AR3: How and to what extent have you considered defensive programming? Discuss and show examples from your code.

Unsere Applikation ermöglicht es Tabellendokumente zu erstellen, bearbeiten, löschen, in anderen Dateiformaten zu speichern etc. Eine große Menge an Funktionen birgt auch eine Fülle an potenziellen Fehlern. In der defensiven Programmierung geht es grundsätzlich darum, möglichst alle Fehleingaben eines Benutzers abzufangen und sinnvoll darauf zu reagieren. Sowohl asserts, als auch try-catch Blöcke und viele andere wichtige Error-Handling-Techniken ermöglichen eine defensive Programmierung. Einige davon haben wir auch in unserem Projekt angewendet. Einige Beispiele dafür werden hier nun präsentiert:

```
 */
public Worksheet addSheet(String name) {
        if (worksheetCollection.containsKey(name)) {
                throw new IllegalArgumentException("Sheet with name " + name + " already exists");
        }

        if (diagramCollection.containsKey(name))
                throw new IllegalArgumentException("Diagram with name " + name + " already exists");
```

1.) Klasse: Worksheet, Methode: addSheet

Hier verwenden wir Exceptions. Will der User ein Worksheet speichern dessen Name schon im Workbook existiert, wird eine Exception mit „Sheet with name – Worksheetname – already exists“ geworfen. Das gleiche Prinzip gilt auch für Diagramme. Somit kann sich der User sicher sein, dass jeder Worksheetname bzw. Diagrammname in einem Workbook nur einmalig auftritt.

```
switch (token) {
        case MINUS:
                _readToken();
                r = new ExpressionTree(TOKEN.MINUS, _readTerm());
                break;
        case PLUS:
                _readToken();
                r = _readTerm();
                break;
        case LEFT_BRACKET:
                _readToken();
        .....
        .....
        case RANGE:
                r = new ExpressionTree(token, (Range) data);
                _readToken();
                break;
        default:
                throw new IllegalArgumentException("unexpected element " + token.toString() + ": "
                                + this.token.toString() + "," + this.charPosition + "," + this.cellFormula);
        }
```

2.) Klasse: ExpressionTree, Methode: _readTerm()

In der Klasse ExpressionTree verwenden wir sowohl Exceptions, als auch andere Error-Handling-Methoden (siehe Beispiel 2). Hierbei haben wir für jeden eingegebenen Wert eine entsprechende Reaktion parat. Wir verwenden hier

einen switch-case-Block um den „Wert“ des Token (das ist ein Enum-Wert der verschiedene mathematische Begriffe beispielsweise Plus, Minus, Between, etc enthält) zu bestimmen. Kann kein passender Case gefunden werden, so kommt der Default-case zum Einsatz. Dieser Default-Case wirft wiederum eine Exception.

```
     */
    public void openFile(String filePath) throws IllegalArgumentException
    {
            try {
                    ObjectInputStream ois = new ObjectInputStream(new FileInputStream(filePath));
                    activeWorkbook = (Workbook)ois.readObject();
                    ois.close();
                    fileActualName = filePath;
            }
            catch (FileNotFoundException e) {
                    throw new IllegalArgumentException("File not found " + filePath);
            }
        catch(ClassNotFoundException e) {
                    throw new IllegalArgumentException("Illegal DataSource " + filePath);
            }
            catch (IOException e) {
                    throw new IllegalArgumentException("IO Exception in DataSource " + filePath);
            }

            loadModel();
    }
```

3.) Klasse: Application, Methode: openFile

In diesem Ausschnitt wird ein try-catch-Block verwendet. Sobald der User eine Csv-File öffnet, wird diese Methode ausgeführt. Bei Erfolg wird die csv-File im Graphical-User-Interface angezeigt, misslingt die Aktion so wird der Fehler aufgefangen und eine entsprechende Fehler-Meldung wird retourniert. Zum einen kann beispielsweise die Datei nicht aufgefunden werden, zum anderen kann es sich um eine fehlerhafte Datei oder um Probleme beim Auslesen handeln.

```
class CsvWriteUtility {

 public static void convertWorkSheetToCsv(Worksheet worksheet, FileWriter writer) throws IOException {
```

4.) Klasse: CsvWriteUtility, Methode: convertWorkSheetToCsv

Falls es Probleme beim Input- oder Output-Stream von Daten gibt, wird auch in diesem Fall eine spezifizierte Exception geworfen. Sollte dies wirklich eintreffen so könnte man beispielsweise noch eine catch-Methode integrieren um den Fehler entsprechend zu behandeln.

```
String previous = this.cellFormula;
try {
        cellFormula = formula;
        cellExpression = ExpressionTree.parse(this, formula);
        cellInputDataType = cellExpression.getDataType();
        calculateCellExpression();
}
catch (IllegalArgumentException e) { this.cellFormula = previous; cellExpression = null; throw e; }

try {
        parentWorksheet.getParentWorkbook().calculateReferenceDependencies(this);
} catch (IllegalArgumentException e) {
        cellFormula = null;
        cellExpression = null;
        this.parentWorksheet.getParentWorkbook().removeReferenceDependencies(this);
        throw e;
```

5.) Klasse: Cell, Methode: setFormula

Auch in dieser Klasse verwenden wir wiederum einige try-catch-Blöcke um relevante Fehler zu entdecken und sie entsprechend zu behandeln. Der Verbesserungsvorschlag der im vorherigen Punkt erwähnt wurde, konnte hier umgesetzt werden. Wie man sehen kann wird hier mit einer Zellformel gearbeitet. Ist die Formel fehlerhaft oder verursacht sie einen Fehler, so wir die entsprechende Exception aufgefangen und die Variablen mit einem null-Wert neutralisiert.

6.) Resümee:

Natürlich können wir noch an unserem defensiven Programmierstil arbeiten, doch schon jetzt konnten wichtige Schritte in Bezug auf erfolgreicher Nutzung von Error-Handling-Methoden setzen. Im nächsten Projekt wäre es sicherlich vorteilhaft eine eigene „Exception“-Klasse zu generieren um die Logik der Fehlerkorrektur zu zentralisieren. Ein weiterer wichtiger Schritt wird es sein, die Applikationstestung auszuweiten und verschiedene Asserts in der Entwicklungsstufe zu Testzwecken einzubauen. Im weiteren Sinne sollten wir uns auch verdeutlichen, dass der Einsatz von Error-Handling-Methoden nicht im Nachhinein, sondern schon beim Quellcodeschreiben geschehen sollte. Schlussendlich muss man aber ganz ehrlich sein und betonen, dass wir uns mit den verschiedenen Error-Handling-Methoden erst seit kurzem intensiver auseinandersetzen und einiges an Erfahrung sammeln müssen.