Veckoövning V4: Djurparken

I veckans föreläsning har vi gått igenom grunderna inom arv och objektorienterade principer. Vi har tittat på exempel där vi modellerat ett fightingspel med hjälp av *arv* och *overriding*. I den här övningen kommer vi att bygga ett större program, men på ett lite fredligare tema.

Notera att denna övning är fullständigt frivillig, och inte kommer betygsättas.

1. Förberedelser

Gå igenom veckans föreläsning och läsanvisningar. Var extra uppmärksam på delarna som handlar om arv och overriding.

2. Modellera en djurpark med hjälp av arv

Med ledning av exempelkoden från måndagens föreläsning, konstruera några klasser som var och en representerar en viss sorts djur. Du kan skapa så få/många djur du vill, men försök att skapa en arvshierarki som är flera nivåer djup.

Till exempel:

- Alla djur har basklassen `Animal`.
- Gräsätare (`Herbivore`) och köttätare (`Carnivore`) är `Animal`s.
- Hästar och alpackor är båda två `Herbivore`s.
- Ormar (`Snake`) och krokodiler (`Crocodile`) är `Carnivore`s.

Hitta därefter på några egenskaper som är gemensamma för samtliga djur, och lägg till dem som fält (med getters och setters) i klassen `Animal`. Exempel på sådana egenskaper är `int age`, `String species` eller `String name`.

Hitta även på några beteenden som är gemensamma för samtliga djur i din hierarki, och implementera dem i form av metoder i klassen `Animal`.

Ett exempel på en sådan metoder är `String eat(String nameOfFood)`, som "matar" ett djur med den givna maten, och ger tillbaka en sträng som talar om hur djuret reagerade. Ett annat är `String pet()`, som talar om hur djuret reagerar på att klappas.

3. Ge varje djurart egna beteenden med hjälp av overriding

Olika djurarter har olika reaktion på olika händelser. Till exempel finns en viss risk att du själv blir uppäten om du försöker klappa en krokodil medan det är relativt riskfritt att klappa en alpacka, och växtätare reagerar troligtvis mer positivt på att bli matade med äpplen än köttätare.

Ge olika djurarter olika beteenden genom att i dina underklasser overridea de olika metoderna du definierade i klassen `Animal`. Försök återanvända så mycket kod som möjligt; om ett visst beteende är gemensamt för alla `Herbivore`s är det

bättre att overridea metoden en gång i `Herbivore`-klassen än att göra det för varje djur som ärver `Herbivore`.

4. Skriv en main-metod

Att bara modellera en djurpark är lite tråkigt om man inte kan göra någonting där, så skriv en main-metod som låter användaren interagera med djuren i din djurpark. Eftersom alla dina djur ärver klassen `Animal` så kan du modellera en samling djur som:

```
Animal[] animals = { new Lion(...), new Giraffe(...), new Otter(...) };
```

Du kan sedan låta användaren välja vilket djur hen vill interagera med genom dess index. En interaktion med användaren skulle kunna se ut såhär:

```
Vilket djur vill du interagera med? (0 - 2)
> 1
Du valde en giraff.
Vad vill du göra?
1. Mata
2. Klappa
> 1
Vad vill du mata giraffen med?
> äpple
Du ger giraffen äpple.
Hen blir glad.
```

Försök göra detta i en loop, så användaren kan interagera flera gånger med samma (eller olika) djur utan att behöva starta om programmet.

5. Extrauppgifter

Utöka ditt program med funktionalitet som gör det mer intressant för användaren. Några förslag:

1. Djur har en "mättnadsnivå" som ökar när de matas och minskar för alla andra interaktioner. Mätta djur vägrar äta, och för hungriga djur svälter ihjäl.
2. Djur kan vara oberäkneliga. Använd klassen `Random` för att ge potentiellt olika resultat för varje gång en metod anropas. Kanske finns det 10% risk att `Lion.feed(String nameOfFood)` kastar ett `YouGotEatenException`?
3. Konvertera din djurpark till en Pokémonarena! Ge alla djur en metod `boolean fight(Animal opponent)` som returnerar `true` om djuret som metoden anropas på vinner över djuret `opponent`.

   Om du försöker dig på den här uppgiften behöver du sannolikt ge dina djur fler attribut, som t ex `int strength`, `int hitPoints`, etc.