

[illegible]

## Examinationsuppgiften

Uppgiften för examinationen 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 designmönster är lämpligt att använda, gör så.

Programmets olika skikt som sköter

- Användargränssnitt
- Programlogik
- Databaskod

**skall** separeras och delarna skall inte känna till annat om de andra delarna än ett java-interface och **skall** använda Dependency Injection för sammansättning av delarna. Konkret objektskapande och sammansättning av delarna behöver inte utföras med något DI-ramverk, utan kan skötas med manuell kod i main-metoden/klassen.

Klassen SimpleWindow kan anses som en "färdig" biblioteksklass och behöver inte modifieras eller refaktoriseras alls eftersom vi inte gått igenom Swing mer än översiktligt. Göm den bakom ett java-interface.

## Testning

Tester **skall** skrivas för programlogikkoden. Eftersom den anropar databaskod och användargränssnittkod, behöver dessa mycket troligtvis 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 DB/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 av java-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 examinationsuppgiften att göra det. (Programmet är säkert inte alls perfekt i dessa avseenden.) Samma sak gäller för JDBC-koden, det ingår inte i uppgiften att använda någon annan teknik, men själva koden kan och skall göras vackrare.

### **Frågor**

Det är helt OK att ställa frågor om arbetets omfattning och krav, om något i denna beskrivning skulle vara oklart, eller min kod 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 utbytbar varierande del-beteende luktar det Strategy-mönstret, 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".

### **Redovisning av VG**

Denna sker på samma sätt som för betyg G:

- Inskickad kod i zip-fil
- Muntlig redovisning med körning
- Halv A4-sida som beskriver hur fler-spels-dynamiken åstadkommit.