Smells

Clean Code laboration

Laborationen utförs individuellt eller i par, dessa par måste bestämmas innan ni påbörjar uppgiften genom att skriva till mig.

Inlämning sker via ZIP fil på ITHSdistans som innehåller programkoden och en PDF med er skriftliga redovisning. Ni bör även använda Git och Github medan ni arbetar och ni får gärna komplettera med en länk till ert Githubprojekt.

Inlämningen ska innehålla:

- Er kod i form av ett C#-projekt
- En skriftlig redovisning där ni kortfattat beskriver de refaktoriseringar och andra förbättringar som har utförts. Minst 1/2 A4 ska skrivas.

Laborationen ska vara inlämnad senast den: 08.09.2023 kl 23:59.

Programmet

Koden som skall refaktoriseras och testas är medvetet fullt av code smells men fungerar vid normal användning. Det implementerar spelet ”Moo” som är ett Mastermind-liknande spel där programmet slumpar fram en fyrsiffrig sträng som spelaren skall gissa, med så få gissningar som möjligt. Den slumpade strängen har garanterat fyra unika siffror.

Spelaren gissar med fyra siffror. Som svar visar spelet ett antal B:n och C:n (bulls and cows). Ett B för varje siffra som har satts på rätt plats och ett C för varje siffra som finns med bland de fyra men är på fel plats. ”B,” betyder alltså att en av fyra siffror i gissningen är rätt och är på rätt plats (men man får inte reda på vilken som är rätt). ”,CCC” betyder att tre gissade siffror finns med, men ingen av dem är på rätt plats. ”BBBB,” är sålunda målet och då är omgången slut.

Spela några gånger med en ”fuskutskrift” av talet så är det lätt att lära sig reglerna. Spelaren får göra gissningar med icke-unika siffror om hen vill, men målet har alltid fyra unika. Efter varje spel skrivs alla spelares genomsnittliga antal gissningar ut i en topplista, ordnad efter bästa gissningsgenomsnitt.

Spelet för statistik över utförda spel i en fil, från vilken topplistan beräknas och skrivs ut efter varje spel.

Er uppgift

Uppgiften är att refaktorisera programmet genom att identifiera code smells och minimera/eliminera dem samt att göra koddelarna testbara för unit-testing och skriva tester som testar programlogiken.

Refaktorisering

Programmet skall göras ”vackrare” genom utförande av refaktorisering på lämpliga delar.
Kursbokens ideer om

- Namngivning
- Funktioner/metoder
- Kommentarer
- Klasser
- Felhantering
- Clean tests

skall beaktas och genomföras. Om något designmöster är lämpligt att använda, gör så.

Programmets olika skikt som sköter

- Användargränssnitt
- Programlogik
- Statistikinsamling/redovisning

skall separeras och delarna skall inte känna till annat om de andra delarna än ett interface och **skall** använda Dependency Injection för sammansättning av delarna. Konkret objektskapande och sammansättning av delarna kan skötas med manuell kod i mainmetoden/klassen.

Testning

Tester **skall** skrivas för programlogikkoden. Eftersom den anropar IO och användargränssnittkod, behöver dessa kanske Mock-Object-implementeras för att kunna utföra testerna. Kanske behöver också metoden som genererar slumpsiffrorna mockas, så att tester på programlogiken kan utföras med kända ”slump”-siffror.

Arbetets omfattning

En uppgift av detta slag har naturligtvis inget enda ”rätt svar”.

Koden **skall** efter utförd refaktorisering sakna tydliga code smells.

Koden **skall** vara uppdelad i helt skilda skikt för användargränssnitt, spel och IO/integration.

Spelet bör troligtvis uppdelas i spel-controller och spellogik, så att logiken kan testas separat.

Testerna **skall** testa väsentliga delar av programlogiken, men behöver inte vara alltäckande, det viktigaste är att visa att tester går att genomföra och att några representativa tester finns på plats.

De viktiga steg som utförts skall dokumenteras på ca en halv A4-sida. Detaljer framgår tydligt (eller hur?) av koden så beskrivningen behöver bara vara mer av en översikt. Om du hittar svagheter, buggar, utelämnad felhantering eller säkerhet, etc. får du gärna påpeka eller rätta, men det ingår inte i laborationen att göra det. (Programmet är säkert inte alls perfekt i dessa avseenden.) Samma sak gäller för fil-IO-koden, det ingår inte i uppgiften att använda någon annan teknik, men själva koden kan och skall göras vackrare. Programmet skall vid normal och korrekt användning fungera precis som tidigare, dvs funktionsförbättringar/förändringar ingår inte i uppgiften. Det är dock OK att onormala operationer och feltillstånd ändras.

Frågor

Det är helt OK att ställa frågor om arbetets omfattning och krav, om något i denna beskrivning skulle vara oklart, eller om koden är obegriplig. Det är också helt OK att diskutera och få handledning av arbetet under kurstid!

Uppgift för VG

För att få betyg VG skall allt ovanstående vara utfört plus en extrauppgift:

Strukturera/utöka programmet så att man kan välja att spela andra liknande spel.

T ex MasterMind, där de fyra siffrorna bara kan ha sex olika värden (sex färger i originalet) och dessutom kan förekomma fler än en gång i slumpgruppen. För varje spel krävs en ny resultattabell i databasen, men spelartabellen kan delas mellan spelen. Implementera minst en extra spelvariant och beskriv vad som behöver göras för att lägga till ytterligare spel. Tester för det nya spelet behöver inte implementeras. Eftersom det handlar om ett utbytbart varierande del-beteende luktar det Strategymönstret, men uppgiften kan kanske lösas snyggt även på annat sätt. Resultatet är ett program där man med minimala ändringar kan lägga till kod för nya spel av ”Moo-karaktär”.

Du kan också välja att utöka programmet genom att lägga till ett annat typ av spel som går att komma åt i en meny där du väljer vilket spel du vill spela. Detta spel ska i så fall även det skapas med väl valda design patterns och följa clean-code standarder som ska beskrivas skriftligt.

Redovisning av VG
Denna sker på samma sätt som för betyg G:

- Inskickad kod
- Halv A4-sida som beskriver hur fler-spels-dynamiken åstadkommits eller som beskriver designvalen du har gjort i ditt egna spel.