Tentamen Objectgeorienteerd Programmeren IN1205
15 januari 2007 9.00-12.00
Afdeling ST Faculteit EWI TU Delft
Uitwerking

Opgave 1.
Als we een object-diagram tekenen, en daarin de gegeven code uitvoeren, vinden we de volgende antwoorden:

a. false -- het zijn verschillende objecten
b. true -- door de laatste assignment is dit hetzelfde object
c. false -- ze hebben een verschillende code
d. false -- ze hebben beide inhoud 12.6
e. false -- "b34" is ongelijk aan "a12"
f. true -- vat1 en vat3 verwijzen naar hetzelfde object

Opgave 2

a.
```
public double totaleInhoud(){
   double totaal = 0.0;
   for (int k = 0; k < aantal; k++)
      totaal += vaten[k].getInhoud();
   return totaal;
}
```

b.
```
public int indexGrootsteRelVulling(){
    int champIndex = -1;
    double champ = 0;
    for (int k = 0; k < aantal; k++){
       Vat vat = vaten[k];
       double volume = vat.getVolume();
       double inhoud = vat.getInhoud();
       double relatief = inhoud / volume;
       if (relatief >= champ){
          champIndex = k;
          champ = relatief;
       }
   }
    return champIndex;
}
```

Eleganter zou hier zijn om in de klasse vat een extra methode toe te voegen:

```
public double relatieveVulling(){
   return inhoud / volume * 100;
}
```

Dan wordt de for-loop in bovenstaande code:

```
for (int k = 0; k < aantal; k++){
   double relatief = vaten[k].relatieveVulling();
   if (relatief >= champ){
      champIndex = k;
      champ = relatief;
   }
}
```

c.
```
public int indexEersteLegeVat(){
   double totaal = totaleInhoud();
   int k = 0;
   while (totaal > 0){
      totaal -= vaten[k].getVolume();
      k++;
   }
   return k;
 }
```

Opgave 3

a.
```
public void schrijfNaar(String bestand){
   try{
        FileWriter fw = new FileWriter(bestand);
        PrintWriter pw = new PrintWriter(fw);
        pw.println(aantal);
        for(int k = 0; k < aantal; k++)
           pw.println(vaten[k]);
        pw.close();
    }
    catch (IOException iox){
       System.out.println(iox);
    }
}
```

We hebben nu nog een toString-methode in Vat nodig:

```
public String toString(){
   return code + " " + volume + " " + inhoud;
}
```

b.
```
public static Vaten leesVan(String bestand){
   Vaten vaten = null;
   try{
      FileReader fr = new FileReader(bestand);
      BufferedReader br = new BufferedReader(fr);
      String regel = br.readLine();
      int aantal = Integer.parseInt(regel);
      vaten = new Vaten(aantal);
      for(int k = 0; k < aantal; k++){
         regel = br.readLine();
         Vat vat = new Vat(regel);
         vaten.voegToe(vat);
      }
      br.close();
   }
   catch (IOException iox){
      System.out.println(iox);
   }
   return vaten;
}
```

De klasse Vat moet nu nog een constructor krijgen waarin een Vat uit een regel(String) wordt geconstueerd:

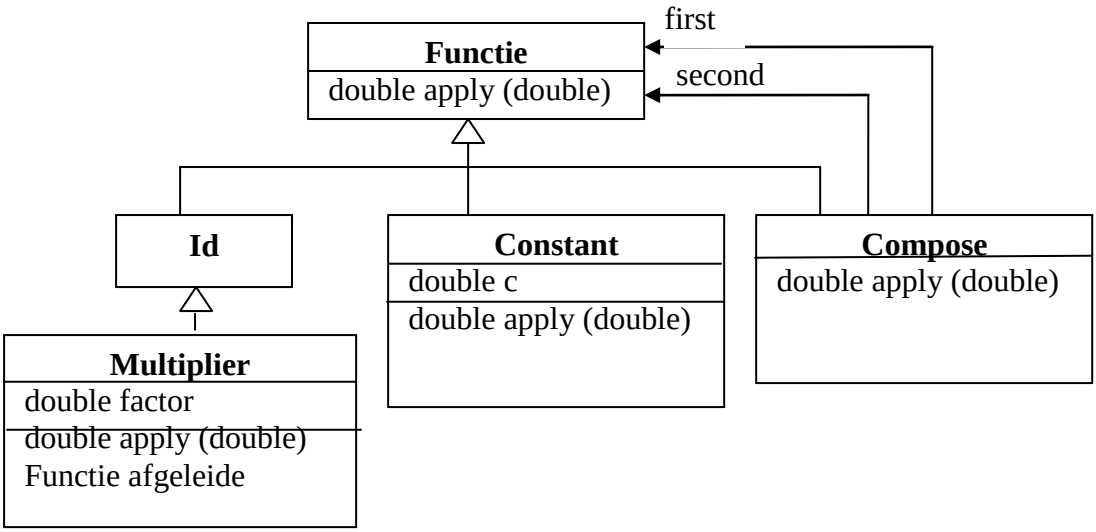
```
public Vat(String regel){
   StringTokenizer st = new StringTokenizer(regel);
   code = st.nextToken();
   volume = Double.parseDouble(st.nextToken());
   inhoud = Double.parseDouble(st.nextToken());
}
```

Met een Scanner gaat dat nog iets eenvoudiger:

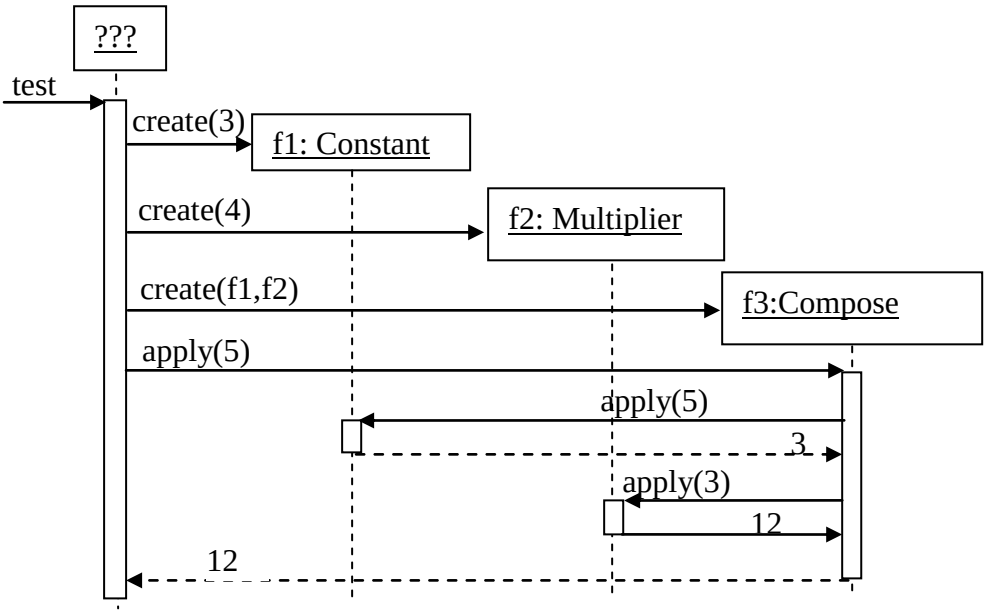
```
public Vat(String regel){
    Scanner sc = new Scanner(regel);
    code   = sc.next();
    volume = sc.nextDouble();
    inhoud = sc.nextDouble();
}
```

Opgave 4

a.



b.



Opgave 5

a. Foutmelding van de vertaler: een Id (statisch type van multiplier) kent geen methode afgeleide.
b. Geen fouten tijdens vertalen. Tijdens het runnen een ClassCastException. Een Multiplier kan niet naar een Constant worden gecast.
c Foutmelding van de vertaler: een Id (statisch type van multiplier) kan niet naar een Multiplier worden gecast.
d. Geen fouten. x = 8.0
e. Geen fouten. x = 6.0
f. Geen fouten. x = 2.0

Opgave 6

1. De rij wordt in twee delen verdeeld: het verwerkte gedeelte, dat geteld is, en het niet verwerkte gedeelte.
2. int volgende;
   de index van de eerste die nog niet geteld is
   int verschillenden;
   het aantal verschillende elementen in rij[0] .. rij [volgende -1]
3. De variabele verschillende bevat het aantal verschillende elementen in rij[0] .. rij [volgende -1]
4. stoppen als volgende == aantal (dan is alles geteld)
5. volgende = 0;
   verschillenden = 0;
6.
```
while (volgende < aantal){
    if (volgende == 0 || rij[volgende]!= rij [volgende-1])
        verschillenden++;
    volgende++;
}
return verschillenden;
```
7. Het algoritme is eindig, omdat aantal – volgende bij elke stap afneemt.
   Bij elke stap blijft de invariant behouden: verschillende wordt opgehoogd als we een nieuwe hebben gevonden. Aan het eind is volgende = aantal, dus volgens de invariant is verschillende het aantal verschillende elementen in rij[0] .. rij [aantal -1]. Dat is precies wat we zochten.

De expressie `rij[volgende]!= rij [volgende-1]` zal echter meteen aan het begin tot een foutmelding leiden, omdat volgende de waarde 0 heeft, dus volgende -1 de waarde -1. We moeten dus bij volgende = 1 beginnen. Maar dat betekent dat we het geval aantal = 0 apart moeten behandelen. Verbeterde versie, (met for-loop):

```
int verschillenden = 1;
if (aantal == 0)
   verschillenden = 0;
for (int volgende = 1; volgende < aantal; volgende++){
    if (rij[volgende] != rij [volgende-1])
        verschillenden++;
}
return verschillenden;
```