Exceptions

Exceptions allow the program to handle gracefully exceptional situations during runtime and continue execution.

When a program encounters a runtime error, the JVM will throw an exception. If the exception is not handled, the program will terminate with a stack dump and the exception that caused the error:

```java
public class Tex {
    public static void main(String[] args) {
        System.out.println("1 / 0 = ", quotient(1,0));
    }

    public static int quotient(int num, int den) {
        return num / den;
    }
}

$ java Tex
Exception in thread "main"
java.lang.ArithmeticException: / by zero
   at Tex.quotient(Tex.java:7)
at Tex.main(Tex.java:3)
```
You could write the method to protect against a runtime error, but then the caller will not be notified that an error has occurred.

Exceptions are designed to work as error flags and allow the caller to decide how to handle the eventual error and how, or if, to continue.

The method where the error has occurred signals it using the `throw` statement:

```java
public static int quotient(int num, int den) {
    if ( den == 0 )
        throw new ArithmeticException("Division by 0");
    return num / den;
}
```

The error is handled in the calling method using the `try-catch` statement:

```java
try {
    System.out.println("1 / 0 = " + quotient(1, 0));
} catch (ArithmeticException e) {
    System.out.println("Error: cannot divide by 0 !");
}
```
The code in the `try` block will be executed normally as long as no exception is thrown.

If an exception is thrown the JVM will look for a `catch` block with an exception that matches and will execute that code.

**Exception types**

Exceptions are objects, as should be clear from the use of the `new` keyword when throwing an exception.

Java defines many exception. The root class for all of them is `java.lang.Throwable`. The root class is extended by `Error` and `Exception`, which are extended by a long list of subclasses (`ArithmeticException` extends `RuntimeException` which extends `Exception`).

`Error`, `RuntimeExceptions` and their subclasses are *unchecked exceptions*, all other are *checked exceptions*.
Declaring exceptions

Methods must declare which **checked** exceptions they might throw:

```java
void func(arguments) throws Excep1, Excep2, ...
```

Methods do not need to declare **unchecked** exceptions, such as runtime exceptions, that they might throw.

Handling exceptions

When an exception is thrown, the JVM will look for a `catch` statement with a matching exception type in the current method.

```java
catch (Exception ex) { }
catch (Exception1 | Exception2 ex) { } // JDK 1.7
```

If one is not found, the exception is **propagated** backward through the method calls chain and each method is searched for the matching `catch` statement. If none is find in any of the methods on the stack, the program terminates and an error statement is printed.
The finally clause

The optional finally clause is a block of statements part of a try-catch which are always executed whether an exception was thrown or not:

```java
try {
    func();
    // Do after func was successful
}
catch (Exception e) {
    // Do after the Exception was thrown
}
finally {
    // Statements always executed
}
```
Rethrowing Exceptions

If an matching `catch` handler is not found in the current method, the exception is propagated up the call stack. There are times when it is desirable to execute some actions before propagating the exception. This is referred to `rethrowing` the exception and is achieved using the `throw` keyword in the `catch` handler:

```java
catch (Exception ex) {
    // Local processing
    throw ex;              // rethrow ex to caller
}
```
Using exception objects

The Throwable class defines several public methods. The most useful are:

```java
String getMessage() - return message from the constructor
void printStackTrace() - print the throwable and its backtrace
String toString() - return class name followed by message
```

Example:

```java
try {
    // code that may throw exceptions
} catch (Exception ex) {
    System.out.println("Error: " + ex.getMessage());
}
```