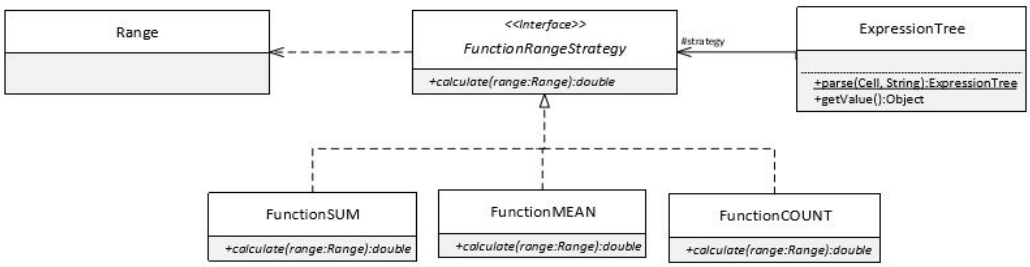
Strategy Pattern

Wir haben uns entschieden in unserem Projekt folgende Design Patterns zu implementieren:

- Strategy Pattern
- Observer Pattern
- Factory Pattern

Das **Strategy Pattern** wird in der Klasse Expression Tree benutzt. Durch dieses Pattern ist es uns möglich die Funktion während der Laufzeit zu definieren und dementsprechend zu nutzen. Das folgende Diagramm zeigt in abstrahierter Weise die Nutzung des Patterns.



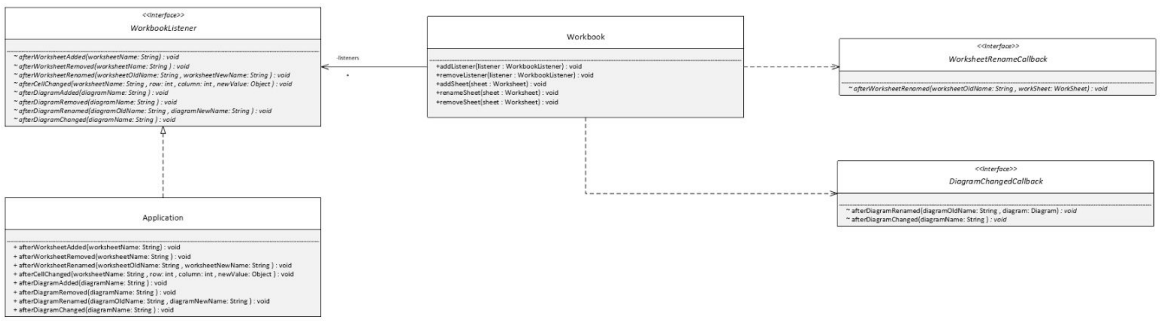
Die Klasse ExpressionTree implementiert alle Funktionalität die erforderlich ist um Formeln zu verarbeiten und zu parsen. In der Klasse befindet sich auch folgendes Switch Statement:

```
case FUNCTION:

    FunctionRangeStrategy strategy;
    switch ((FUNCTION) this.data) {
    case SUM:
        strategy = new FunctionSUM();
        break;

    case COUNT:
        strategy = new FunctionCOUNT();
        break;

    case MEAN:
        strategy = new FunctionMEAN();
        break;

    default:
        strategy = null;
        break;
    }

    return strategy.calculate((Range) this.expressions.get(0).data);
}
```

In diesem Statement werden während der Laufzeit Objekt instantiiert die das FunctionRangeStrategy Interface implementieren. Dadurch ist es möglich die Funktionalität der Berechnung (calculate()) in separate Klassen auszulagern.

Observer Pattern

Wir benutzen das **Observer Pattern** um Änderungen an unser GUI zu kommunizieren. Dadurch ist es möglich dass ein numerischer Wert einer Zelle geändert wird und eine Formel die von der Zelle abhängig ist dementsprechend neu berechnet wird. Insgesamt werden drei Interfaces implementiert um das Observer Pattern in unserer Applikation zu realisieren. Im folgenden Diagramm sind die einzelnen Details ersichtlich.



Das Interface ***WorksheetRenameCallback*** und ***DiagramChangedCallback*** werden von keiner Klasse implementiert. Die Interfaces hingegen werden in einem Worksheet und Diagram (beides Klassen) Kontruktor als Parameter benutzt.
Beispiel (Worksheet):

```
public Worksheet(String name, Workbook workbook, WorksheetRenameCallback worksheetRenameCallback)
```

Die Funktionalität der Interfaces wird direkt beim Aufruf des Konstruktors definiert.

```
worksheetCollection.put(name, new Worksheet(name, this, new WorksheetRenameCallback() {
    @Override
    public void afterWorksheetRenamed(String worksheetOldName, Worksheet sheet) {
        renameSheet(worksheetOldName, sheet);
    }
}));
```

Factory Pattern

Wir benutzen für die Erstellung von Diagrammen das Factory Pattern. In der Datei Diagram.java werden zwei Klassen definiert. Eine davon ist abstrakt und heißt DiagramCreator. Die Klasse die die abstrakte Klasse DiagramCreator implementiert befindet sich in der Datei DiagramBar und DiagramLine. Details sind aus dem folgenden Diagramm ersichtlich.