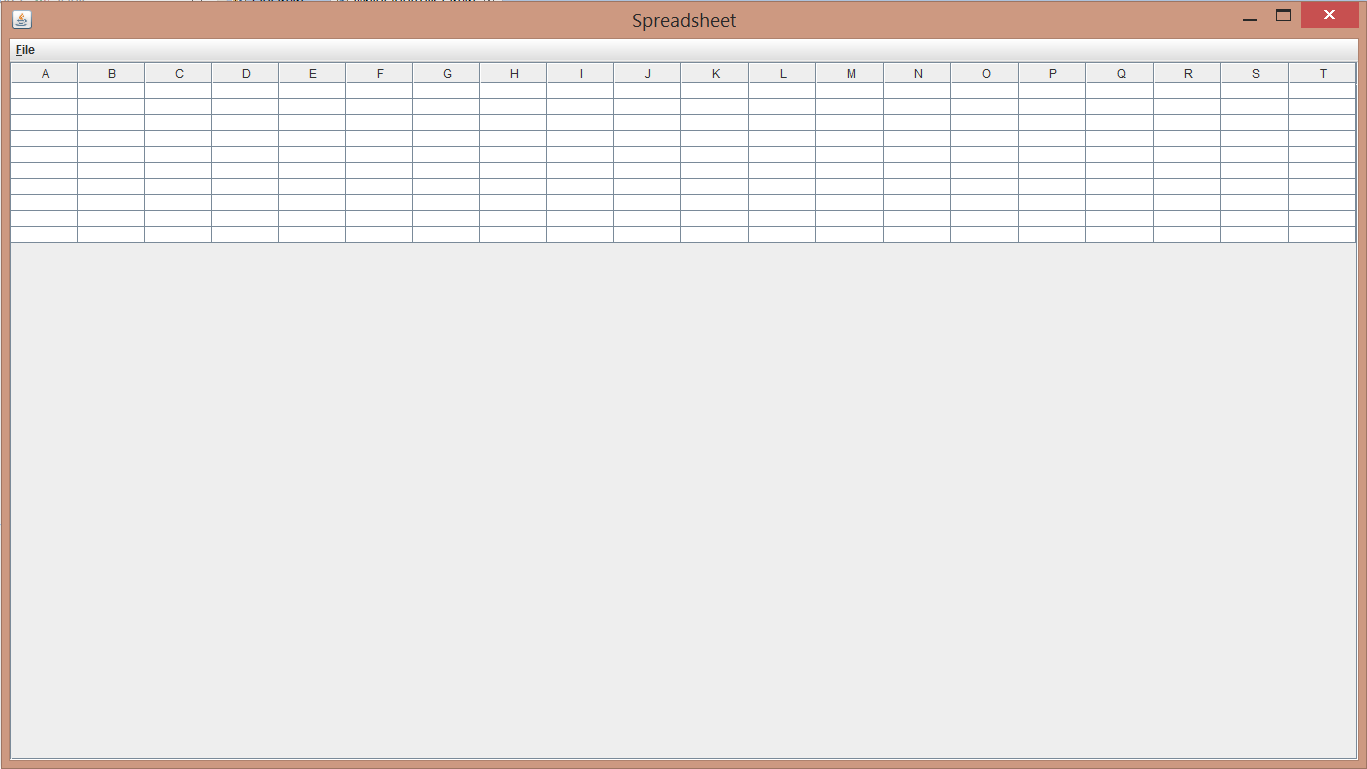
Spreadsheet in Java

Door Team Scarlett



Martijn Gribnau - 4295374

Mitchell Olsthoorn - vul in

Roy Klip - vul in

Ike Rijsdijk - vul in

Robin Borst - vul in

Alan van Rossum - vul in

Bachelor Technische Informatica, 2013-2014

OOP Project

**Inhoudsopgave**

1.

2.

3.

4.

5.

6.

**Algemeen: Verloop van het project**

**Planning**

**Samenwerking**

**Rolverdeling**

**Pair-programming**

**Apart thuis etc.**

**Communicatie**

**Whatsapp, toch geen Trello, github, face-to-face**

**wekelijks/dagelijks**

**verslagen opgemaakt? (sidenote\_martijn: bv bij team vergadering)**

**Versiebeheer, heeft het geholpen**

**Ontwerp proces**

**Welke problemen opgelost**

**Welke technologische afwegingen**

**UML, hoe heeft het wel/niet geholpen**

**test coverage(inclusief screenshot package of klasse niveau plus beargumenteer klassen die heel goed of niet heel goed gecovered zijn)**

**Verbeterpunten**

**Hoe kan onze software verbeterd worden?**

**Hoe kan het vak OOP Project verbeterd worden?**

**Hoe kan ons proces verbeterd worden (planning, overleg, commits, uml gebruik)**

**Individuele feedback**

**Reflectie op functioneren binnen het team pp, sterke/zwakke punten, conflicten etc.**

**Martijn**

**Mitchell**

**Roy**

**Ike**

**Robin**

**Alan**

**Ontwerpkeuzes**

“Maak een spreadsheet in Java”. Deze opdracht laat vrijwel alle keuzes over aan de ontwerpers. We hebben daarom ons best gedaan om voor elk onderdeel van het project de best passende methode te gebruiken. Hier volgen onze belangrijkste keuzes.

**De opslag van inhoud van de cellen**

Dit was een van de eerste problemen waar we tegenaan liepen. Eerst lag het voor de hand om een tweedimensionale Array van Cell-objecten te maken, in de vorm van Cell[][]. Dit leek ons in eerste instantie de beste optie, omdat het zo makkelijk was om de inhoud van een willekeurige cel op coördinaten (x,y) op te vragen. Je kan dan namelijk gewoon Cell[x][y].getValue opvragen, en zo zeer snel de benodigde waarde ontvangen.

Het nadeel van deze methode was dat we de hele Array vol moesten zetten met null-values voor lege cellen. Nu kwam het idee van een tweedimensionale ArrayList tevoorschijn. Een ArrayList in een ArrayList dus. Het voordeel hiervan was dat je alleen de waardes van volle cellen hoeft op te slaan. Zodra een Cell een value heeft, voeg je hem toe aan de ArrayList. Het nadeel was echter dat elke Cell nu ook een x en y-coördinaat moet hebben, aangezien de index van de Cell niet altijd overeenkomt met de x- en y-coördinaten van de Cell. Het grote nadeel is dat het niet makkelijk is om de inhoud van een willekeurige ccel op coördinaten (x,y) op te vragen. Er moet namelijk gebruik worden gemaakt van een for-loop in een for-loop, die elke index afgaan en checken of de Cell.getX en Cell.getY de coördinaten zijn die zijn opgevraagd. Dus een ArrayList gebruikt minder geheugen, maar is wel langzamer.

In week 4 leerden we echter de ultieme methode kennen: de HashMap. In een HashMap link je Objecten niet aan een index, maar aan een key. Door in deze key een x- en y-coördinaat te zetten, bijvoorbeeld in de vorm van (11,2), link je elke Cell-object aan zijn betreffende coördinaten. Hierdoor is een willekeurige cel op coördinaten (x,y) snel op te vragen, zonder loops die elk Object in de lijst afgaan. Bovendien slaat een HashMap alleen de Cellen op met een waarde, net als een ArrayList, dus is een HashMap qua geheugengebruik ook veel efficiënter.

Onze HashMap heeft dus de snelheid van een Array[][], en de geheugencapaciteit van een ArrayList<ArrayList<>>.

**Git of SVN**

- Git is gedecentraliseerd

**Aparte klassen voor formules**

- modulair

**Parser**

- Patterns / recursive voor nested formulas