Combined PDF:
Group Members: Ousman Jobe, Maddie Lebiedzinski, Dylan McCann, Hector Padilla

Next Meet: Sunday 10/22, Time TBD

Part 1 - User Stories(Dylan)

https://drive.google.com/file/d/1XTE-DFfRh6y_bFWarJqVBszJSPU6mfNt/view?usp=drive_link

Part 2 - UI Designs(Maddie)

https://www.figma.com/file/SQUP7goF69QxufMAhObWVL/Spreadsheet_UI_sketches?type=design&node-id=0-1&mode=design

Part 3 - UML Class Diagrams(Hector)

https://www.figma.com/file/LpHnyNrlLkuu6zdgP8RH0y/Phase-B-UML?type=whiteboard&node-id=0-1

Part 4 - TypeScript classes and interfaces corresponding to the UML Diagrams(Ousman)

```
import React from 'react';

enum NumericKeys {
ZERO = '0',
ONE = '1',
TWO = '2',
THREE = '3',
FOUR = '4',
FIVE = '5',
SIX = '6',
SEVEN = '7',
EIGHT = '8',
NINE = '9'
}

enum ActionKeys {
CLEAR = 'C',
EQUALS = '=',
DOT = '.'
}
```

```
enum OperatorKeys {
PLUS = '+',
MINUS = '-',
MULT = '*',
DIV = '/'
}

interface IExpression {
getRangeValues(firstCell: string, lastCell: string): Array<number>;
}

class CellReference implements ICell{
constructor(public rowNum: number, public colLetter: string) {}

  displayValue();
}

class SumExpression implements IExpression {
private firstCell: CellReference;
private lastCell: CellReference;

constructor(firstCell: CellReference, lastCell: CellReference) {
this.firstCell = firstCell;
this.lastCell = lastCell;
}

getRangeValues(): Array<number> {
return [];
}

calcSum(values: Array<number>): number {
return 0;
}
}

class AverageExpression implements IExpression {
private firstCell: CellReference;
private lastCell: CellReference;

constructor(firstCell: CellReference, lastCell: CellReference) {
this.firstCell = firstCell;
this.lastCell = lastCell;
```

```
}

getRangeValues(): Array<number> {
return [];
}

calcAVG(values: Array<number>): number {
return 0;
}
}

interface ICells {
displayValue(): string;

}

interface Cell {
value: string | number;
formula?: string;
isEditing?: boolean;
cellReference: CellReference;
displayValue(): string;

}

interface ICalculator {
pressNumericKey(key: NumericKeys): void
pressOperatorKey(key: OperatorKeys): void
pressActionKey(key: ActionKeys): void
display(): string
}

class CalculatorModel implements Calculator {
currentCell: Cell;

constructor(initialCell: Cell) {
this.currentCell = initialCell;
}

pressNumericKey(key: NumericKeys): void {
// Implementation needed
}
```

```
pressOperatorKey(key: OperatorKeys): void {
// Implementation needed
}

pressActionKey(key: ActionKeys): void {
// Implementation needed
}

display(): string {
return this.currentCell.displayValue();
}
}

interface Spreadsheet {
cells: { [key: string]: Cell };
selectedCells: cellReference[];
insertRow(row: number): void;
insertColumn(column: number): void;
deleteRow(row: number): void;
deleteColumn(column: number): void;
clearCell(cellReference: string): void;
evaluateFormula(formula: string): string | number;
}

class MySpreadsheet implements Spreadsheet {
cells: { [key: string]: Cell } = {};
selectedCells: string[] = [];
utils: SpreadsheetUtils;

constructor() {
this.utils = new SpreadsheetUtils(this);
}

insertRow(row: number): void {
}

insertColumn(column: number): void {
}

deleteRow(row: number): void {
}
```

```
deleteColumn(column: number): void {
}

clearCell(cellReference: string): void {
}

evaluateFormula(formula: string): string | number {
return 0;
}
}

class SpreadsheetUtils {
spreadsheet: Spreadsheet;

constructor(spreadsheet: Spreadsheet) {
this.spreadsheet = spreadsheet;
}

exportToSpreadsheet(): void {
}

importFromSpreadsheet(data: string): void {
}

colorCell(cellReference: string, color: string): void {
}
}

class SpreadsheetUI extends React.Component {
render() {
return (
<div>
{/* Render the spreadsheet UI here */}
</div>
);
}
}
```