Java Opdrachten week 2

Opdracht 1 - Subclasses en Superclasses

Stel: je maakt een computersysteem voor een dierentuin. Je moet voor elk dier het volgende bijhouden:

- Dier is gevoerd.
- Dier is bij de dierenarts geweest.
- Naam en leeftijd

a) Maak een class Animal die de methods *feed()* en *visitVet()* heeft. De precieze uitwerking van de methods is nu nog niet belangrijk - als ze een berichtje in de console printen is dat al goed.
b) Geef je Animal class 2 variabelen - naam en leeftijd.
c) Maak ook een constructor die de naam en leeftijd van het dier als parameter neemt.
d) Maak tenslotte een static variable “type” en geef deze de waarde “Animal”. Waarom dit belangrijk is volgt in een latere vraag.

Vervolgens merk je: dit is niet het enige wat je voor je dieren moet bijhouden. Er zijn in jouw dierentuin ook een stel beren die naar een speciale beren-arts moeten. Daarvoor ga je een ***Subclass*** gebruiken. Syntax:

```
public class Bear extends Animal
```

Hiermee geef je dus aan dat je class Bear een *subclass* is van je class Animal. Andersom heet dit een ***superclass*** - Animal is hier dus een *superclass* van Bear. Dit is vergelijkbaar met de parent-child naamgeving in HTML.

Net zoals bij HTML de child elementen bepaalde properties overnemen van de parent, nemen de subclasses bepaalde properties over van hun superclass. Hoe dit precies in zijn werk gaat gaan we in de volgende opdracht verder op in.

Opdracht 2 - Ik zag 2 beren instantiëren...

We moeten de code van de Bear class eerst wat uitbreiden:

a) Geef je Bear class een instance variabele “type”.

Nu heeft je Bear class ook een constructor nodig. Je kan dit net zo uitschrijven als je bij de vorige Animal constructor hebt gedaan. Je kan ook ***super()*** gebruiken. Super verwijst naar de superclass - super() verwijst dan ook altijd naar de constructor van de superclass.

b) Maak een constructor voor je class Bear. Deze moet de variabelen name, age en type als parameter gebruiken.

Vervolgens gaan we onze beren instantiëren. Dat doen we in de main method van de class Bear.

c) Maak 2 Bear Objects aan. De naam en leeftijd mag je zelf kiezen - zorg er wel voor dat de ene de type “Brown Bear” heeft en de andere de type “Polar Bear”.

Zoals je waarschijnlijk al hebt opgemerkt kan je dus een variabele die in je superclass zit ook gebruiken in de subclass. **Toch hebben je beren de informatie van beide variabelen “type”!** Dit gaan we nu laten zien.

d) Maak in je Bear class een method showType(). Print in deze method de type van je class Bear EN die van de superclass Animal. Roep deze vervolgens in je main method voor beide beren aan. *Tip: Gebruik het keyword super. Let ook op je access modifiers!*

Je Bear Objects zijn dus niet alleen een Bear met een property “type”, maar tegelijkertijd ook een Animal met een property “type”. Dat een Object tegelijkertijd als meerdere soorten Object behandeld kan worden kan je ook testen:

e) Maak in de main method van je Bear class naast de 2 beren nu ook een Animal object aan. Zet vervolgens alle beren in een Bear[] array EN alle drie je dieren in een Animal[] array.

Het concept dat Objecten tegelijkertijd als meerdere verschillende soorten Object behandeld kunnen worden wordt in Java ***Polymorphism*** genoemd.

Opdracht 3 - Inheritance, Overriding & Overloading

De methods *feed()* en *visitVet()* van Animal zijn te gebruiken door je beren, ook al heeft je Bear class die nog zelf niet.

a) Maak in je main een for-each loop die door je Animal array gaat en bij elk Animal de method *feed()* aanroept.
b) Doe nu hetzelfde voor je Bear array

Dit is een voorbeeld van ***Inheritance***.
Je kan *inherited methods* ook aanpassen om ze wat specifieker te maken.

c) Maak nu in je Bear class de methods *feed()* en *visitVet()*. Roep bij beiden eerst de method van de superclass aan, en voeg er vervolgens aan toe dat de beer de dierenarts heeft opgegeten.

Dit heet ***Method Overriding***.
Als je niet wil dat een method door je subclass veranderd wordt, kan je de method **final** maken. Je kan ook de hele class final maken, dan kan deze niet als superclass worden gebruikt.

Stel dat je nu een method wil maken die doorgeeft dat je beer iets speciaals heeft gegeten. Wat voor speciaals wil je als parameter aan je method doorgeven. Dit kan je doen door een nieuwe method aan te maken, maar dit kan ook door de bestaande method *feed()* te **overloaden**.

Method Overloading houdt in dat je dezelfde naam voor verschillende methods gebruikt, met als verschil welke/hoeveel parameters worden gebruikt. Als de method wordt opgeroepen, kijkt je programma naar de parameters en selecteert aan de hand daarvan welke hij moet gebruiken.

d) Maak een 2e *feed()* method die een String als parameter neemt en dat in de output weergeeft.

Waar dit vooral veel bij wordt toegepast is bij constructors.

e) Overload je *Bear()* constructor met een extra constructor die naast name+age ook weight als parameter gebruikt.

Bonus Opdracht - Dierentuin

Nu gaan we een realistisch dierentuin-systeem nabouwen. Dit systeem houdt het volgende bij:

- Een lijst met alle dieren in de dierentuin
- De naam van elk dier
- De leeftijd van elk dier
- De geboortedag van elk dier
- De dag dat een dier voor het laatst gevoerd is.
- Per dag een nieuwe lijst met dieren die gevoerd moeten worden.
    - Als het de verjaardag van een dier is, ook een bericht dat dit meldt.

Gebruik de volgende classes:

- Zoo - je Zoo class heeft de volgende methods
    - *add(...)* - voegt een dier aan de dierentuin toe
    - *remove(...)* - haalt een dier uit de dierentuin weg
    - *update(...)* - Deze geeft terug:
        - Welke dieren vandaag jarig zijn
        - Welke dieren vandaag gevoerd moeten worden
        - Gebruik een Date Object als parameter, zodat je dit ook op bijvoorbeeld de dag van morgen kan testen.
- Animal
- Bird, subclass van Animal
    - Vogels worden 1x per 2 dagen gevoerd.
- Reptile, subclass van Animal
    - Reptielen worden 1x per week gevoerd.
- Cat, subclass van Animal
    - Katten worden dagelijks gevoerd.