

Inlämning: Left To Do, 5 Yhp

Inlämningsdatum 3 dec 2021 av 23.59 Poäng 100

Lämnar in en länk till webbplats

Tillgänglig 22 nov 2021 kl 0:00–3 dec 2021 kl 23.59 12 dagar

Den här uppgiften låstes 3 dec 2021 kl 23.59.



Scenario

Du ska i denna uppgift skapa en konsolapplikation för att hålla koll på, och ordna arbetsuppgifter; det vill säga en **digital att-göra lista**. Applikationen ska ha namnet *Left To Do*. En användare ska vid användning av programmet kunna se vad som är kvar att göra i listan, markera uppgifter som avklarade, lägga till nya uppgifter i listan samt arkivera avklarade uppgifter.

I denna uppgift ställs högre krav på att du har själv väljer en lämplig kodstruktur, och uppgiften blir mycket enklare att utveckla om du tillämpar det du lärt dig om de objektorienterade principerna. Dessutom måste du i denna uppgift utveckla din kod med stöd av enhetstester.

Applikationen måste innehålla enhetstester för att lägga till en uppgift i listan, markera en uppgift som avklarad samt arkivera alla avklarade uppgifter i listan. För VG på uppgiften krävs det att det ska finnas två ytterligare typer av "att-göra"-uppgifter, två nya klasser ska i så fall skapas i programmets källkod, som förslagsvis ärver från en gemensam superklass.

Inlämning & betygsättning

Du ska utveckla och strukturera en konsolapplikation som uppföljer kravlistan för uppgiften. Genom att uppfylla kraven kommer du påvisa din förmåga att konstruera program på ett objektorienterat sätt (läranderesultat 4) samt strukturera din kod för god läsbarhet och att själv kunna analysera och förstå dess funktion (läranderesultat 5). Dessutom kommer du i denna uppgift att skriva enklare enhetstester för att garantera att programmet fungerar över tid (läranderesultat 6).

Läs igenom nedanstående kravlista/poängtabell och några korta tips på uppgiften nedanför, starta sedan arbetet genom att acceptera uppgiften på Github Classroom:

[Github Classroom: Left To Do](https://classroom.github.com/a/1Ls1nyei) [_\(https://classroom.github.com/a/1Ls1nyei\)](https://classroom.github.com/a/1Ls1nyei)

När du är klar med uppgiften lämnar du in din lösning som en länk till det repo som skapats på Canvas. Efter rättning kommer du få feedback med korta kommentarer för varje rad i poängtabellen. För att ett krav ska vara poänggivande måste även de krav som angetts som *kriterium* vara uppfyllda! För betyget godkänt krävs minst **50 poäng** från poängtabellen, för betyget väl godkänt krävs **80 poäng**, samt att minst 20 av dessa är från de krav som är angivna med (vg) i poängtabellen. Erhålls åtminstone 25 poäng så erbjuds ett försök att komplettera din inlämning, annars ges betyget *underkänt*.


Självskattning och deadline

Läs nedanstående kravlista noggrant, när du påbörjar programmeringsuppgiften från Github Classroom så kommer du se att det finns en readme.md fil där du ska fylla i din självskattning. Självskattningen är frivillig men du bör alltid fylla i denna så att du vet att din uppgift rättas korrekt - läs [Om inlämning och deadlines](#) för mer information.

Kravlista

Följande kravlista är vad du ska programmera och lämna in, den fungerar som **poängtabell** för uppgiften:

Nummer Max.Poäng Kriterium Beskrivning			Läranderesultat
1	2p	Lösningen ska bestå av en konsolapplikation skriven i C# som är körbar med .NET Core	4, 5

Nummer	Max.Poäng	Kriterium	Beskrivning	Läranderesultat
2	2p	1	När man startar programmet ska man mötas av en startmeny som förklarar hur programmet används och låter användaren genomföra resterande krav.	4, 5
3	8p	2	Man ska som användare kunna <u>lista alla dagens uppgifter</u> , alla uppgifter ska då listas om dom inte är "arkiverade".	4, 5
4	8p	3	När dagens uppgifter är listade ska de kunna markeras som utförda , eller som kvar att göra beroende på sin nuvarande status. <i>Detta kan förslagsvis köra med en symbol som X eller en emoji som ✓ eller </i>	4, 5
5	8p	2	Från programmets menyer ska man även kunna <u>lägga till en ny uppgift</u> .	4, 5
6	8p	2	Från programmets menyer ska man även kunna <u>arkivera alla uppgifter som för närvarande är utförda</u> .	4, 5
7	4p	2	Man ska kunna <u>lista alla arkiverade uppgifter</u> , det behöver inte gå att göra något med uppgifterna när de listas på detta vis.	4, 5
8	4p	1	Den inlämnade lösningen ska bestå av en "Program.cs" fil, en ".csproj" fil samt en fil för varje extra klass du skapar.	5
9	2p	1	Ett enhetstest finns i lösningen för att testa att uppgifter går att lägga till i listan.	6
10	2p	1	Ett enhetstest finns i lösningen för att markera en uppgift som avklarad.	6
11	2p	1	Ett enhetstest finns i lösningen för att arkivera alla avklarade uppgifter i listan.	6
12	4p	5	Man ska kunna lägga till en särskild typ av uppgift med en "deadline", deadline ska sparas i systemet som antalet dagar kvar att slutföra uppgiften. Denna typ av uppgift ska vara en egen klass.	4, 5

Nummer	Max.Poäng	Kriterium	Beskrivning	Läranderesultat
13	4p	5	Man ska kunna lägga till en typ av uppgift som har en inbyggd checklista med ytterligare detaljer - för att markeras som avklarad så måste alla punkterna i checklistan också markeras som avklarade. Denna typ av uppgift ska vara en egen klass.	4, 5
14	2p	12/13	Lösningen ska innehålla ytterligare 1 eller 2 enhetstester som är lämpliga för de nya typerna av uppgifterna.	6
15 (vg)	2p		Lösningen ska förutom kod innehålla en fil med namnet "reflections" i formatet <i>md</i> , <i>txt</i> eller <i>pdf</i>	4, 5, 6
16 (vg)	8p	15	<i>reflections</i> -filen ska innehålla en kort beskrivning av kodens funktioner och struktur så som du förstår den. Ungefär 2-3 paragrafer.	5
17 (vg)	15p	12, 13 & 15	Filen <i>reflections</i> ska innehålla rubriken "använda principer" där du förklarat hur lösningen använder de objektorienterade principerna <i>arv</i> , <i>enkapsulation</i> och <i>polymorfism</i> för de ytterligare uppgifterna i krav 12 och 13	4
18 (vg)	15p	14 & 15	Filen <i>reflections</i> ska innehålla rubriken "tester för klasser" där du förklarat hur de nya testerna som du skapat för kravet 14 fungerar och varför de är väl valda enhetstester.	6

Tips

Testdriven utveckling är en mycket lämplig arbetsmetod att följa för denna uppgift. Börja med att utveckla de enhetstester som krävs i kravlistan och utveckla din applikation med *objektorienterad* struktur - tänk på att separera koden på ett sådant sätt att klasserna för "uppgifterna" inte är beroende av *System.Console*-klassen.

Notera att ingen information om uppgifterna måste sparas mellan att programmet startar och stängs av. Stängs programmet av så försvinner helt enkelt alla skapade uppgifter.

Det krävs egentligen inte i denna uppgift att du använder någon mer avancerad logik än i Multifabriken - men sammanhanget (relationerna) mellan dina klasser blir mycket mer avancerat om du siktar mot VG - tänk då noga igenom hur interpretatorn .NET kommer att gå igenom din

kodstruktur och hur du kan undvika att upprepa kod mellan klasser.

För att skapa ett bra UI behöver du på något vis låta användaren välja från en lista. Detta görs enklast genom att användaren får mata in ett nummer - som du sedan använder som ett index till listan. Andra samlingsklasser kan också vara användbara! Om du vill spänna bågen på denna uppgift så fundera på om du det finns ett visuellt snyggare sätt att visa för användaren vilken uppgift som är markerad? Oavsett hur du skriver användargränssnittet är det viktigt att du skyddar din applikation från felaktig inmatning.

Så här kan *exempelvis* programmet se ut och fungera när det är klart:

```
LEFT TO DO
Gör dina menyval genom att trycka på siffra och Enter.
Välj vad du vill göra:
[1] Lägga till en ny uppgift
[2] Lista alla dagens uppgifter
[3] Lista alla arkiverade uppgifter
[4] Arkivera alla avklarade uppgifter
[Q] Avsluta programmet
```

