THE ADAPTER PATTERN

JAVAFX TABLES ARE QUITE NEAT

TableView IS THE STANDARD JAVAFX TABLE CONTROL

Company	Market Cap	Revenue	Growth Rate
Google	427.3	17.72	11.0
Amazon	243.9	23.18	20.0
Apple	657.7	49.6	33.0
Alibaba	155.0	3.27	28.0
Facebook	259.4	4.04	39.0
Twitter	18.5	0.502	61.0

TABLEVIEWS HAVE 2 IMPORTANT COMPONENTS

1. A BUNCH OF TableColumn OBJECTS

JAVAFX TABLES

ARE QUITE NEAT

TableView IS THE STANDARD JAVAFX TABLE CONTROL

Company	Market Cap	Revenue	Growth Rate
Google	427.3	17.72	11.0
Amazon	243.9	23.18	20.0
Apple	657.7	49.6	33.0
Alibaba	155.0	3.27	28.0
Facebook	259.4	4.04	39.0
Twitter	18.5	0.502	81.0

THE REASON WE SAID JAVAFX TABLES ARE NEAT IS BECAUSE THERE IS A REALLY CLEVER WAY PROVIDED FOR WRITING THE ADAPTER CODE

TABLEVIEWS HAVE 2 IMPORTANT COMPONENTS

1. A BUNCH OF TableColumn OBJECTS

JAVAFX MAKES IT VERY EASY FOR A MODEL TO MAKE ITS DATA AVAILABLE IN ROWS AND COLUMNS

AS THE NAME OF THE CLASS SUGGESTS, THE TABLEVIEW IS MERELY THE VIEW

2. DATA, I.E. A BUNCH OF ROWS AND COLUMNS, THIS DATA SITS IN THE MODEL OF COURSE

NOW THE DATA WAS NOT ORIGINALLY WRITTEN TO BE USED IN A TABLE, SO WE NEED LITTLE BITS OF

ADAPTER CODE

WHICH TAKES DATA FROM THE MODEL, AND MAKES IT AVAILABLE IN THE FORM OF ROWS AND COLUMNS AS NEEDED BY THE TABLE

IN GENERAL, WRITING ADAPTER CODE IS TEDIOUS, REPETITIVE AND PRONE TO ERRORS

WIRING MODEL TO TABLEVIEW

TYPICALLY, WE HAVE A CLASS WHERE EACH OBJECT REPRESENTS ONE ROW OF DATA IN THE TABLE

```
public static class CompanyMetrics
    private StringProperty name = new SimpleStringProperty();
    private DoubleProperty mktCap = new SimpleDoubleProperty();
    private DoubleProperty revenue = new SimpleDoubleProperty();
    private DoubleProperty growthRate = new SimpleDoubleProperty();

    public CompanyMetrics(String name, double mktCap, double revenue, double growthRate) {
        this.name.set(name);
        this.mktCap.set(mktCap);
        this.revenue.set(revenue);
        this.growthRate.set(growthRate);
    }
```

CREATE AN OBSERVABLELIST OF THESE OBJECTS, SO THAT EACH OBJECT REPRESENTS ONE ROW

```
ObservableList<CompanyMetrics> data =
        FXCollections.observableArrayList(
                new CompanyMetrics("Google",427.3,17.72,11),
                new CompanyMetrics("Amazon",243.9,23.18,20),
                new CompanyMetrics("Apple",657.7,49.6,33),
                new CompanyMetrics("Alibaba",155,3.27,28),
                new CompanyMetrics("Facebook",259.4,4.04,39),
                new CompanyMetrics("Twitter",18.5,0.502,61)
        );
```

NOW - CREATE A BUNCH OF TABLECOLUMN OBJECTS..

..AND THEN WIRE UP EACH TABLECOLUMN TO THE CORRESPONDING PROPERTY OF THE CLASS

```
TableColumn companyColumn = new TableColumn("Company");
companyColumn.setMinWidth(100);
companyColumn.setCellValueFactory(new PropertyValueFactory<CompanyMetrics, String>("name"));

TableColumn marketCapColumn = new TableColumn("Market Cap");
marketCapColumn.setMinWidth(50);
marketCapColumn.setCellValueFactory(new PropertyValueFactory<CompanyMetrics, Double>("mktCap"));

TableColumn revenueColumn = new TableColumn("Revenue");
revenueColumn.setMinWidth(50);
revenueColumn.setCellValueFactory(new PropertyValueFactory<CompanyMetrics, Double>("revenue"));

TableColumn growthRateColumn = new TableColumn("Growth Rate");
growthRateColumn.setMinWidth(50);
growthRateColumn.setCellValueFactory(new PropertyValueFactory<CompanyMetrics, String>("growthRate"));
```

REMEMBER TO DO THIS FOR EACH PROPERTY

AND NOTE THAT THIS IS THE CRUCIAL STEP, WHICH HAPPENS VIA THE NAME OF THE PROPERTY

WIRING MODEL TO TABLEVIEW

THEN, ADD THE TABLECOLUMNS
AS WELL AS THE MODEL TO THE
TABLEVIEW

```
tableView.getColumns().addAll(companyColumn, marketCapColumn, revenueColumn, growthRateColumn);
tableView.setItems(data);
```

JAVAFX DOES ALL THE REST -
IT FIGURES OUT THAT THE OBSERVABLELIST
CORRESPONDS TO THE ROWS OF THE TABLE,

IT FIGURES OUT FOR EACH COLUMN OF EACH
ROW, WHICH PROPERTY OF THE UNDERLYING
OBJECT NEEDS TO BE USED

IT ALSO MAKES IT EASY TO WIRE
UP LISTENERS IN CASE YOUR TABLE IS
EDITABLE