
  - an **IndexOutOfBoundsException** is thrown if you access an element in an array outside the bounds of the array.
- These are the logic errors that should be corrected in the program.
- Unchecked exceptions can occur anywhere in the program.
- To avoid cumbersome overuse of try-catch blocks, Java does not mandate you to write code to catch unchecked exceptions.

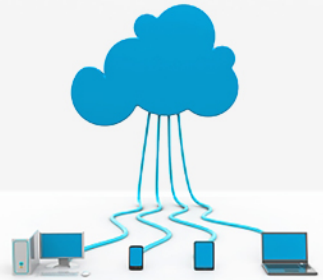


Exception Handling

Declaring, Throwing, and Catching Exceptions

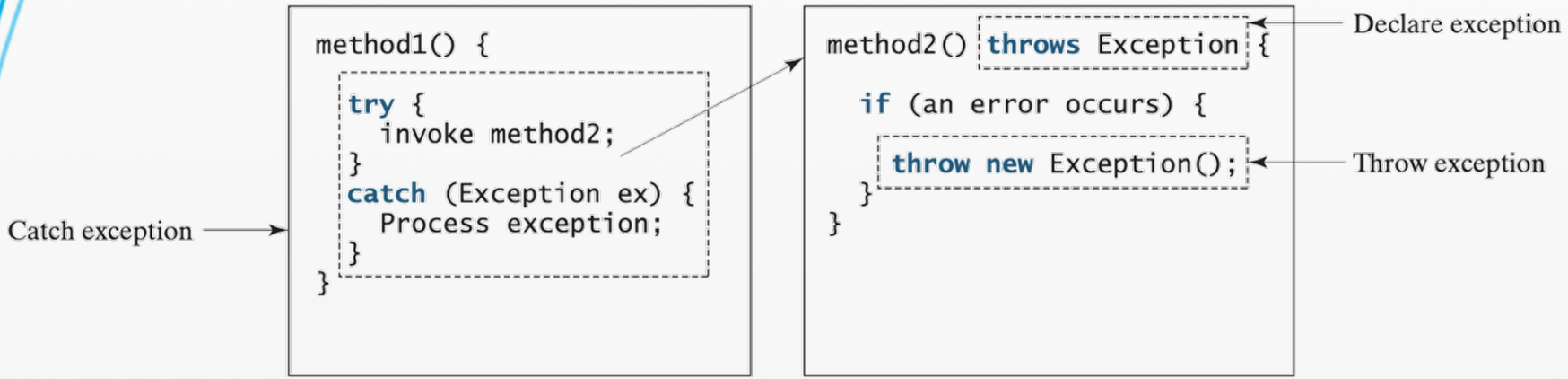
Java's exception-handling model is based on three operations:

1. **Declaring** an exception: throws
2. **Throwing** an exception: throw
3. **Catching** an exception

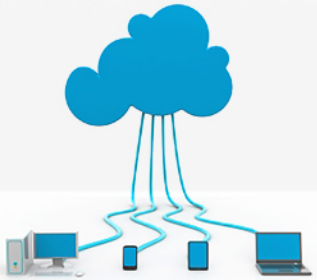


Exception Handling

Exception Handling Model



Exception handling in Java consists of declaring exceptions, throwing exceptions, and catching and processing exceptions.



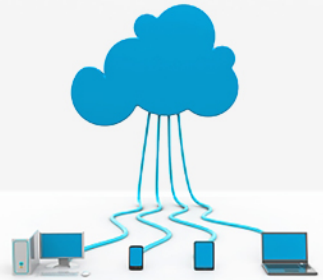
Exception Handling

Declaring an Exception

- Every method must state the types of checked exceptions it might throw.
- Use **throws** keyword in the method header to declare an exception:

  public void myMethod ( ) **throws** IOException
- Use separate commas to throw multiple exceptions

  public void myMethod ( ) **throws** Exception1, Exception2, ..., ExceptionN
- No need to try & catch inside the method when throwing.



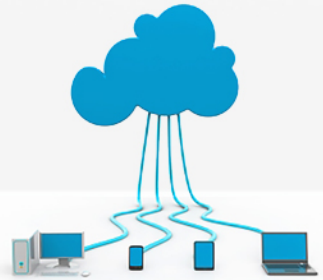
Exception Handling

Throwing an Exception

- A program that detects an error can create an instance of an appropriate exception type and throw it; e.g.
  - Exception ex = new Exception (“Something goes wrong");
    throw ex; // or you can write:
  - throw new Exception (" Something goes wrong");
- In general, each exception class in the Java API has at least two constructors:
  - a no-argument constructor, e.g.

    Exception ex = new Exception ( );
  - a constructor with a String argument that describes the exception. This argument is called the exception message, which can be obtained using **getMessage ( )** , e.g.

    ex.getMessage ( );



Exception Handling

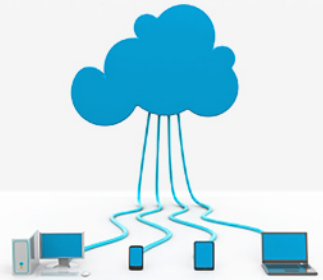
Throw and Throws in Java

Throw keyword:

- Is used to explicitly throw an exception from a method or any block of code.
- We can throw either checked or unchecked exception.
- The throw keyword is mainly used to throw custom exceptions.
- e.g. `throw new ArithmeticException ("/ by zero");`

Throws keyword:

- Is used in the signature of method to indicate that this method might throw one of the listed type exceptions.
- The caller to these methods has to handle the exception using a **try-catch block**.
- Syntax: type method_name (parameters) throws exception_list

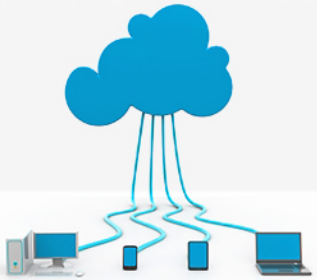


Exception Handling

Catching an Exception

When an exception is thrown, it must be caught.

```
try {
  statements; // Statements that may throw exceptions
}
catch (Exception1 exVar1) {
  handler for exception1;
}
catch (Exception2 exVar2) {
  handler for exception2;
}
...
catch (ExceptionN exVarN) {
  handler for exceptionN;
}
```



Handling several exceptions with same code:

```
catch (Exception1 | Exception2 | ... | Exceptionk ex) {
    // Same code for handling these exceptions
}
```

Exception Handling

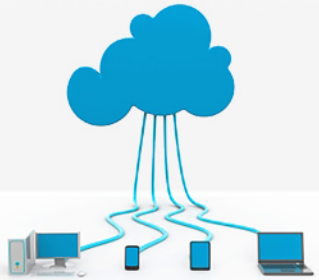
Catching an Exception

1. If no exceptions arise during the execution of the try block, the catch statements are skipped.

2. If one of the statements inside the try block throws an exception, the remaining statements are skipped. Then, the code will skip to the catch block which handles the exception that has occurred.

3. If no handler is found, the method is exited, the exception is passed to the calling method, and the same process is repeated.

4. If no handler is found in the chain of methods being invoked, the program terminates and prints an error message on the console.



Exception Handling

Order of catch blocks

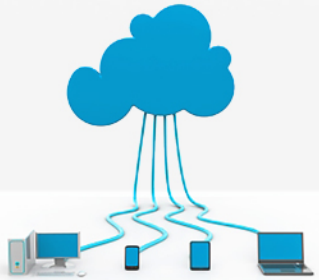
Catch the most specified first, then their parent, grandparents, and so on...

```
try {
  ...
}
catch (Exception ex) {
  ...
}
catch (RuntimeException ex) {
  ...
}
```

(a) Wrong order

```
try {
  ...
}
catch (RuntimeException ex) {
  ...
}
catch (Exception ex) {
  ...
}
```

(b) Correct order



Exception Handling

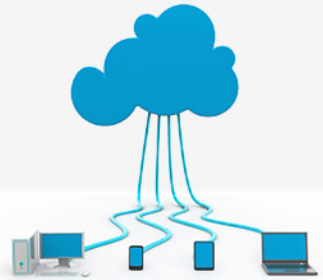
Order of catch blocks

- If the protected code can throw different exceptions which are not in the same inheritance tree, i.e. they don't have parent-child relationship, the catch blocks can be sorted any order.

- If the exceptions have parent-child relationship, the catch blocks must be sorted by the most specific exceptions first, then by the most general ones.

→

When you are handling multiple catch blocks, make sure that you are specifying exception sub classes first, then followed by exception super classes. Otherwise we will get compile time error.



Exception Handling

Catching an Exception

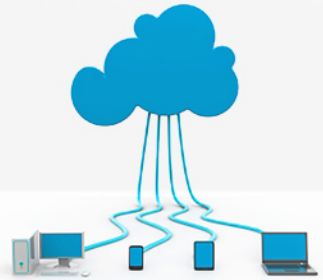
```
main method {
  ...
  try {
    ...
    invoke method1;
    statement1;
  }
  catch (Exception1 ex1) {
    Process ex1;
  }
  statement2;
}
```

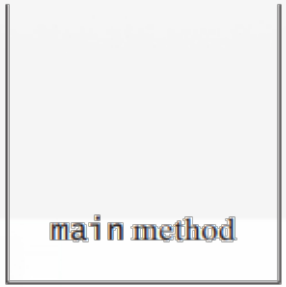
```
method1 {
  ...
  try {
    ...
    invoke method2;
    statement3;
  }
  catch (Exception2 ex2) {
    Process ex2;
  }
  statement4;
}
```

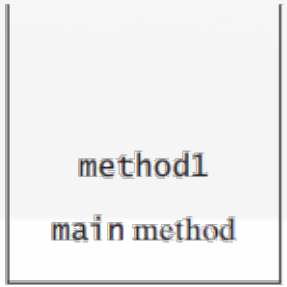
```
method2 {
  ...
  try {
    ...
    invoke method3;
    statement5;
  }
  catch (Exception3 ex3) {
    Process ex3;
  }
  statement6;
}
```

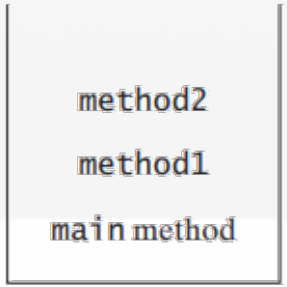
An exception is thrown in method3

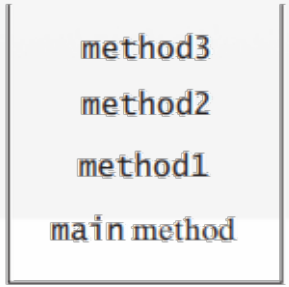
Call stack











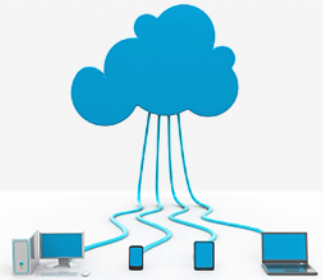
Exception Handling

The finally Clause

The finally clause is always executed regardless whether an exception occurred or not

(executed under all circumstances, regardless of whether an exception occurs in the try block or whether the exception is caught or not)

```
try {
    statements;
}
catch (TheException ex) {
    handling ex;
}
finally {
    finalStatements;
}
```



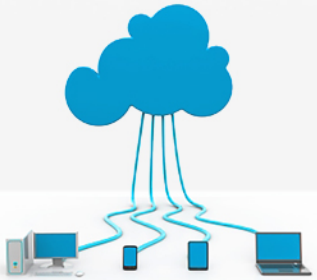
Exception Handling

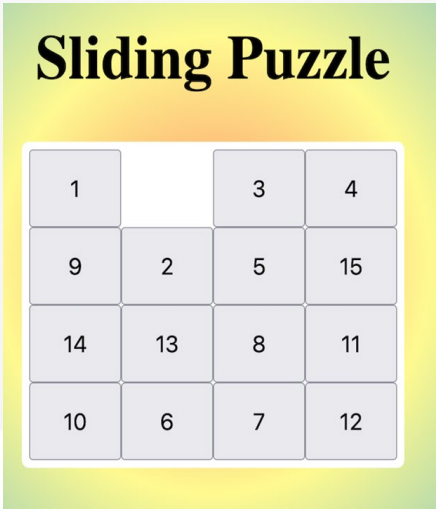
Creating Your Own Exception Class

You can define a custom exception class by extending the **java.lang.Exception** class.

E.g.

```
class IllegalMoveException extends
Exception {

    @Override

    getMessage()

    ...

}
```



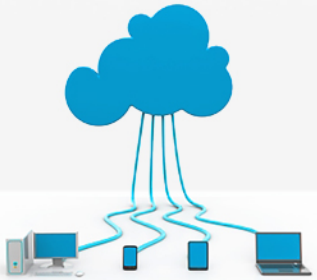


Exception Handling

Creating Your Own Exception Class

Then, in any method, declare it as:

```
public void move() throws IllegalMoveException {
…
    if(...) {
        throw new IllegalMoveException(...);
    }
}
```



Exception Handling

Getting Information from Exceptions

An exception object contains valuable information about the exception.

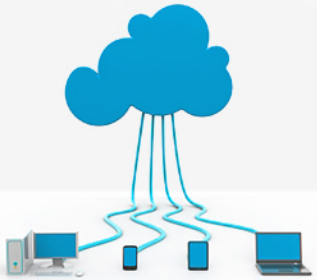
java.lang.Throwable	
`+getMessage(): String`	Returns the message that describes this exception object.
`+toString(): String`	Returns the concatenation of three strings: (1) the full name of the exception class; (2) `":"` (a colon and a space); (3) the `getMessage()` method.
`+printStackTrace(): void`	Prints the `Throwable` object and its call stack trace information on the console.
`+getStackTrace(): StackTraceElement[]`	Returns an array of stack trace elements representing the stack trace pertaining to this exception object.

Exception Handling

Rethrowing Exceptions

Java allows an exception handler to rethrow the exception if the handler cannot process the exception or simply wants to let its caller be notified of the exception.

```
try {
    statements;
    statements;
}
catch ( TheException ex ) {
   perform operations before exits;
   throw ex;
}
```



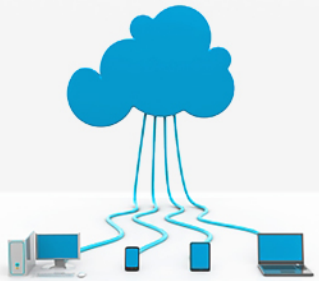
The statement **throw ex** rethrows the exception to the caller so that other handler in the caller get a chance to process the exception **ex**.

Exception Handling

Trace a Program Execution

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
catch(Exception2 ex) {
  handling ex;
  throw ex;
}
finally {
  finalStatements;
}

Next statement;
```

statement2 throws an exception of type Exception2.

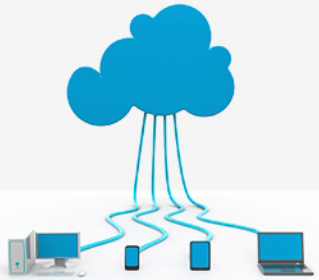


Exception Handling

Trace a Program Execution

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
catch(Exception2 ex) {
  handling ex;
  throw ex;
}
finally {
  finalStatements;
}

Next statement;
```

Handling exception

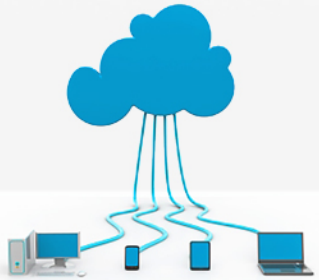


Exception Handling

Trace a Program Execution

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
catch(Exception2 ex) {
  handling ex;
  throw ex;
}
finally {
  finalStatements;
}

Next statement;
```

Execute the final block

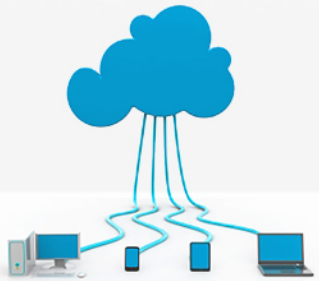


Exception Handling

Trace a Program Execution

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
catch(Exception2 ex) {
  handling ex;
  throw ex;
}
finally {
  finalStatements;
}

Next statement;
```

Rethrow the exception and control is transferred to the caller

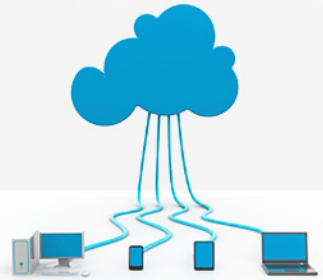


Exception Handling

Chained Exceptions

Sometimes, you may need to throw a new exception (with additional information) along with the original exception.

```
try {
  statements;
}
catch(TheException ex) {
  perform operations before exits;
  throw new Exception("Message", ex);;
}
```

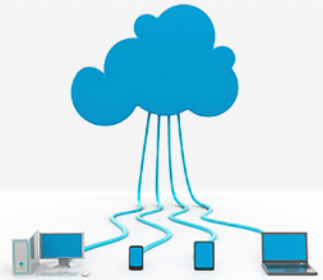


The exception ex is wrapped in a new exception, which is thrown and caught in the catch block in the caller method. The new exception will contain the original exception, in addition to its own parameters.

Exception Handling

Chained Exceptions

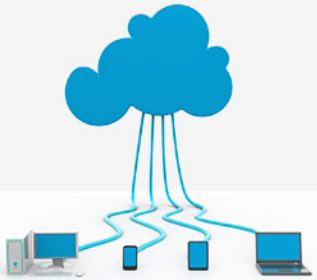
```
public class ChainedExceptionDemo {
  public static void main(String[] args) {
    try {
      method1();
    }
    catch (Exception ex) {
      ex.printStackTrace();
    }
  }

  public static void method1() throws Exception {
    try {
      method2();
    }
    catch (Exception ex) {
      throw new Exception("New info from method1", ex);
    }
  }

  public static void method2() throws Exception {
    throw new Exception("New info from method2");
  }
}
```



Exception Handling

Chained Exceptions

```
java.lang.Exception: New info from method1
  at ChainedExceptionDemo.method1(ChainedExceptionDemo.java:16)
  at ChainedExceptionDemo.main(ChainedExceptionDemo.java:4)
Caused by: java.lang.Exception: New info from method2
  at ChainedExceptionDemo.method2(ChainedExceptionDemo.java:21)
  at ChainedExceptionDemo.method1(ChainedExceptionDemo.java:13)
  ... 1 more
```



What would be the output if the body of the catch block in method1 is replaced by the following line?

```
throw new Exception("New info from method1");
```