Examination Javaverktyg och utvecklingsmiljöer

Denna examination fokuserar helt på testning och mockning.

Examinationen görs individuellt och redovisas muntligt med demo och diskussion av lösningen, samt inskickning av samtliga Java-filer till ulf@bilting.se

Du skall skriva ett så heltäckande testpaket för klassen `GuessingGame` som är rimligt/möjligt.

Utgångspunkten är en (nästan korrekt) fungerande applikation med GUI, controller och spellogik, med noggrann uppdelning an ansvarsområden.

Användargränssnitt
Interfacet `UI` beskriver ett mycket enkelt strängbaserat användargränssnitt. En GUI-implementering i Swing finns i `SimpleWindow`. Denna implementering är färdig och behöver inte testas eller vidröras alls.

Business/Domän
Spellogiken definieras helt i klassen `GuessingGame`. Interaktionen mellan `UI` och spelet sköts av `GuessingGameController`. Resultatet av en gissning rapporteras med en enum-typ `GuessResult`.

Integration/Resurs/Data
Applikationen spar inga resultat så här är det tomt.

Applikationsstart och konfigurering
De olika komponenterna sätts samman med Dependency Injection som utförs manuellt i klassen `Main` som också startar applikationen. Några konfigurationskonstanter finns i klassen `Config`.

Kravspec GuessingGame-klassen
Klassen skall erbjuda logiken för ett spel där man skall gissa ett slumpmässigt valt heltal i omfånget 1 till `Config.RANGE` (inklusive, standardvärde 100). Varje gissning skall ge feedback om talet är för stort eller litet. Är gissningens avstånd från slumptalet mindre eller lika med `Config.CLOSE_LIMIT` (standardvärde 10) skall man dessutom få reda på att man är nära. GuessResult har alltså fem olika möjligheter: `correct, tooSmall, tooLarge, tooSmallButClose` resp. `tooLargeButClose`.
Klassen håller reda på hur många gissningar som gjorts. Ett nytt slumptal skapas när spelet skapas eller när metoden `newGame()` anropas, då också antalet gissningar nollställs.

Test
Uppgiften är att skapa tester för GuessingGame-klassen som kollar att alla krav ovan uppfylls korrekt. De skall även kolla att klassen klarar ”felaktiga” anrop. Det finns en liten subtil bugg i koden som gör att den inte alltid rapporterar rätt resultat enligt kravspecen. Rätta ***inte*** buggen i `GuessingGame` förrän du har ett test som visar på felaktigheten och lyser rött!

Mocking
Det finns ett problem: `GuessingGame` skapar själv det slumptal som skall gissas. Det gör att det näst intill blir omöjligt att testa, åtminstone `makeGuess()`-metoden. Lägg slumptalsgenereringen i en egen klass, som injiceras i `GuessingGame` med DI vid programstart. Beskriv slumptalsgeneratorn med ett interface så att testerna kan använda en alternativ mock-implementering av slumptalsgeneratorn genom att de kan styra vilket ”slumptal” det blir. Använd manuell mocking eller något mock-verktyg, förslagsvis Mockito.

Betyg G
Förse `GuessingGame`-klassen med fullständig uppsättning tester. Mocka slumpgenereringen så att testerna överhuvudtaget kan genomföras, med känt ”slumpvärde”.

Betyg VG
Som för Betyg G plus extra uppgift:
Två alternativ, välj ett av dem för betyget VG.

1.
Utför test av klassen GuessingGameController isolerad, dvs utan `SimpleWindow` och `GuessingGame`. Då behöver båda dessa mockas på lämpligt sätt.

2.
Förse applikationen med ett lämpligt DAO-interface för att lagra resultat och topp-tio-resultat. Du skall inte implementera DAOn, bara mocka den, för att bekräfta att GuessingGame och GuessingGameController beter sig korrekt i förhållande till den. Lägg alltså till mock-implementeringar och lämpliga tester för detta.