C# Övningssamling - Inkapsling, arv och polymorfism

OBS - Resultatet av övningen skall visas för lärare och godkännas innan den kan anses vara genomförd.

Löpande i uppgifterna finns några kunskapsfrågor (startar med "F:"). Dessa frågor besvaras som kommentarer i koden.

3.1) Inkapsling

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

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

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

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

   Använd den inskickade personens *Age property* för att sätta personens *age-attribut*
   inom den här 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 *Person* från Program.cs, och instansiera istället en *PersonHandler*. Skapa därigenom några personer och testa era metoder.

3.2) Arv

1. Skapa klassen *Animal*
2. Fyll klassen *Animal* med egenskaper (*properties)* som alla djur bör ha.
3. Skapa Subklasserna (ärver från *Animal*): *Horse*, *Dog*, *Hedgehog*, *Worm* och *Bird*.
4. Ge dessa minst en unik *egenskap* var.
5. Skapa en lista i program.cs som tar emot djur.
6. Skapa några djur (av olika typ) i din lista.
7. Skriv ut vilka djur som finns i listan med hjälp av en *foreach-loop*
8. Skapa en lista för hundar.
9. Försök att lägga till en häst i listan av hundar. Varför fungerar inte det?
10. Skapa nu följande tre klasser: *Pelican*, *Flamingo* och *Swan*. Dessa ska ärva från *Bird*.
11. Ge dessa minst en unik egenskap var
12. F: Vilken typ måste listan lagra för att dessa tre nya klasser ska kunna lagras tillsammans?
13. F: Vilken typ måste listan vara för att alla klasser skall kunna lagras tillsammans?
14. F: Om vi under utvecklingen kommer fram till att samtliga fåglar behöver ett nytt attribut, i vilken klass bör vi lägga det?
15. F: Om alla djur behöver det nya attributet, vart skulle man lägga det då?

3.3) 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 returerar *"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 returerar *"You tried to use a text input in a numericonly 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.
11. F: Varför är polymorfism viktigt att behärska?
12. F: Hur kan polymorfism förändra och förbättra kod via en bra struktur?

Lycka till!