Inlämningsuppgift - Pengamaskinen i Fulköping

Din uppgift är att bygga en uttagsautomat till Fulköpings Bank. Banken kommer att vidareutveckla och underhålla kodbasen i många år framöver och har därför bestämt att utvecklingen skall ske med testdriven utvecklning (TDD) och att samtliga användarfall skall vara täckta med enhetstester. I automaten skall du stoppa in ditt bankkort och verfiera att du är du med en PIN-kod. Därefter skall du kunna se saldo, ta ut pengar men även sätta in pengar på ditt konto.

Målet är att du skall visa:

- hur man utvecklar en applikation med hjälp av TDD.
- hur man skapar enhetstester.
- hur man kan mocka funktionallitet.

Banken gillar dig, men du är inte betrodd att ansluta direkt till bankens apier. Därför kommer du att vara tvungen att utveckla denna applikationen med hjälp av en mock av bankapiet. Till uppgiften får du ett enkelt skal med kod (se inlämning nedan) - här finns ett påbörjat inferface för bankens API, men eftersom detta projekt är den första konsumenten av API:et kommer banken att implementera de metoder du behöver (dvs du kan, och behöver, lägga till och ändra metoder i interfacet).

Användarfall

1. När användaren matar in sitt kort används kortets serienummer för att identifera användaren.
2. Användaren anger sin PIN-kod. Pin-koden verifieras via bankens api.
    - Om pin-koden stämmer blir användaren inloggad.

    - Om pin-koden är fel, hämta antalet misslyckade försök från banken.
        - Om användaren har försökt 3 gånger med fel pin-kod, låses kortet via bankens api.
        - Om det är färre antal skall användaren meddelas om hur många försök som återstår - spara antalet försök +1 till banken via dess api.
3. Vid insättning av kort, kontrollera om kortet redan är låst. Om kortet är låst får användaren ett meddelande och blir inte inloggad.
4. När användaren är inloggad blir hen presenterad med följande möjligheter
    - Kontrollera saldo: Hämta saldo från banken och visa det för användaren.
    - Sätt in pengar: Användaren anger ett belopp som hen vill sätta in. Säkerställ att bankens insättningsfunktion anropas på rätt sätt.
    - Ta ut pengar: Användaren anger ett belopp som hen vill ta ut.
        - Kontrollera att beloppet täcks av användarens saldo.
        - Om saldot är tillräckligt, behandla uttaget och säkerställ att bankens uttagsfunktion anropas på korrekt sätt.
        - Om saldot är för lågt visa ett felmeddelande för användaren om att saldot är för lågt.
5. Användaren skall kunna avsluta sin inloggning och kortet skall då matas ut.
6. Bankidentifiering
    - Utagsautomaten skall kunna bekräfta vilken bank den är kopplad till.
    - Använd en statisk metod i den mockade banken för att visa bankens namn. Säkerställ att funktionen fungerar på ett korrekt sätt. Detta steg kräver att du mockar statiska metoder.

Testkrav

- Använd Mockito för att mocka bankens funktioner
- Använder **verify** för att säkerställa att korrekt metoder i den mockade banken anropas vid speciella operationer (t.ex. insättning och uttag).
- Mocka den statiska funktionen som bekräftar bankens identitet för att säkerställa korrekt bankanslutning.
- Din kod skall ha minst 80% testtäckning (ATM-klassen).
- Koden skall vara uppdelad i två paket; main och test.
- Använd anoteringarna **@DisplayName** och **@Test** på alla testmetoder.
- Använd gärna flera annotationer.

Tips för implementation

- Implementera varje steg i TDD-cykeln; skriv först ett test som inte passerar, skapa sedan implementationen och verifiera att testen passerar, refaktorera sedan koden om nödvändigt.
- Använd gränssnitt (interfaces) och mockade klasser för att representera banken, vilket gör att du kan testa uttagsautomaten utan att interagera med ett faktiskt banksystem.
- Använd @BeforeEach för att initialisera mock-objekt och återställ dem mellan varje testfall.
- Se bifogat projekt för inspiration och för att komma i gång snabbare.

Muntlig redovisning

Spela in en video där du beskriver ditt projekt och hur du har gått tillväga. Du skall tydligt synas i videon och det skall gå att höra vad du säger. Du pratar fritt i redovisningen dvs du får inte läsa innan till. Videon skall vara mellan 5 och 10 minuter lång, inte mer eller mindre. Använd följande guide för din redovisning.

1. Kör dina tester och visa att du har minst 80% av din kod täckt med test.
2. Förklara hur du följde TDD-processen.
3. Vad var den största utmaningen du stötte på när du implementerade din tester och hur löste du dem?
4. Visa och förklara hur du använder olika assert-metoder i dina tester.
5. Finns det någon del av koden du tänkte refaktorera men valde att inte göra det? Varför?
6. Vad lärde du dig av att skriva test för koden?

Några frivilliga frågor du kan använda i din muntliga redovisning.

1. Vad används @BeforeEach- och @AfterEach-annoteringarna till?
2. Vad är syftet med @Test-annoteringen och vilka andra annoteringar känner du till?
3. Vad är skillnaden mellan assertEquals, assertTrue, och assertThrows i Junit?
4. Vad är skillnad mellan assertAll() och assertTrue eller assertFalse?
5. Vad är skillnaden mellan en mock och en stub, och när skulle du använda respektive?
6. Hur använder du ett verktyg som Mockito för att skapa mocks i dina tester?
7. Vilka är fördelarna med att använda mocking när man testar klasser som har externa beroenden?
8. Vad är skillnad melan @Mock och @Spy?

Inlämning

Använd GitHub Classroom för att få tillgång till ett repo där utvecklingen skall ske. Detta repo kommer att innehålla ett startskelett som du skall använda i uppgiften. Sista tidpunkt för commit är 2025-11-14 kl. 23:59.

Lämna in din videoinspelade muntliga redovisning på Google Classroom senast 2025-11-14 kl. 23:59.

Koden skall vara välformaterad och följa god praxis.

Betygskriterier

Godkänt

1. Implementera alla grundläggande funktionaliteter som specificerats i projektbeskrivningen.
2. Skriva enhetstester för alla metoder du implementerar med korrekt användning av mocking för att isolera beroenden.
3. Genomföra en muntlig redovisning där ni presenterar din kod och de tester du skrivit. Du ska kunna förklara ditt val av implementering och testning, samt svara på frågor om arbetet.

Väl Godkänd

1. Implementera ytterligare funktioner eller förbättringar som går utöver de grundläggande kraven, vilket visar på en djupare förståelse och kreativ tillämpning av TDD och mocking.
2. Excellera under den muntliga redovisningen genom att tydligt och övertygande argumentera för era tekniska beslut.

Lycka till!