Opdracht Peertutoring – P2W3

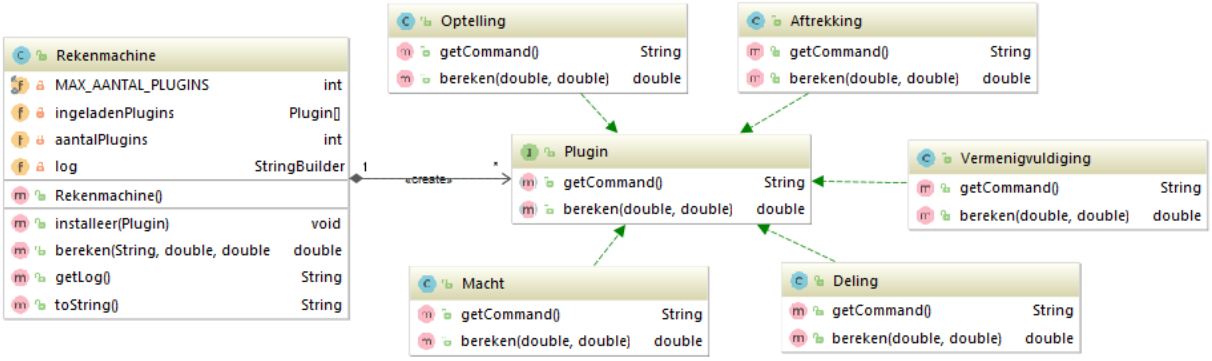
(Academiejaar '17 - '18)

Opgave: Rekenmachine

Situatie:

De bedoeling is dat we een rekenmachine gaan implementeren. Bij deze implementatie beschouwen we de rekenmachine als een verzameling **plugins** (zoals: optelling, deling, macht, …). Het moet ook mogelijk zijn om nieuwe plugins te installeren op de rekenmachine.
Je kan vertrekken vanuit het IntelliJ project dat meegeleverd is in de zip-file "peer_P2W3.zip"

UML klasse diagram:



OPMERKING: het attribuut **`log`** en de methode **`getLog`** in de klasse **`Rekenmachine`** is een onderdeel van opgave 4.

Basisopdracht

1. De interface **`Plugin`**:
   1.1. Voorzie volgende methoden in de interface **`Plugin:`**
   - een methode **`getCommand`** die een **`String`** retourneert en géén parameters heeft
   - een methode **`bereken`** die twee doubles (x en y) als parameters heeft en een **`double`** retourneert

   1.2. Maak nu in de package **`plugins`** de volgende vijf klassen: **`Optelling, Aftrekking, Deling, Vermenigvuldiging en Macht`**. Elk van deze klassen moet de interface **`Plugin`** implementeren op zijn eigen specifieke manier.
   Bijvoorbeeld: de klasse **`Optelling`**:

```
public class Optelling implements Plugin{
    @Override
    public String getCommand() {
        return "+";
    }
    @Override
    public double bereken(double x, double y) {
        return x + y;
    }
}
```

   Bedenk zelf nu nog 2 andere plugins die aan deze interface voldoen en implementeer beide klassen.

2. De klasse **`Rekenmachine`** bevat volgende attributen en methoden:
   - Een constante **`MAX_AANTAL_PLUGINS`** met de waarde 10
   - Een array **`ingeladenPlugins`**. In de default constructor wordt deze array klaargemaakt om evenveel **`Plugin`**-objecten te bevatten als de constante toelaat.
   - Een int **`aantalPlugins`** om het actuele aantal plugins in de array bij te houden.

   2.1. Werk volgende methoden verder uit:
   a) Zorg dat je via de methode **`installeer`** een plugin aan de array **`ingeladenPlugins`** toevoeg kan voegen. Doe hier eerst controle of de nieuwe plugin nog niet voorkomt in de array (door hun commando's te vergelijken) EN of de array nog niet vol zit. Geef eventueel een gepaste foutmelding.
   b) In de methode **`bereken`** moet je eerst onderzoeken of er een plugin bestaat voor het meegegeven commando. Indien dat NIET het geval is, toon je een gepaste foutmelding en je geeft de waarde 0 terug**`.`**
   Indien de plugin wel gevonden wordt, dan roep je de methode **`bereken`** daarvan op. Vervolgens ga je de uitkomst van die berekening zelf terugsturen als returnwaarde.
   c) Doe een override van de methode **`toString`** en zorg dat deze als uitvoer alle geïnstalleerde plugins toont als volgt:
   `Geïnstalleerde Plugins: + - * / ^`

3. De klasse **`TestRekenmachine`**
   3.1. Gebruik de klasse **`TestRekenmachine`** om je code te testen. Haal de code in de main-methode onder **`//Opgave3.1`** uit commentaar.
   Je zou onderstaande uitvoer moeten krijgen:

```
5,000000 + 2,000000 = 7,000000
5,000000 - 2,000000 = 3,000000
5,000000 * 2,000000 = 10,000000
5,000000 / 2,000000 = 2,500000
5,000000 ^ 2,000000 = 25,000000
Plugin ? niet geïnstalleerd.
Geïnstalleerde plugins: + - * / ^
```

(TIP: de opgave gaat nog verder op de volgende pagina ☺)

3.2. Pas de **`main`** verder aan (Baseer je op de verwachte uitvoer):

- Maak een lus waarin telkens alle geïnstalleerde plugins (TIP: **`toString`** methode van de rekenmachine) waaruit gekozen getoond worden, alsook waarin de gebruiker de gewenste berekening kan invoeren alsook om twee getallen (**`double`**, gescheiden door een spatie) in te voeren waarmee de berekening moet uitgevoerd worden.
    - Wanneer de gebruiker bij het keuze van berekening een 'kale' <<enter>> invoert, stoppen we de lus alsook de applicatie
      OPGELET: het kan zijn dat er nog een **`'\n'`** in de keyboardbuffer zit na het inlezen van de doubles. Gevolg is dat hij niet wacht om en er daardoor niet gewacht totdat de gebruiker wordt op invoer!
      TIP: Je kan dit oplossen door de methode skip("\n") uit te voeren op de keyboardbuffer
- Toon het resultaat op het scherm. Alle kommagetallen worden getoond met een precisie van 2 cijfers na de komma.
- Verwachte uitvoer:

```
Welkom bij de dynamische rekenmachine!
Geïnstalleerde plugins: + - * / ^

Welke berekening wenst U uit te voeren (<ENTER> om te stoppen)? +
Geef twee getallen in (gescheiden door een spatie): 8 12
8,00 + 12,00 = 20,00

Welke berekening wenst U uit te voeren (<ENTER> om te stoppen)? *
Geef twee getallen in (gescheiden door een spatie): 15 97
15,00 * 97,00 = 1455,00

Welke berekening wenst U uit te voeren (<ENTER> om te stoppen)? ?
Geef twee getallen in (gescheiden door een spatie): 20 20
Plugin ? niet geïnstalleerd.
20,00 ? 20,00 = 0
```

3.3. Voeg via anonieme inner klassen in **`TestRekenmachine`** nog twee extra Plugins toe. Eentje om het minimum van twee getallen te bepalen (commando =”MIN”) en eentje om het maximum van twee doubles te bepalen (commando = “MAX”).
Voeg deze twee anonieme innerklassen toe als extra plugin aan de rekenmachine. Aan alle andere code – dus ook aan de lus gecreëerd in opgave 3.2 – zou je niets mogen aanpassen om volgende uitvoer te krijgen:

```
Welkom bij de dynamische rekenmachine!
Geïnstalleerde plugins: + - * / ^ MIN MAX

Welke berekening wenst U uit te voeren (<ENTER> om te stoppen)? MIN
Geef twee getallen in (gescheiden door een spatie): 10 20
10,00 MIN 20,00 = 10,00

Welke berekening wenst U uit te voeren (<ENTER> om te stoppen)? MAX
Geef twee getallen in (gescheiden door een spatie): -5 5
-5,00 MAX 5,00 = 5,00
```

Uitbreiding :

4. In de klasse **`Rekenmachine`** gaan we zorgen voor een log die de historiek van alle berekeningen bijhoudt.
   4.1. Voorzie een extra attribuut: **`log`** van het type **`StringBuilder`**.
   4.2. Zorg ervoor dat bij elke berekening de log aangevuld wordt met een regel waarin de volledige bewerking staat, inclusief de datum met uur (let op het formaat) van uitvoering
   Bijv: **`[15-nov-2017 23:15:05] 10,000000 + 20,000000 = 30,000000`**
   4.3. Maak een methode **`getLog`** die de log terug geeft als **`String`** met daarvoor een header "==== LOG ====" (zie uitvoer)
   4.4. Toon de inhoud van de log op het einde van je **`main`** methode

Verwachte uitvoer:

```
Welkom bij de dynamische rekenmachine!
Geïnstalleerde plugins: + - * / ^ MIN MAX

Welke berekening wenst U uit te voeren (<ENTER> om te stoppen)? +
Geef twee getallen in (gescheiden door een spatie): 15 32
15,00 + 32,00 = 47,00

Welke berekening wenst U uit te voeren (<ENTER> om te stoppen)? MIN
Geef twee getallen in (gescheiden door een spatie): 10 30
10,00 MIN 30,00 = 10,00

Welke berekening wenst U uit te voeren (<ENTER> om te stoppen)?
==== LOG ====
[15-nov-2017 23:15:05] 15,000000 + 32,000000 = 47,000000
[15-nov-2017 23:15:16] 10,000000 MIN 30,000000 = 10,000000
```

Succes!