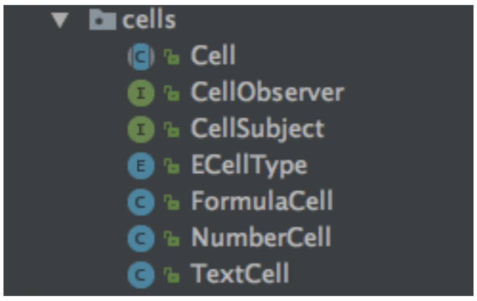
Desing Patterns

We use observer pattern multiple times:

Resource Observer pattern to listen for changes in locale and reload localized strings.
/g2017s_se2_0303/src/utilities/ResourceObserver.java

```
        public interface ResourceObserver {

    void updateFromResources();
}

public interface ResourceSubject {

    void registerObserver(ResourceObserver observer);
    void unregisterObserver(ResourceObserver observer);

    void notifyObservers();
}

public class R implements ResourceSubject{
```

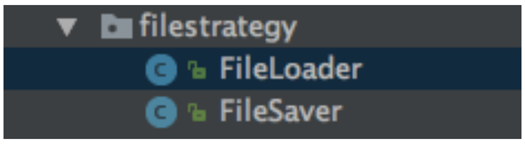
Cell Observers are cells that listen to each other for value changes. Formula cell for example are observers and subjects at the same time.
/g2017s_se2_0303/src/cells/FormulaCell.java



```
▼ cells
    Cell
    CellObserver
    CellSubject
    ECellType
    FormulaCell
    NumberCell
    TextCell
```

```
public class FormulaCell extends Cell implements CellObserver, CellSubject {
/**
* Observer interface methods
* */
@Override
public void update() {
    //update the subscribed Observers
    notifyCellObserver();
}
```

We use strategy pattern for choosing how to load or save files by dispatching commands to either the CSV Parsing Module or the ODS Parsing Module.
/g2017s_se2_0303/src/swingfrontend/filestrategy/FileLoader.java



```
▼ filestrategy
    FileLoader
    FileSaver
```

The respective classes then utilize switch cases to decide which parser for example is being required.

We use abstract Factory Pattern to create Cell Factories (which in turn create cells) and Formula Factories to create formulas.

aldinbradaric/g2017s_se2_0303/src/factory/

```
public abstract class AbstractFactory {

    abstract public Cell createCell(ECellType eCellType) ;

    abstract public void createFunction(EFormulaType eFormulaType);
}
public class CellFactory extends AbstractFactory {

    @Override
    public Cell createCell(ECellType eCellType) {
        switch (eCellType){
            case FORMULA:
                return new FormulaCell();
            case NUMBER:
                return  new NumberCell();
            case TEXT:
                return new TextCell();
        }
```

We make extensive use of the Singleton pattern because it´s just super awesome. And we use it to access modules like the resources class or the main jframe from any point in the application without passing the parameters through long function call chains far away from the needed fields.

/g2017s_se2_0303/src/utilities/R.java

```
private static R singletonResource = null;
private R(){
    observers = new ArrayList<>();
    currentLocale = Locale.GERMAN;
    resBundle = ResourceBundle.getBundle("Labels",currentLocale);
}
public static R getR()
{
    if(singletonResource == null){
        singletonResource = new R();
    }

    return singletonResource;
}
```