Det er antatt at brukeren av applikasjonen og leseren av denne dokumentasjonen har grunnleggende kunnskap om kasinospillet blackjack.

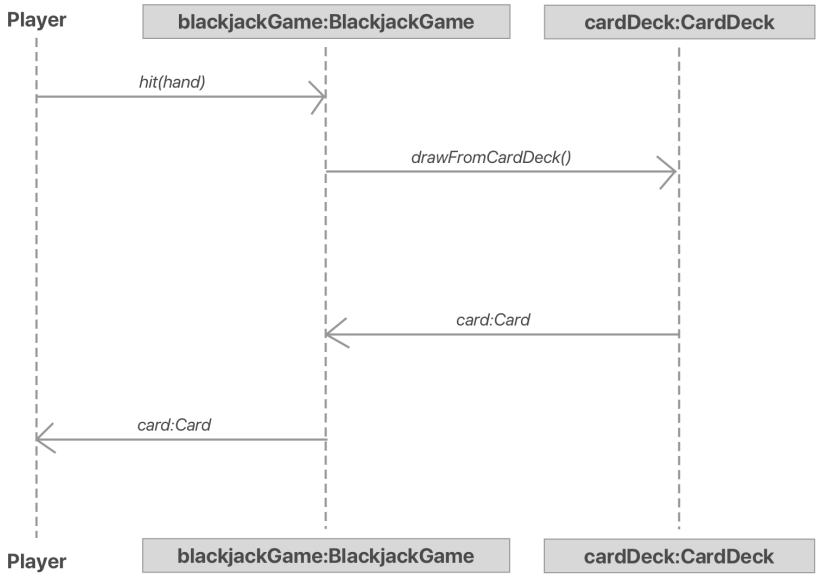
Del 1 – Beskrivelse

Vi har i dette prosjektet utviklet en blackjack-applikasjon. Applikasjonen er et spill der spilleren (brukeren) spiller mot dealeren (datamaskinen). Applikasjonen følger tradisjonelle blackjack-regler med noen unntak:

1. Fem kort uten å «sprekke» gir automatisk blackjack
2. Det er ikke mulighet for «insurance»
3. Man kan doble når man vil
4. Det er ikke mulighet for å splitte

Videre er kortstokken alltid 6 separate kortstokker stokket sammen, dersom det gjenstår mindre enn 10 kort i kortstokken ved starten av en runde, vil den bli stokket på nytt. Spilleren får utdelt en lommebok på 100 fiktive penger hver gang spillet åpnes, og satser en sum før hver runde. Når hver runde begynner vil spilleren og dealeren få utdelt to kort hver som vises på skjermen, hvor et av dealerens kort er skjult. Spilleren har da tre valg, «hit», «double» eller «stand». Dersom spilleren vinner, utbetales 2x innskudd. Dersom spilleren får «naturlig blackjack» er gevinsten på 2.5x innskudd. Resultathistorikken til spilleren logges i en fil som leses og vises i appen. Historikken kan også nullstilles fra appen. Summen for «hånden» vises ikke og dette må regnes ut selv av brukeren. Dette er et bevisst valg for å gjøre spillopplevelsen mer autentisk.

Del 2 – Diagram



Sekvensdiagram

Sekvensdiagrammet illustrerer et utplukk fra kortstokken. Dette kjøres hver gang et kort blir valgt ut av kortstokken som er en svært vesentlig funksjon for applikasjonen. Dette gjelder når kortene deles ut i starten av runden og hver gang spilleren ønsker fler kort. Vi valgte å bruke et sekvensdiagram for å illustrere relasjonen mellom klassene, samt differensieringen og delegeringen av funksjonalitet i applikasjonen.

Del 3 – Spørsmål

1. **Hvilke deler av pensum i emnet dekkes i prosjektet, og på hvilken måte? (For eksempel bruk av arv, interface, delegering osv.)**

   Vi har dekket store deler av pensumet i dette prosjektet. Det er verdt å merke seg at dette er et relativt lite prosjekt med lite kode. Det er derfor ikke alle prinsipper og teorier som er relevant eller hensiktsmessig å benytte da dette kan virke mot sin hensikt og lage mer rot. Likevel har vi implementert forskjellige teknikker og «best practices» som er verdt å ta med seg videre til større prosjekter. Disse er beskrevet under.

   Interface

   I klassen *WinStatistics.java* implementeres grensesnittet *GameStatisticsManager.java*. Dette er strengt tatt ikke nødvendig ettersom grensesnittet ikke benyttes i flere klasser og størrelsen på prosjektet er lite. Likevel bidrar grensesnittet til økt kontroll på oppsettet av klassen og dets metoder, spesielt i et større prosjekt der flere klasser hadde delt samme grensesnitt.

   Feilhåndtering

   Gjennom hele prosjektet er det implementert feilhåndtering for å sikre god UX (user experience). Dette er gjort ved å bruke *try/catch* og kasting av *error*. Det er også en dialogboks i spillet som viser disse feilmeldingene og fasiliterer for god UX.

   Lese/skrive til fil

   Resultathistorikken til spilleren blir skrevet til en .txt fil gjennom *WinStatistics.java*-klassen etter hver runde. Resultatene blir også lest av programmet, og et forhold over antall seire og antall runder vises til brukeren. Brukeren kan selv også nullstille historikken.

   Innkapsling og gyldighet

   For å sikre klassenes integritet og gyldighet har vi benyttet oss av innkapslingsprinsippet. Alle endringer av klassenes tilstand skjer ved å benytte klassens egne metoder for å gjøre endringer i instansen.

   Delegering

   Delegeringsprinssippet er benyttet i prosjekter, noe som gjør prosjektet ryddigere og lettere å vedlikeholde. Et eksempel på dette er at funksjonaliteten når det gjelder enkeltkort ligger i *Card.java* klassen. Videre er funksjonalitet knyttet til kortstokken i *CardDeck.java* klassen. Dermed er all funksjonalitet knyttet til kortene delegert vekk fra de forskjellige klassene som benytter dem, som *BlackjackHand.java* klassen.

2. **Dersom deler av pensum ikke er dekket i prosjektet deres, hvordan kunne dere brukt disse delene av pensum i appen?**

Arv

Det er ikke benyttet arv i dette prosjektet. Grunnen til dette er at det ikke har vært hensiktsmessig i et så lite prosjekt. Det er ikke klart for oss hvor det hadde vært gunstig å innføre arv i vår applikasjon, slik den er bygget opp nå. Innledningsvis ble det forsøkt å lage en abstrakt klasse med navn CardContainer, som både BlackjackHand og CardDeck skulle arve fra. Underveis i prosjektet innså vi derimot at det eneste felles attributtet mellom BlackjackHand og CardDeck var et tilfelle av List<Card>. Videre var både konstruktør og samtlige av metodene deres helt forskjellige. Av denne grunn gikk vi derfor vekk fra ordningen med å benytte CardContainer som en abstract, og definerte de to klassene uavhengig av hverandre.

Observatører

Vi har ikke benyttet oss av observatører i henhold til observatør/observert teknikken. Slik applikasjonen er utviklet nå ser vi ikke hvor en observatør hadde vært hensiktsmessig. Eventuelt kunne et kort ha en observatør som trigges hvis kort blir trukket ut av bunken eller lignende, men som tidligere beskrevet ville ikke dette vært særlig hensiktsmessig.

3. **Hvordan forholder koden deres seg til Model-View-Controller-prinsippet? (Merk: det er ikke nødvendig at koden er helt perfekt i forhold til Model-View-Controller standarder for å få full uttelling på dette spørsmålet. Det er mulig (og bra) å reflektere rundt svakheter i egen kode)**

Vi har i all hovedsak fulgt MVC prinsippet i prosjektet. Det er minimalt med logikk i controlleren. Eneste logikken her er noen if setninger og feilhåndteringer som kreves for å lese verdier fra input-feltene. Denne logikken kan selvfølgelig helt fint flyttes ut fra controlleren også, men vår oppfatning var at dette kunne blitt både knotete og rotete. Men det er klart at dersom vi skulle fulgt MVC fullt ut, måtte også denne logikken ut av controlleren. Resterende, og dermed nesten all logikk befinner seg derfor i model-delen av applikasjonen.

Mesteparten av kode knyttet til brukergrensesnittet ligger i fxml filene. Vi har noe java kode, for å generere bildene knyttet til kortene som ligger på bordet, som ikke er separert fra resten av koden. Dette måtte da også flyttes ut til en egen klasse for å tilfredstille MVC prinsippet 100%.

4. **Hvordan har dere gått frem når dere skulle teste appen deres, og hvorfor har dere valgt de testene dere har? Har dere testet alle deler av koden? Hvis ikke, hvordan har dere prioritert hvilke deler som testes og ikke? (Her er tanken at dere skal reflektere rundt egen bruk av tester)**

   Vi har benyttet JUnit for å skrive test cases for prosjektet. Vi har gått frem ved å skrive testkode til alle klassene og alle mulige edge-cases vi kunne forestille oss. Vi har erfart at man støter på forskjellige feil, selv etter vi trodde vi hadde testet alt, så det er derfor fortsatt mulig at vi ikke tester alle aspekter ved programmet.

   Bruk av tester har også gjort selve utviklingen raskere. Vi begynte å skrive testkode fortløpende, og dette har sikret at alle funksjonene ved applikasjonen funket underveis i utviklingen. Vi har flere ganger opplevd å legge til mer funksjonalitet som samtidig har påvirket tidligere funksjonalitet negativt. Det har derfor vært svært hensiktsmessig å bruke test-cases aktivt i utviklingen for å spare tid ved feilsøking.

5. **Har dere møtt på noen utfordringer i løpet av prosjektet? Hva ville dere gjort annerledes en annen gang?**

   Vi hør støtt på få problemer i utviklingen av prosjektet. Vi har selvfølgelig stått fast ved små utfordringer knyttet til løsninger av ulike programmeringsproblemer, men samarbeidet har funket godt. Et av problemene vi har møtt på er bildekonvertering. Dette befant seg utenfor pensum og vi måtte ty til StackOverflow for å finne en fungerende løsning. Ellers har utfordringene vi har kommet over blitt løst med prøve/feiling og oppslag i pensum. Til en annen gang ville vi brukt enda mer tid på å planlegge prosjektet i forkant før selve utviklingen. En grundigere planlegging, helt ned til hvilke metoder som skal være med i hvilke klasser, kunne gjort utviklingen mer effektiv. Vi måtte flere ganger endre litt på oppbygningen og innhold i klassene underveis i utviklingen. Litt mer planlegging og vi kunne spart oss for en del unødvendig arbeid.