# Exceptions

# Overview

- Reasons for applications to fail:
  - Programmers didn't anticipate possible problems
  - Programmers didn't test enough
  - Bad data from users
  - Corrupt files, network connection issues, hardware device failures, etc.

# Motivation

```
int status = loadTextFile();
if (status != 1) {
    // something unusual happened, describe it
    switch (status) {
        case 2:
            System.out.println(“File not found”);
            break;
        case 3:
            System.out.println(“Disk error”);
            break;
        case 4:
            System.out.println(“File corrupted”);
            break;
        default:
            System.out.println(“Error”);
    }
} else {
    // file loaded OK, continue with program
}
```

- Problems:
  - Many nested if-else or switch-case blocks.
  - Documentation of all return codes.
  - Code that manages errors obscures the business logic code.
  - Invoker method is obliged to deal with error handling.
  - Return value is taken.

# Exceptions

- Exceptions are modeled as classes in a Java application.
- JRE comes with many built-in exceptions.
- **Throwable** sits at the top of the exception class hierarchy; **Error** and **Exception** extend from it.
- Errors are internal and involve the virtual machine; they're usually fatal to the program.
- Exceptions are more relevant to the program and subclasses fall into two groups:
  - **Checked** exceptions
  - **Unchecked** (runtime) exceptions

# Unchecked (runtime) exceptions

- Usually occur because the code isn't very robust.
- Extend the **RuntimeException** class.
- The compiler doesn't enforce exception management when using methods that throw runtime exceptions.
- Examples:
  - ArrayIndexOutOfBoundException should never be thrown if properly checking to make sure that the code stays within the bounds of an array.
  - NullPointerException happens when using a variable that doesn't refer to any object.

# Checked exceptions

- Used to indicate events that are "expected“.
- The compiler enforces exception management when using methods that throw checked exceptions. If not dealt with, the code won't compile.
- Example:
  - The createNewFile() method of the File class could throw an IOException in case  e.g. there is no free space available on the hard drive.

# Handling exceptions with try-catch

- Definition: “Try this bit of code that might cause an exception. If it executes okay, go on with the program. If the code doesn't execute, catch the exception and deal with it.”

  Example:

```
try {
    float in = Float.parseFloat(input);
} catch (NumberFormatException nfe) {
    System.out.println(input + “ is not a valid number.”);
    nfe.printStackTrace();
}
```

# Handling multiple exceptions

- A single catch statement that catches a specific type of exception "e" actually catches all exceptions which are subclasses to exception "e".
- Handling multiple exceptions that don't belong to the same exception class hierarchy is achieved by using multiple catch statements.
- Example:

```
try {
    // code that might generate exceptions
} catch (IOException ioe) {
    System.out.println(“Input/output error”);
    System.out.println(ioe.getMessage());
} catch (ClassNotFoundException cnfe) {
    System.out.println(“Class not found”);
    System.out.println(cnfe.getMessage());
} catch (InterruptedException ie) {
    System.out.println(“Program interrupted”);
    System.out.println(ie.getMessage());
}
```

# The “finally” clause

- In case there exists code which needs to be executed regardless of whether an exception is thrown or not, it can be placed inside a **finally** block. The finally block is usually used for releasing some resource.

  Example:

```
try {
    readTextFile();
} catch (IOException ioe) {
    // deal with IO errors
} finally {
    closeTextFile();
}
```

# Declaring method that throw exceptions

- Indicating that a method throws an exception is achieved using the **throws** keyword after the closing parenthesis of the method followed by the name or names of the exceptions.
- Both checked and unchecked exceptions can be declared as thrown.
- Example:

```
public boolean getFormula(int x, int y) throws NumberFormatException {
    // body of method
}

public boolean storeFormula(int x, int y) throws NumberFormatException, EOFException {
    // body of method
}
```

# Passing on exceptions

- There are times when it doesn't make sense for method "m" to deal with an exception. It might be better for the method invoking method "m" to deal with the exception. In such cases, the exception can be declared using the "throws" keyword.
- Rule of thumb: if you don't know how to deal with an exception intelligently, throw it.

# “throws” and inheritance

- If a method overrides a method in a superclass that includes a throws clause, the overriding method isn't required to throw the same set of exceptions. The reason is that the new method might possibly "deal" better with exceptions rather than just throwing them.

  Example:

```
public class RadioPlayer {
    public void startPlaying() throws SoundException {
        // body of method
    }
}

public class StereoPlayer extends RadioPlayer {
    public void startPlaying() {
        // body of method
    }
}
```

# Throwing exceptions

- Apart from the JVM and the JRE class methods, exceptions can be thrown by the application code classes as well.

  Example:

```
NotInServiceException nise = new NotInServiceException();
throw nise;
```

# Creating own exceptions

- The exceptions provided within the JRE might not be suitable to describe an exception event that is specific to the program business logic. Own exceptions can be defined.
- Custom exceptions should extend either from Exception, RuntimeException or from any other exception which is close to the nature of the problem.
- Example:

```
public class SunSpotException extends Exception {
    public SunSpotException() {}
    public SunSpotException(String msg) {
        super(msg);
    }
}
```

# Exercises

# Exercise: ZipCodeWithExceptions

- Modify the Zip Code application so that it throws an exception if an invalid argument is passed.