Välkomna alla SUVx-23-are till ***Boiler room fre 3/11, kl 9-12.***

Idag presenteras ett lite större uppdrag till klassen för dagens Boiler room.

Uppdrag Lönespecifikation

Så här berättar en av företaget *Soft Consulting*:s kunder när chefsäljaren Anders är på kundbesök:

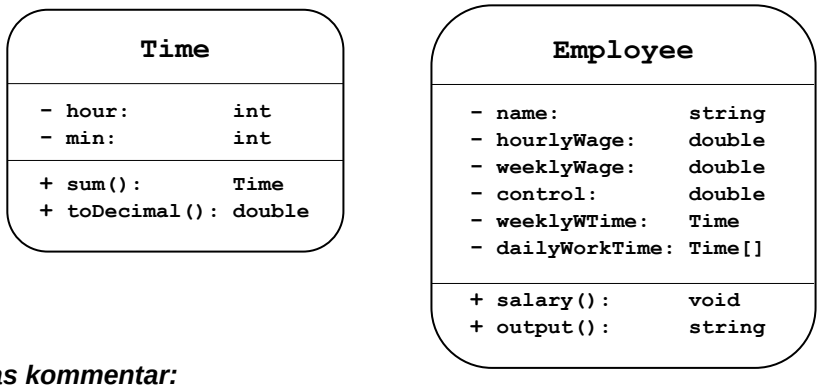
> ”Vi vill datorisera lönespecifikationen till våra timanställda. Varje vecka får vi timrapporter över deras arbetstider i timmar och minuter.
>
> Ett program behövs som läser in en timanställds namn, timlön och arbetstider för varje veckodag. Sedan ska programmet summera arbetstiderna, beräkna veckolönen samt visa både veckans totala lön samt veckans totala arbetstid i timmar och minuter. En oberoende kontrollräkning ska bekräfta resultatet. Lönespecifikationen ska skrivas ut, så att den kan skickas ut till våra medarbetare.”
>
> ***Kundens kravspecifikation***

Vi på *Soft Consulting* döper uppdraget till *projekt Lönespecifikation* och bestämmer oss för att lösa problemet med ett objektorienterat program. Som vanligt delar vi upp projektet i två utvecklingsfaser:

Fas 1: Modellering och design

Fas 2: Implementation

Projektets modelleringsfas skickas till företagets designavdelning. Implementationen tilldelas vårt utvecklingsteam och ska baseras på designkollegornas modellering. När resultatet kommer från designavdelningen ser vi att modelleringen består av följande klassdiagram samt en kommentar:



Designkollegornas kommentar:

Den objektorienterade modellen baseras på två entiteter **`Time`** och **`Employee`** som vid implementeringen blir klasser. Entiteten **`Time`** har attributen **`hour`** och **`min`** som vid implementeringen blir datamedlemmar. Metoderna visas i diagrammen. **`sum()`** ska addera två tider, dvs två **`Time`**-objekt och returnera ett **`Time`**-objekt, medan **`ToDecimal()`** ska omvandla tiden till decimaltal för att kontrollera summeringen med **`sum()`**, se kundens krav på oberoende kontroll.

I entiteten **`Employee`** har vi en anställds *namn*, *timlön (hourlyWage)* och *dagarbetstider (dailyWorkTime)* som attribut. Kundens krav på kontrollräkning och att ”beräkna veckolönen samt visa både veckans totala arbetstid i timmar och minuter” leder till att inkludera *veckolön (weeklyWage)*, *kontroll* och *veckoarbetstid (weeklyWTime)* som attribut till entiteten **`Employee`**. Metoden **`salary()`** ska beräkna veckolönen samt kontrollen, medan **`output()`** ska skriva ut klassens data som sträng.

Observera att **`Time`** är returtypen till metoden **`sum()`** i klassen **`Time`**, därför att summan av två tider också är en tid.

Dessutom är arrayen **`Time[]`** datatypen till datamedlemmen **`dailyWorkTime`** i klassen **`Employee`**, därför att det finns 5 arbetsdagar i veckan, som kan modelleras som en array av 5 **`Time`**-objekt. **`Time[]`** är en sådan array.

Modelleringen har genomförts enligt standarden UML *(Unified Modeling Language)*. □

Så långt designavdelningens modell samt kommentar.

Vårt uppdrag:

Soft Consultings utvecklingsteam ska implementera designavdelningens modell i ett C++ program och i stort sett följa deras modellering.

Stöter man av programmeringstekniska skäl på anledningar att avvika från modellen, har man möjligheten att göra det. I så fall borde avvikelsen motiveras.

För att underlätta utvecklingsarbetet kan man till att börja med deklarera alla klassers medlemmar som **`public`**, för att sedan gå över till att ha datamedlemmarna som **`private`** och metoderna som **`public`**, så som designavdelningens modell föreskriver.

Frivilligt!

När programmet fungerar fundera i mån av tid på följande vidareutveckling av uppdraget:

Förse lönespecifikationen med *aktuellt datum* och *veckonummer*. Ta själv reda på informationen om hur man i C++ kan inkludera datorns tid i sitt program. Om veckonumret inte explicit finns i datortidens fördefinierade komponenter, fundera hur man kan beräkna veckonumret utgående från de befintliga komponenterna.

Anmärkning

Av förekommen anledning skulle jag vilja påminna att Boiler room uppdragen inte är betygsgrundande. De ska vara inspirerande, bidra till diskussion, kommunikation och utbyte av idéer. På så sätt ska de neutralisera distansundervisningens brist på social kontakt.

Förefaller de ibland vara svårare än kursens ordinarie övningar och projekt, beror det på konceptet att använda verklighetsbaserade problem.

Hinner man inte med ett uppdrag under den planerade tiden, kan man fortsätta med det till nästa Boiler room. Nästa termin ska vi ha uppdrag som går över flera Boiler rooms.

Lycka till!

Hälsningar

Taifun