1 What are exceptions?

Runtime errors in Java are indicated by exception objects that represent error conditions that prevent execution from continuing normally. Some of these errors can be handled so that the program continues, others cannot be recovered from.

The exceptions are thrown (signaled) by methods that encounter a problem so that the calling methods get an indication of the problem. The calling method can either handle the exception or pass it to its caller, and so on. The exceptions should be thrown ONLY if the method itself cannot handle the error in any good way, or if it cannot be decided at that level what the good way of handling the problem is.

Silly Example:

Without exceptions

    public int divide ( int x, int y) {
      return x/y;
    }
    public static void main (String [] args ) {
      System.out.println(divide(78, 3));
      System.out.println(divide(78, 0));
      System.out.println("you should not see this");
    }

With exceptions

    public int divide ( int x, int y) throws ArithmeticException {
      if ( y ==0 ) throw new ArithmeticException ;
      return x/y;
    }
    public static void main (String [] args ) {
      try {
        System.out.println(divide(78, 3));
        System.out.println(divide(78, 0));
      }
The method `divide()` does not know what to do if the divisor is zero. Its return type is `int`, so what should it return when one tries to divide by zero?

This is a situation in which the `divide()` method can communicate to the calling method (main in this case) that something went wrong. The `main()` method can either handle it by printing an error message, ignoring the attempted division, terminating the program, etc, OR it can ignore the exception (not catch it) which passes the exception down the stack to `main`’s calling method and then the JVM terminates the program with an error message.

**General syntax for catching exceptions:**

```java
try {
    // code that may throw exception
} catch (ExceptionType1 e) {
    // handle exception of type 1
} catch (ExceptionType2 e) {
    // handle exception of type 2
} ...
```

**General syntax for throwing exceptions:**

```java
throw new ExceptionType("message");
```

**General syntax for declaring exceptions:**

```java
methodName (parameterList) throws ExceptionType {
    ...
}
```
2 To catch or not to catch

Java requires you to catch and handle some exceptions, others do not need to be handled. This depends on the type of the exception. The following figure shows partial inheritance tree for different exception classes.

The **Error** class represents system errors - the program does not have any means of handling them, other than gracefully terminating itself.

The **Exception** class represents errors that can be handled within the code. The classes of exceptions shown in gray are so called **unchecked exceptions** - the programs are not required to catch them and/or handle them. The classes of exceptions shown in white are so called **checked exceptions** - the programs have to catch them or explicitly pass them to the calling methods.

If your method throws a checked exception, the method has to declare that exception.

3 Getting useful information from exceptions

The Throwable class provides methods that can be used when an exception is caught.

- **getMessage()** returns a String object containing the message that describes the exception object (that’s the message with which the exception was thrown).

- **printStackTrace()** prints a copy of the stack trace information on the console.

- **getStackTrace()** returns a String object containing a copy of the stack trace information.