

C# Övningssamling - Inkapsling, arv och polymorfism

Löpande i uppgifterna finns några kunskapsfrågor (startar med "F:"). Dessa frågor besvaras som kommentarer i koden. Koden behöver inte ta emot input från en användare.

3.1) Inkapsling

1. Skapa en klass *Person* och ge den följande privata fält:
age, fName, lName, height, weight

Skapa publika properties med *get* och *set* som hämtar eller sätter tilldelad variabel. Instansiera en person i *Program.cs*, kommer du direkt åt variablerna?

Implementera validering i de skapade properties:

- Age kan bara tilldelas ett värde större än 0.
- FName är obligatorisk och får inte vara mindre än 2 tecken eller längre än 10 tecken.
- LName är obligatorisk och får inte vara mindre än 3 tecken eller större än 15 tecken.

Kasta ett undantag av typen *ArgumentException* i varje property om dess validering inte fullföljs, undantaget ska innehålla ett beskrivande meddelande.

Se till att hantera undantagen i *Program*-klassen med en try-catch block.

2. För att inkapsla *Person*-objekten ytterligare skall vi skapa klassen *PersonHandler* - en klass vars syfte är att skapa och hantera dina *Person*-objekt.
I *PersonHandler*-klassen skapa metoden:

```
public void SetAge(Person pers, int age)
```

Använd den inskickade personens *Age property* för att sätta personens *age-attribut* via *SetAge-metoden*. Istället för att enbart använda en *property* har vi nu abstraherat med två lager.

3. I *PersonHandler*, skriv en metod som skapar en person med angivna värden:

```
public Person CreatePerson(int age, string fname,  
                           string lname, double height, double weight)
```

4. Fortsätt skapa metoder i *PersonHandler* för att kunna hantera samtliga operationer som man kan vilja göra med en *Person*.

5. När denna klass är klar, kommentera bort er tidigare instans av *Person* från *Program.cs*, och instansiera istället en *PersonHandler*. Skapa därigenom några personer och testa era metoder.

3.2) Polymorfism

1. Skapa den abstrakta klassen *UserError*
2. Skapa den abstrakta metoden *UEMessage()* som har returtypen *string*.
3. Skapa en vanlig klass *NumericInputError* som ärver från *UserError*
4. Skriv en *override* för *UEMessage()* så att den returnerar *"You tried to use a numeric input in a text only field. This fired an error!"*
5. Skapa en vanlig klass *TextInputError* som ärver från *UserError*
6. Skriv en *override* för *UEMessage()* så att den returnerar *"You tried to use a text input in a numeric only field. This fired an error!"*
7. I *Program.cs* Main-metod: Skapa en lista med *UserErrors* och populera den med instanser av *NumericInputError* och *TextInputError*.
8. Skriv ut samtliga *UserErrors* *UEMessage()* genom en *foreach* loop.
9. Skapa nu tre egna klasser med tre egna definitioner på *UEMessage()*
10. Testa och se så det fungerar.

3.3) Arv

1. Skapa abstrakta klassen *Animal*
2. Fyll klassen *Animal* med egenskaper (properties) som alla djur bör ha. Tex namn, vikt, ålder.
3. Skapa en abstrakt metod som heter *DoSound()*.
4. Lägg till en konstruktor.
5. Skapa Subklasserna (ärver från *Animal*): *Horse*, *Dog*, *Hedgehog*, *Worm* och *Bird*, *Wolf*.
6. Ge dessa minst en unik egenskap var. Vilken egenskap det är är inte det viktiga här. Exempel *Worm*: *IsPoisonous*, *Hedgehog*: *NrOfSpikes*, *Bird*: *WingSpan* osv.
7. Implementera så att *DoSound()* metoden skriver ut hur djuret låter..
8. Skapa nu följande tre klasser: *Pelican*, *Flamingo* och *Swan*. Dessa ska ärva från *Bird*.
9. Ge dessa minst en unik egenskap var.
10. Skapa gränssnittet (interface) *IPerson* med en metod deklaration *Talk()* ;

11. Skapa klassen `Wolfman` som ärver från `Wolf` och implementerar `IPerson` gränssnittet.
12. Implementera `Talk()` som skriver ut vad `Wolfman` säger.
13. **F:** Om vi under utvecklingen kommer fram till att samtliga fåglar behöver ett nytt attribut, i vilken klass bör vi lägga det?
14. **F:** Om alla djur behöver det nya attributet, vart skulle man lägga det då?

3.4) Mer polymorfism

1. Skapa metoden `Stats()` i klassen `Animal` som har returtypen `string`. Metoden ska kunna *override*s i dess Subklasser. Metoden ska returnera alla egenskaper (*properties*) som djuret har.
2. Skriv en *override* för `Stats()` i subklasserna för `Animal` så den returnerar alla *properties* för det djuret.
3. Skapa en lista `Animals` i `Program.cs` som tar emot djur.
4. Skapa några djur (av olika typ) i din lista.
5. Skriv ut vilka djur som finns i listan med hjälp av en *foreach-loop*
6. Anropa även `Animals Sound()` metod i *foreach-loopen*.
7. Gör en check i *for-loopen* ifall ett djur även är av typen `IPerson`, om den är det *type-casta* till `IPerson` och anropa dess `Talk()` metod.
8. Skapa en lista för hundar.
9. **F:** Försök att lägga till en häst i listan av hundar. Varför fungerar inte det?
10. **F:** Vilken typ måste listan vara för att alla klasser skall kunna lagras tillsammans?
11. Skriv ut samtliga `Animals Stats()` genom en *foreach loop*.
12. Testa och se så det fungerar.
13. **F:** Förklara vad det är som händer.
14. Skriv ut `Stats()` metoden enbart för alla hundar genom en *foreach* på `Animals`.
15. Skapa en ny metod med valfritt namn i klassen `Dog` som endast returnerar en valfri sträng.
16. Kommer du åt den metoden från `Animals` listan?
17. **F:** Varför inte?
18. Hitta ett sätt att skriva ut din nya metod för dog genom en *foreach* på `Animals`.

Lycka till!