Rapport for Eksamen – PG3300 Software design 2018

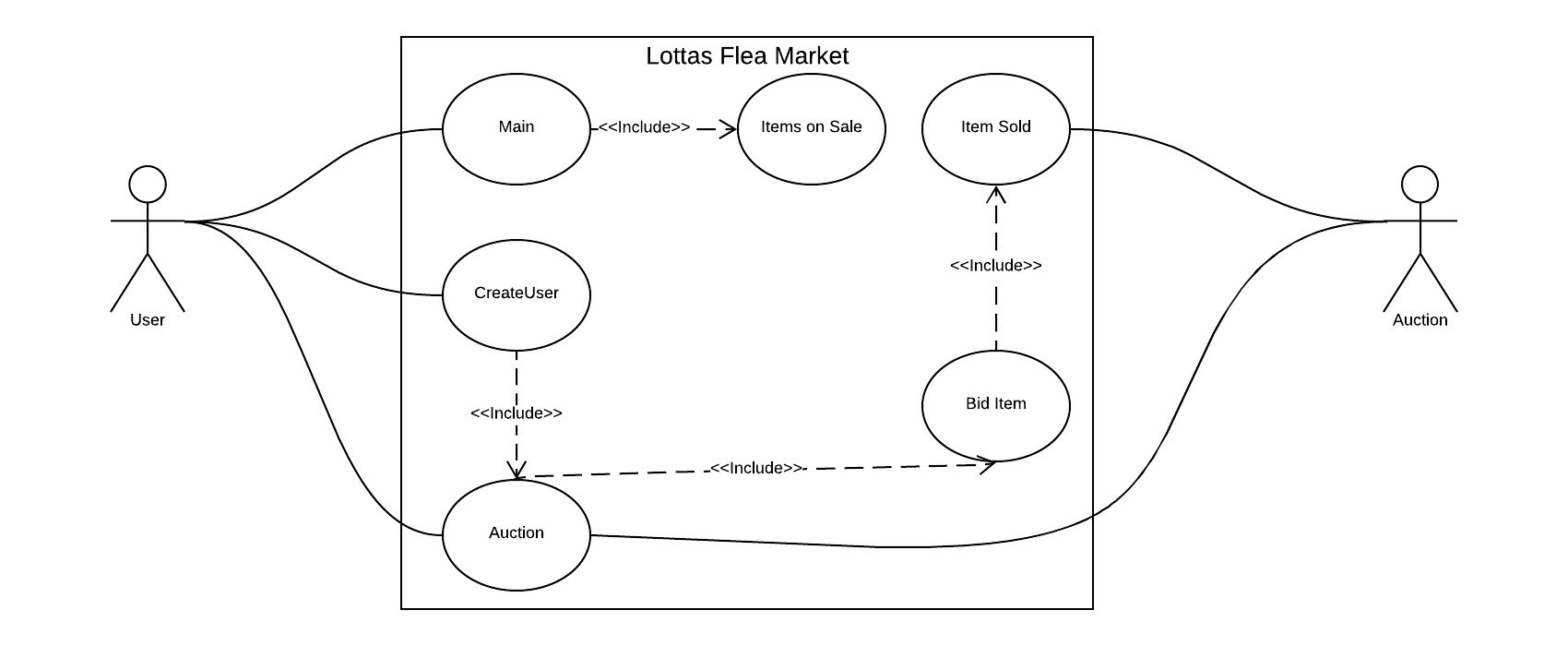


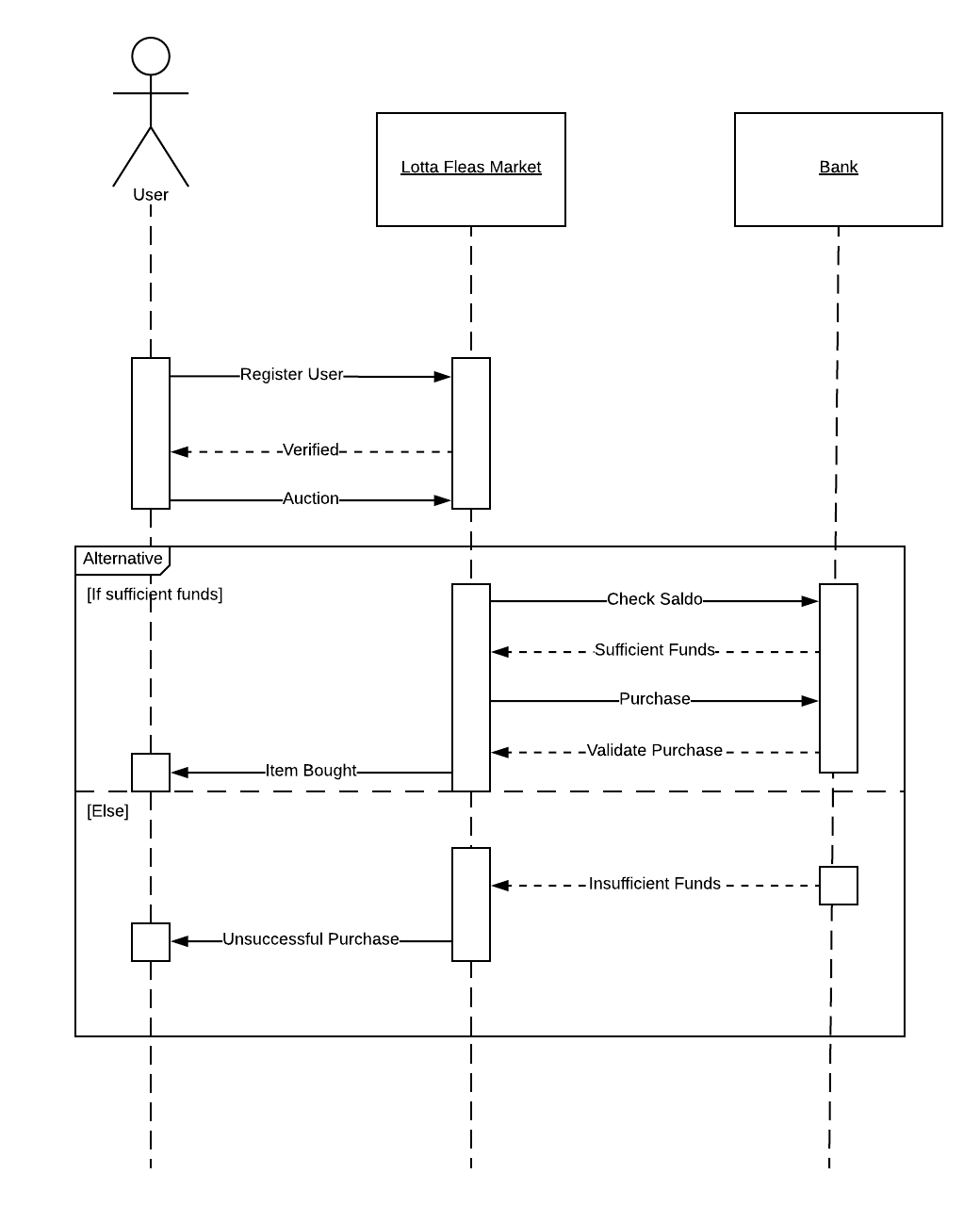
Skrevet av:

Olav Angell, Benjamin Le og Andreas Martila

UML, design og oppgave tolkning:

Måten vi tolker oppgaven er to delt. På en side ser vi på hvordan «Lottas Loppemarked» kan ha en digital løsning, men på en annen side tenkte vi på hvordan vi kan simulere en løsning for Lotta. Vår tanke er å lage en løsning som besvarer begge disse tolkningene og har ledet oss til vårt endelige produkt.

I starten var det pen og papir for planlegging av hvordan vi ville takle oppgaven. Vi så for oss at en bruker registrer seg hos «Lottas Loppemarked» og så kan delta på en auksjon for å kjøpe artikler som er ute for salg. UML under viser hva vi startet med. 



Vi ser for oss at vi kan besvare begge tolkninger av oppgaven på denne måten. I vår «main» klasse generer vi falske brukere via bruker klassen, slike en ekte bruker vil, og så få tilgang til å by på auksjonene med bank data. Vi har valgt å bruke en bank-klasse for å simulere programmets kommunikasjon med en eventuell bank, for penge delen av løsningen.

Modellering:

I vår løsning har vi fokusert på flest mulig patterns. Vår UserFactory-klasse har factory design, Program-klassen har decorator, AuctionHouse-klassen er composite og bank-klassen har singleton. Bakgrunnen for dette hovedsakelig å vise at vi behersker flest mulig design patterns.

UserFactory-klassen har vi skrevet til å være switch-case basert, enten lager den bots eller så lager de en bruker til brukeren. Brukeren i vår løsning kan være med å by blant de andre botsene på de virtuelle artiklene som er lagt ut for salg.

AuctionHouse-klassen tar for seg dekorator, fordi den har kontroll over artiklene som auksjoneres og bud rundene der til. I tillegg sjekker den om brukeren har råd til å by og tar det vinnende budet til banken for loggføring.

Bank-klassen har bare en instans og passer til å lage etter singleton design.

Vi diskuterte lenge om hvordan vi kunne lage en fornuftig trådløsning og om det er gunstig for vår løsning. Det kunne vært en implementering om man vil holde flere auksjoner samtidig, men vår løsning har muligheten for en bruker av programmet kan ta del i auksjonene, og da er det ikke realistisk å ha flere auksjoner gående samtidig.

Det er heller ikke realistisk at en bruker er på flere auksjoner samtidig og byr, derfor har vi valgt å ikke bruke flere tråder i vår løsning. Fordelen er at vi da har en bedre simulering av vårt program.

Inkluderte bibliotek:

* System
* System.Collections.Generic
* System.Threading
* System.Diagnostics

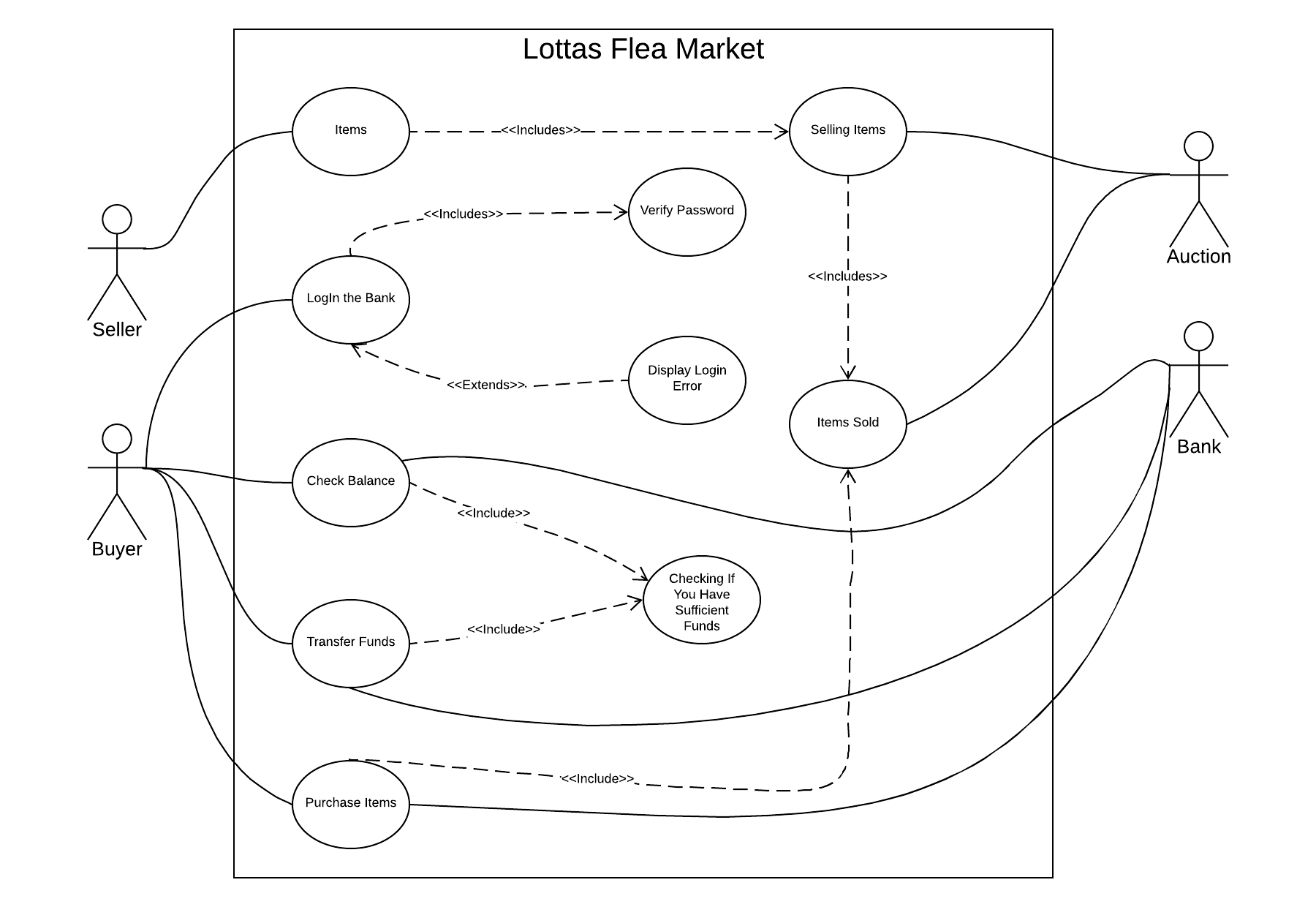
Implementering:

Vi har benyttet parprogrammering i stor grade i denne løsningen. Vi oppdaget tidlige at vår prototype på papiret ikke ville bli lik når vi kodet. Vi har bevart mesteparten av det vi startet med, men ekspandert for en bedre løsning jo lengre vi kom på kode delen.

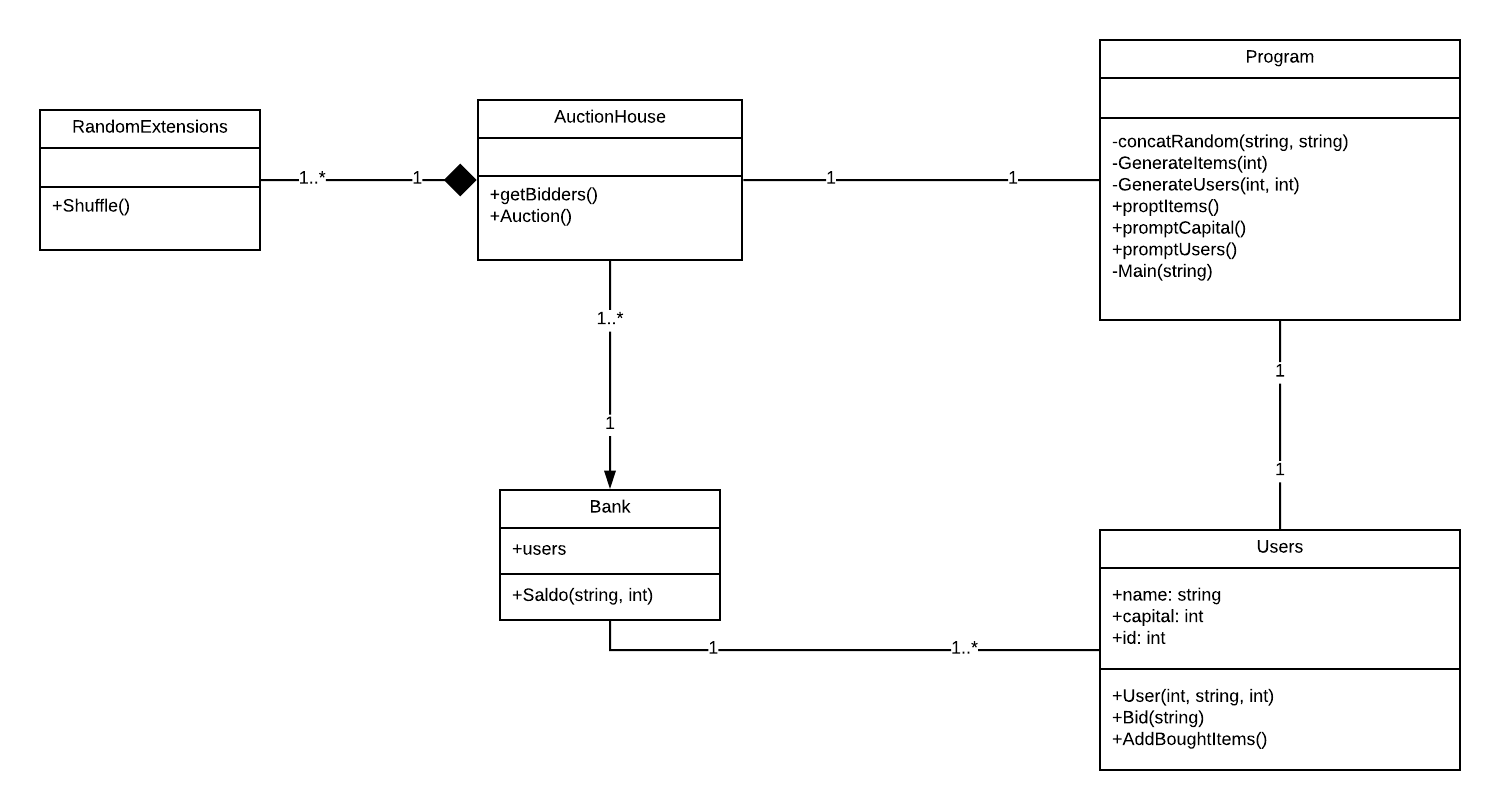
Det å bruke parprogrammering var veldig hjelpsomt når vi stod fast. I starten var det veldig greit å bevare flyten i kodingen. Problemer for skribent ble hurtig løst fra side mannen som observerte problemstillingen, og vi byttet ofte på hvem som fysisk skrev koden.

Vårt endelige produkt UML er under:

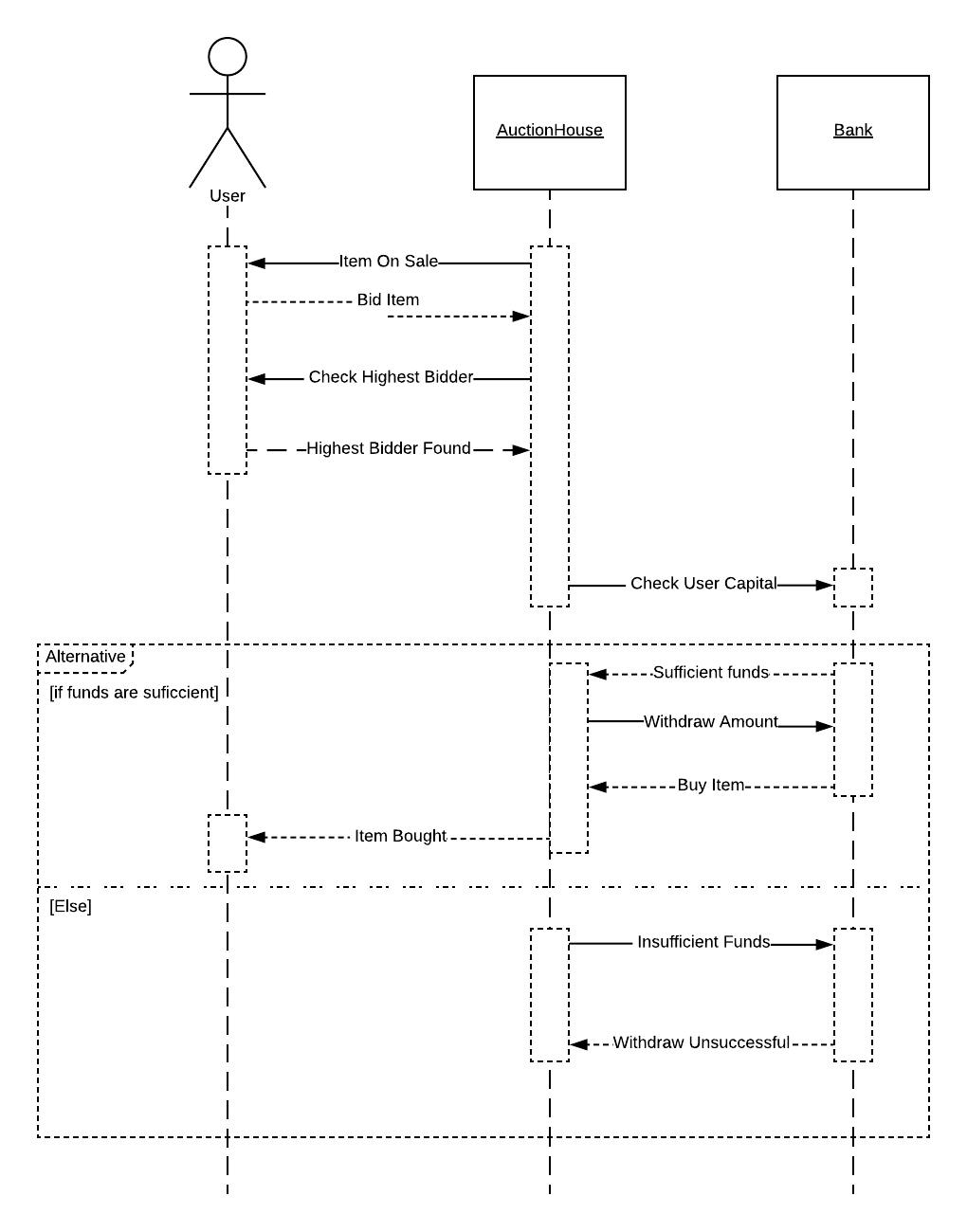
Use Case Diagram:



Class Diagram:



Sequence Diagram:



Konklusjon:

Kilder:

<https://www.dotnetperls.com/>

<https://www.lucidchart.com/>

<https://visualstudio.microsoft.com/>

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/>