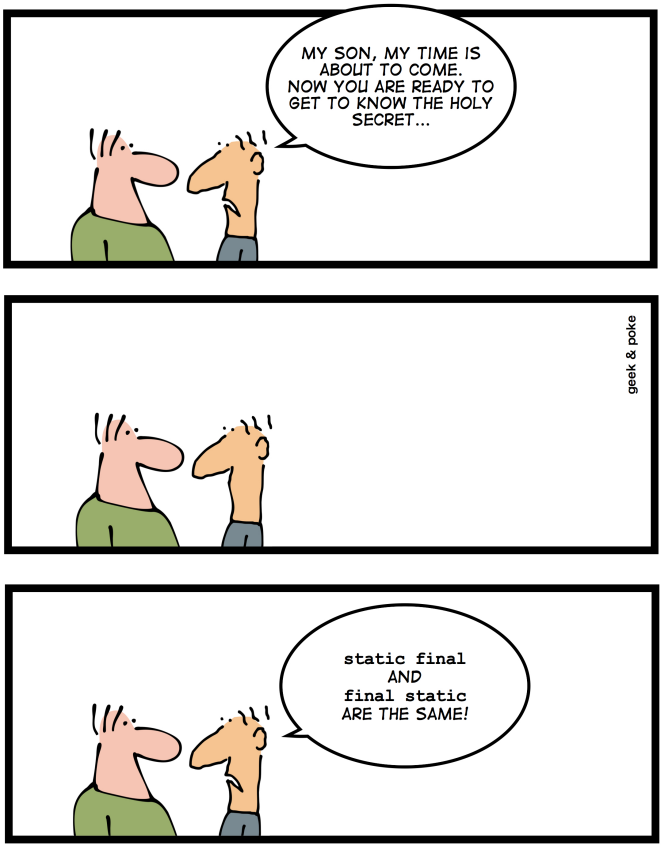
Universiteit van Amsterdam
Inleiding Programmeren

Opgave 5

José Lagerberg - Robin de Vries - Devin Hillenius - Stephen Swatman - Folkert van Verseveld

Polynomen

Heritage of a Coder



Deadline: zaterdag 5 oktober 2019, 18:00

1 Leerdoelen

Aan het einde van deze opdracht kan je:

- een goed objectgeoriënteerd programma schrijven,
- nadenken over de efficiëntie van zelfbedachte algoritmen,
- een gegeven interface implementeren,
  - werken met de standaard Comparable interface,
- gebruik maken van *getters* en *setters*
- gebruik maken van bestaande bibliotheken:
  - unit tests schrijven door middel van JUnit,
  - parsen van command line-argumenten met behulp van `commons-cli`.

2 Introductie

De afgelopen weken hebben jullie helemaal zelf enkele relatief eenvoudige programma's geschreven die geheel bestonden uit de code die jullie zelf hadden geschreven. Als programmeur zul je echter vaak gebruik moeten maken van andermans code om te voorkomen dat je het wiel opnieuw moet uitvinden – je bent niet de enige programmeur op aarde en er bestaan voor heel veel problemen al oplossingen!

Deze week gaan jullie een programma schrijven waarmee je (twee) polynomen kunt inlezen en wiskundige operaties kunt toepassen op deze ingelezen polynomen. Wat het programma precies doet is afhankelijk van de argumenten die je door middel van zogenaamde *vlaggen* (dit zijn commando's met streepjes ervoor; als je met Linux werkt ken je ze misschien wel) meegeeft tijdens het aanroepen. Omdat de mogelijkheden die Java hiervoor ingebouwd heeft beperkt zijn, gaan we gebruik maken van een bibliotheek: Apache Commons CLI[1].

Daarnaast gaan jullie de correctheid van je eigen implementatie controleren door middel van JUnit tests. JUnit biedt de mogelijkheid om eenvoudig tests te schrijven voor jouw klassen. Veel dingen die je bij je Breuken-opgave nog zelf moest implementeren, bijvoorbeeld het printen of een test wel of niet gelukt is, wordt je door JUnit uit handen genomen.

Tip Lees de opgave eerst tot het einde door voordat je begint met het schrijven van code.

Tip Vanaf deze opdracht is het belangrijk rekening te houden met de access modifiers binnen klassen. Onthoud dat je het beste zo veel mogelijk private kunt maken.

2.1 Framework

Om je een beetje op gang te helpen, wat voorbeelden te bieden en ook om je wat werk uit hand te nemen leveren we je voor deze opdracht een *framework* aan. Dat is een mapje waarin al het een en ander voorgeprogrammeerd staat door de vakassistenten. Je kunt daarop voortbouwen in plaats van helemaal vanaf het begin te moet beginnen. Je kunt het framework vinden op Canvas. De structuur van het framework is als volgt (de uitleg zal onduidelijk zijn tot je de rest van de opdracht hebt gelezen):

`invoer/` In dit mapje staan een aantal voorbeeldpolynomen die je als invoer kunt gebruiken voor je programma.

`lib/` In dit mapje staan softwarebibliotheken die je gaat gebruiken. Je hoeft hier **geen** aanpassingen in te maken, je hoeft er zelfs niet naar te kijken.

`windows/` Hier kun je wat scripts vinden die voor Windows gebruikers belangrijk zijn. Gebruik je Linux, dan is dit **niet** relevant.

`Opgave5.java` Dit is het uitvoerbare programma waarin je invoer van de gebruiker gaat afhandelen.

[1] `http://commons.apache.org/proper/commons-cli/`

`Paar.java` Dit is een enkele term in een polynoom die je zelf moet implementeren.

`Polynoom.java` Hierin ga je de polynoom zelf implementeren.

`PolynoomInterface.java` Dit is de interface die de polynoom moet implementeren. Pas hier **niets** aan!

`TestPolynoom.java` Hierin staan de tests die uitgevoerd worden voor de Polynoom klasse.

Overal in het framework waar wij verwachten dat je aanpassingen maakt hebben we dat aangegeven met `TODO` comments. Als je ergens anders aanpassingen maakt is dat geen probleem, deze comments dienen enkel als handvat.

2.2 Compileren

Je bent waarschijnlijk gewend te compileren met een commando als `javac *.java`. Omdat we in deze opdracht ook bibliotheken gebruiken van buitenaf, zal dit niet naar behoren werken en zul je op die manier compilerfouten krijgen. Het is daarom belangrijk dat je, elke keer dat je een terminal opent om aan deze opdracht te werken, het volgende commando uitvoert: "`export CLASSPATH=.:lib/*`". Dit zorgt ervoor dat Java klassen gaat zoeken in de `lib` map en zo niet in de war raakt.

2.2.1 Windows

Voor de Windows gebruikers zijn er in het mapje `windows` batch scripts toegevoegd die dit instellen. Kopieer de bestanden vanuit dit mapje naar de bovenliggende map (daar waar alle java-bestanden staan).

Als je vervolgens cmd opent tik je niet "`javac.exe ...`" maar "`compile.bat`" om de bestanden te compileren. Om de Opgave5.class uit te voeren voer je niet "`java.exe Opgave5`" uit maar "`opgave5.bat`" en voor TestPolynoom.class voer je "`testpolynoom.bat`" uit.

2.3 Opfrisser polynomen

In de wiskunde is een polynoom (of veelterm) in één variabele x een uitdrukking van de vorm:

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + ... + a_2 x^2 + a_1 x + a_0$$

Hierin is n een natuurlijk niet-negatief getal en de getallen a_k $(k = 0 \dots n)$ zijn de coëfficiënten van de polynoom. Zo'n polynoom bestaat dus uit een som van termen $a_k x^k$, vandaar de naam veelterm. Als de coëfficiënt van de hoogste macht van x ongelijk is aan 0, heet n de graad van de polynoom. Met andere woorden, de graad van de polynoom is gelijk aan de grootste macht waarvoor de coëfficiënt niet nul is.

3 Klasse `Paar`

Om de individuele termen binnen een polynoom op te slaan, is het nodig om een Paar klasse te maken. Binnen een Paar object slaan we een term op als een paar, en elk van deze paren bevat een coëfficiënt (een reëel getal) en een macht (een natuurlijk niet-negatief getal).

Omdat we de termen van een polynoom in de juiste volgorde moeten weergeven, moeten we paren kunnen sorteren. Deze keer gebruiken we geen eigen sorteeralgoritme maar een methode die al in Java zit[2]. Een lijst kan echter alleen op deze manier gesorteerd worden als het elementen van een klasse bevat die de `Comparable` interface implementeert. Om de termen van een polynoom te kunnen sorteren naar de macht moet de Paar klasse `Comparable` gemaakt worden en moet in deze klasse de `compareTo` methode geïmplementeerd worden. In deze methode moeten de machten en eventueel de coëfficiënten van de termen vergeleken worden. De return waarde van de `compareTo` methode moet zo ingesteld worden dat de termen in aflopende volgorde van de machten gesorteerd worden. Wat deze methode precies moet doen, kun je vinden in de documentatie van de `Comparable` interface[3].

[2] `https://docs.oracle.com/javase/8/docs/api/java/util/Collections.html`
[3] `https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html`

Implementeer nu de bovenstaande functionaliteit binnen Paar.java. Maak hierbij gebruik van de `TODO` comments, welke dienen als aanwijzing van wat er allemaal geïmplementeerd moet worden.

4 Klasse `Polynoom`

Aangezien Java geen klasse heeft om een polynoom in te representeren is het jouw taak een Polynoom klasse te schrijven. De volgende operaties moeten op die polynomen kunnen worden uitgevoerd:

- Optellen
- Aftrekken
- Vermenigvuldigen
- Differentiëren
- Integreren

Bij het integreren van polynomen heb je te maken met een integratieconstante `C`. In deze opgave moet je zorgen dat we een integratieconstante kunnen meegeven. Dit kun je bereiken door alle methoden van `PolynoomInterface.java` te implementeren. Let op dat je in `Opgave5.java` altijd 1 als integratieconstante neemt bij de commandline-optie `integreer`.

Naast de bovenstaande operaties moeten de polynomen ook met elkaar vergeleken kunnen worden, en moeten deze leesbaar kunnen worden geprint.

De termen van een polynoom moeten worden ingelezen uit een bestand en opgeslagen in een `ArrayList`. Het formaat van de invoerbestanden is een opeenvolging van regels met telkens een coëfficiënt en een macht zodat elke regel een term voorstelt. We gaan ervan uit dat de coëfficiënten van een polynoom in het bestand als reële getallen worden gespecificeerd en de machten ook. Het kan zijn dat er overbodige witruimte in de bestanden staat, dus houd daar rekening mee. Een voorbeeldpolynoom $P(x) = 5x^{10} - x - 10$ wordt als volgt opgeslagen (met dus één paar per regel):

```
5 ,10
−1,1
−10,0
```

De Polynoom klasse moet de bijgevoegde `PolynoomInterface` implementeren. Daarnaast heeft deze klasse een aantal constructoren, namelijk:

- een constructor die als argument een ArrayList meekrijgt,
- een lege constructor (dus zonder argumenten),
- een constructor die als argument een bestandsnaam als String meekrijgt en hier paren uit moet lezen,
- een constructor die als argument een lijst van doubles meekrijgt (te gebruiken voor je tests).

Denk hierbij, net als in de Breuken opgave, aan het chainen van de constructoren. Een mogelijke opbouw hiervoor is om de constructor die een ArrayList meekrijgt te gebruiken als hoofdconstructor. Je kunt dan, met behulp van eventuele hulp methodes om de invoer om te zetten in een ArrayList, vanuit de andere constructoren hiernaar chainen.

Een belangrijk onderdeel van de opdracht is het versimpelen van polynomen. Neem bijvoorbeeld de polynoom $P(x) = x^2 + 3x^2 + 1$. Deze polynoom is te versimpelen naar $P(x) = 4x^2 + 1$. Het is belangrijk dat jouw programma dit doet maar waar, wanneer en hoe dit gebeurt laten we aan jou. Wat belangrijk is, is dat dit net als in de breuken opgave, DRY is.

Bij het versimpelen van de polynoom is het belangrijk dat je efficiënt te werk gaat – als je een geneste for-loop nodig hebt voor het versimpelen doe je het hoogstwaarschijnlijk fout.

Operator	Lange vorm	Korte vorm
Optellen	`--optellen`	`-o`
Aftrekken	`--aftrekken`	`-a`
Vermenigvuldigen	`--vermenigvuldigen`	`-v`
Differentiëren	`--differentieer`	`-d`
Integreren	`--integreer`	`-i`

Table 1: De lijst met operatoren voor het `Opgave5` programma.

5 Klasse `Opgave5`

De `Opgave5` dient als *command line interface* (CLI), het is de manier waarop de gebruiker gebruik kan maken van de `Polynoom` klasse. Het `Opgave5` programma werkt door operaties en bestandsnamen te verwerken die *bij de aanroep van het programma* opgegeven worden. Het programma gebruikt dus *geen* `Scanner` objecten om gebruikersinvoer te vragen. De vijf verschillende operaties worden opgegeven als vlaggen zoals `--optellen` (korte versie `-o`) en `--differentieer` (korte versie `-d`). Deze vlaggen nemen geen argumenten. De complete lijst met vlaggen kun je vinden in tabel 1. Korte versies van vlaggen kun je ook combineren, dus `-oav` betekent optellen, aftrekken én vermenigvuldigen. Dit handelt de bibliotheek automatisch voor je af.

Alle overige invoer wordt gezien als een verzameling bestandsnamen die met spaties gescheiden zijn. Deze bestanden worden gebruikt als invoer voor de `Polynoom` constructor. Er dienen altijd één of twee polynomen te zijn. Als er maar één is, geeft een programma een foutmelding bij operaties die twee polynomen nodig hebben maar stopt het programma niet. Zie de voorbeelduitvoer om te zien hoe het programma zich moet gedragen. De logica die het optellen afhandelt hebben we alvast voorgedaan en voor de rest van de operaties is het vrijwel hetzelfde.

6 Testen met JUnit

Om de `Polynoom` klasse te testen gebruik je het JUnit systeem. Implementeer je eigen tests in `TestPolynoom.java` door methoden toe te voegen met de `@Test` *decorator* erboven net zoals de twee voorbeelden. Bij het uitvoeren van de `TestPolynoom` klasse worden al deze tests uitgevoerd en wordt het resultaat geprint. Schrijf zoveel mogelijk relevante tests. De `main` methode hoef je niet aan te passen.

7 Inleveren

Pak alle java en jar bestanden in (in bijvoorbeeld een .zip of .tar.gz archief), en lever dit archief in op Canvas.

A Voorbeelduitvoer

```
$ java Opgave5 --optellen --aftrekken invoer/polynoom1 invoer/polynoom2
Polynoom 1: x + 2.0
Polynoom 2: x^2 + x
Som: x^2 + 2.0 x + 2.0
Verschil: - x^2 + 2.0

$ java Opgave5 --optellen --differentieer invoer/polynoom1
Polynoom 1: x + 2.0
Polynoom 2: null
Polynoom 2 nodig om op te tellen!
Polynoom 1 gedifferentieerd: 1.0

$ java Opgave5 invoer/polynoom1
Polynoom 1: x + 2.0
Polynoom 2: null

$ java Opgave5 --differentieer --integreer invoer/polynoom1
Polynoom 1: x + 2.0
Polynoom 2: null
Polynoom 1 gedifferentieerd: 1.0
Polynoom 1 geintegreerd: 0.5 x^2 + 2.0 x + 1.0

$ java Opgave5 -oavdi invoer/polynoom1
Polynoom 1: x + 2.0
Polynoom 2: null
Polynoom 2 nodig om op te tellen!
Polynoom 2 nodig om op af te trekken!
Polynoom 2 nodig om op te vermenigvuldigen!
Polynoom 1 gedifferentieerd: 1.0
Polynoom 1 geintegreerd: 0.5 x^2 + 2.0 x + 1.0

$ java Opgave5 -oav invoer/polynoom1 invoer/polynoom2
Polynoom 1: x + 2.0
Polynoom 2: x^2 + x
Som: x^2 + 2.0 x + 2.0
Verschil: - x^2 + 2.0
Product: x^3 + 3.0 x^2 + 2.0 x

$ java Opgave5 --optellen invoer/polynoom1 invoer/polynoom2 invoer/polynoom3
Het aantal bestanden moet een of twee zijn.
```