

1 Teori

Markera om följande påståenden är sanna eller falska:
(+1 för rätt svar, ingen förändring för fel svar)

Ett objekt har metoder och attribut.

☐ Sant



☐ Falskt

Ett Use Case beskriver hur olika aktörer använder systemet

☐ Sant



☐ Falskt

Ett Use Case beskriver vilka metodanrop som användare gör mot specifika klasser i systemet.

☐ Falskt



☐ Sant

Ett Sekvensdiagram visar på hur olika objekt samarbetar för att lösa och ge ett svar till en systemhändelse.

☐ Sant



☐ Falskt

Ett Sekvensdiagram beskriver vilka attribut och metoder varje klass måste ha.

☐ Sant

☐ Falskt



Ett Klassdiagram visar alltid den minsta möjliga mängden klasser som behövs för att bygga systemet.

☐ Falskt



☐ Sant

En Domänmodell visar hur olika klasser samarbetar.

☐ Sant

☐ Falskt



Ett Use Case kan användas som ett testfall för ett system.

- ☐ Falskt
- ☐ Sant



Ett Use Case beskriver alla enhetstester för en viss klass.

- ☐ Falskt
- ☐ Sant



Ett Samarbetsdiagram innehåller samma information som ett Sekvensdiagram.

- ☐ Falskt
- ☐ Sant



Ett Entity-Relationship-diagram visar på hur olika objekt samarbetar med varandra.

- ☐ Sant
- ☐ Falskt



Totalpoäng: 11

2 Use Cases

Givet nedanstående Use Case och de tre förslagen på systemsekvensdiagram, vilket systemsekvensdiagram följer *närmast* arbetsflödet beskrivet i use caset?

Use Case: Apply Voodoo

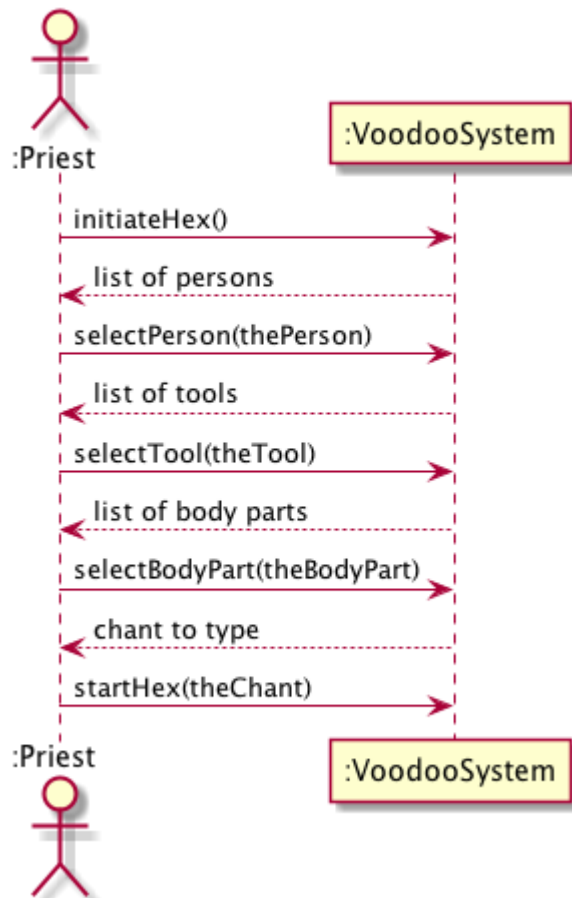
Actors: Priest

Description: A Voodoo Priest selects a person to hex, a tool to use, and a part of the selected person's body. The system performs the magic and puts a hex on the selected person.

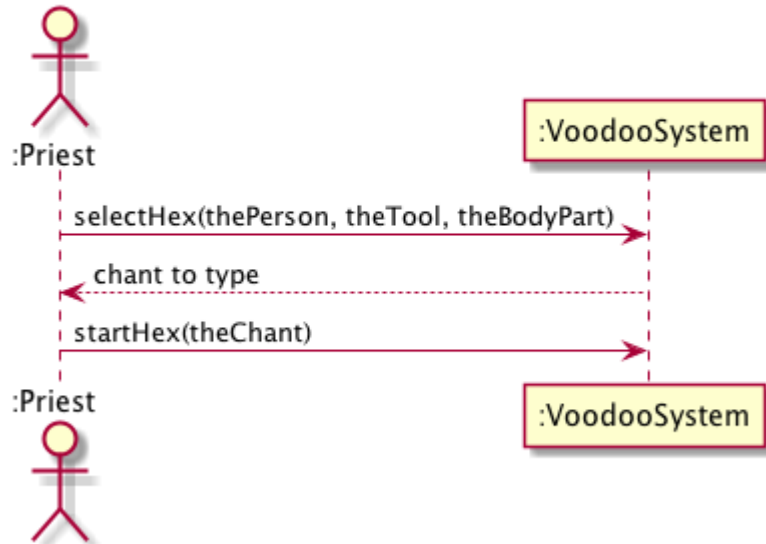
Main Course of Events

Actor	System
1. Priest initiates a Voodoo Hex	2. The system presents a list of persons to hex
3. The Priest selects a person	4. The system indicates that the person is selected and presents a list of tools.
5. The Priest selects a tool	6. The system indicates that the tool is selected and asks which part of the body to hex.
7. The priest selects a part of the body.	8. The system applies the tool to the body part. The system displays a suitable chant for the priest to read in order to seal the hex.
9. The Priest types the chant back to the system and starts the hex.	10. The system performs magic in order to put a hex on the selected person and body part.

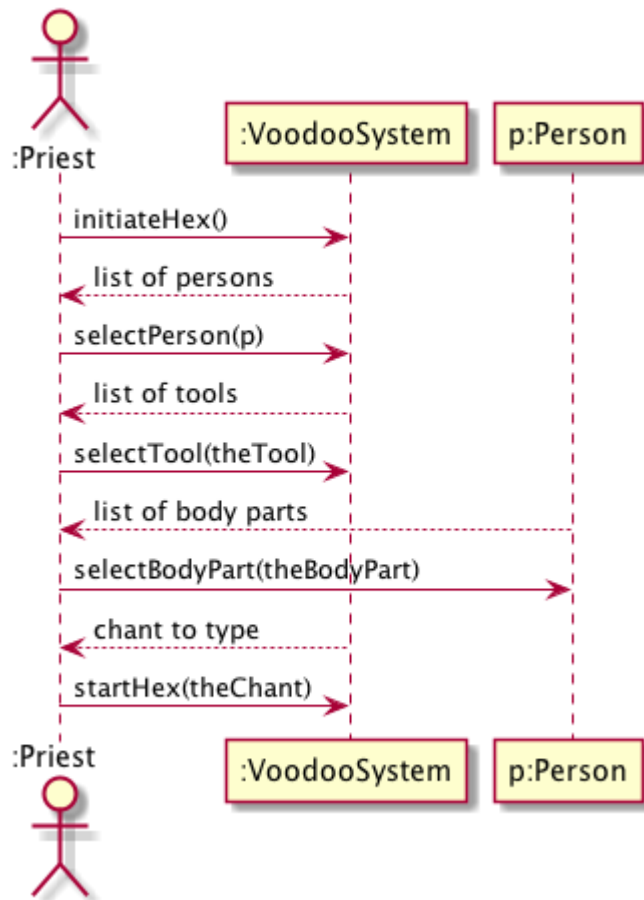
Alternative A



Alternative B



Alternative C



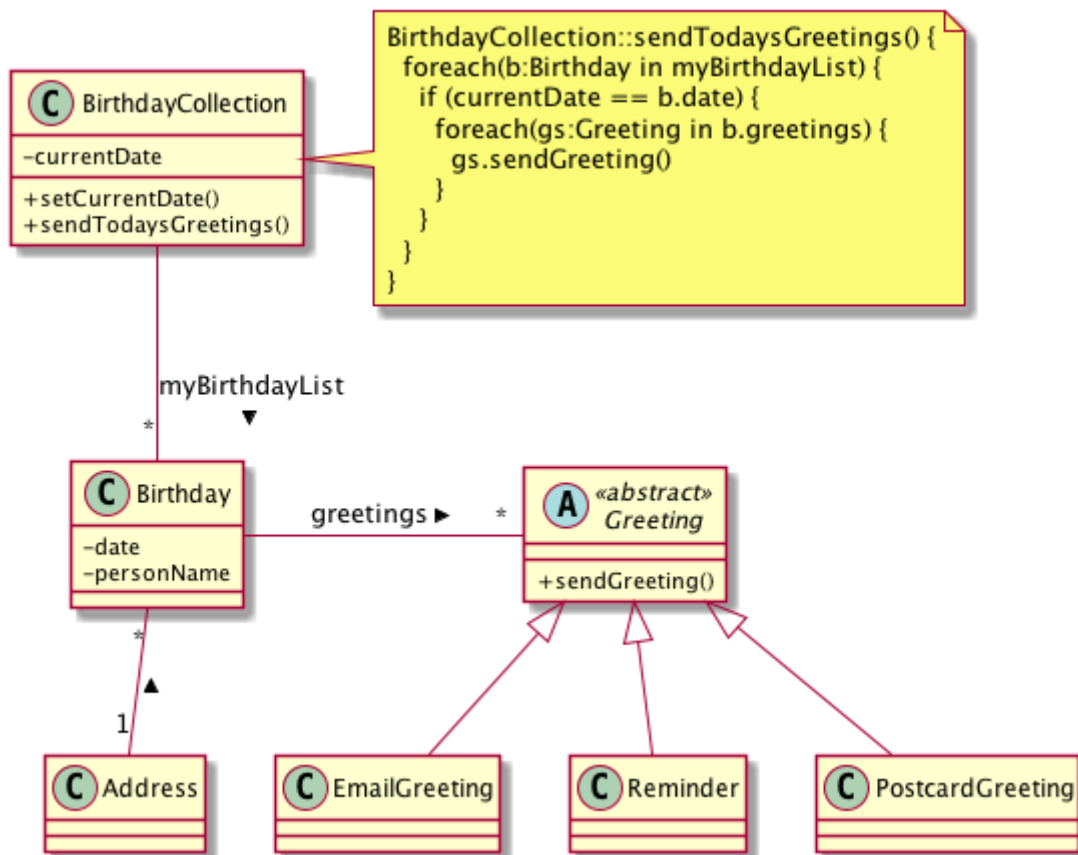
vilket systemsekvensdiagram följer närmast arbetsflödet beskrivet i use case?

- ☐ Alternative A
- ☐ Alternative C
- ☐ Alternative B



Totalpoäng: 1

3 Klassdiagram



Klassdiagrammet beskriver en del av en applikation som håller koll på dina vänners födelsedagar och skickar olika sorters födelsedagshälsningar till dem.

För varje påstående nedan, markera om klassdiagrammet stödjer påståendet (sant) eller inte stödjer påståendet (falskt).

Vid en viss address kan det finnas flera födelsedagar.

- ☐ Falskt
☐ Sant



En födelsedag kan ha flera olika gratulationer.

- ☐ Sant
☐ Falskt



<> Greeting betyder att man får skapa abstrakta objekt av typen Greeting.

- ☐ Falskt
☐ Sant



metoden `BirthdayCollection::sendTodaysGreetings()` bryter mot GRASP-mönstret **low coupling**.

- ☐ Sant ✓
- ☐ Falskt

objektet `r` av typen `Reminder` har en metod `sendGreeting()`.

- ☐ Falskt
- ☐ Sant ✓

objektet `a` av typen `Address` har två privata attribut `date` och `personName`.

- ☐ Falskt ✓
- ☐ Sant

klasserna `Birthday`, `Greeting`, `EmailGreeting`, `Reminder`, och `postcardGreeting` bildar tillsammans designmönstret **Strategy**.

- ☐ Sant ✓
- ☐ Falskt

Klassen `Birthday` har ett attribut `greetings`.

- ☐ Falskt
- ☐ Sant ✓

Klassen `Birthday` har ett attribut `myBirthdayList`.

- ☐ Sant
- ☐ Falskt ✓

attributet `greetings` kan bara innehålla objekt av typerna `EmailGreeting`, `Reminder`, och `PostcardGreeting`.

- ☐ Falskt
- ☐ Sant ✓

Klassen `Birthday` är en **Information Expert** för en viss födelsedag.

- ☐ Falskt
- ☐ Sant ✓

Klassen BirthdayCollection är en Singleton.

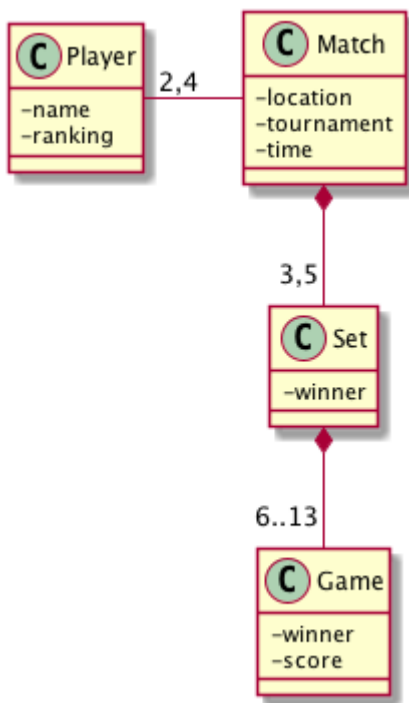
☐ Falskt



☐ Sant

Totalpoäng: 12

4 Relationer mellan Klasser



Klassdiagrammet beskriver en tennismatch. Notera att detta diagram fokuserar på relationerna mellan klasserna, och går inte in på detaljer om vilka metoder eller attribut som finns.

För varje påstående nedan, markera om relationerna mellan klasserna stödjer påståendet (sant) eller inte stödjer påståendet (falskt).

En match har mellan två och fyra Players.

- ☐ Sant
☐ Falskt



När en Player har vunnit 6 st Game så har han vunnit ett Set.

- ☐ Sant
☐ Falskt



En Match består av 3 eller 5 Set.

- ☐ Sant
☐ Falskt



Objekten s1:Set och s2:Set har båda en referens till samma objekt g1:Game.

☐ Sant

☐ Falskt



p1:Player spelar i m1:Match samtidigt som p2:Player spelar i m2:Match.

☐ Sant

☐ Falskt



p1:Player möter p3:Player i samma m1:Match.

☐ Falskt

☐ Sant



Det kan högst finnas $5 \cdot 13 = 65$ objekt av typen Game i det här systemet.

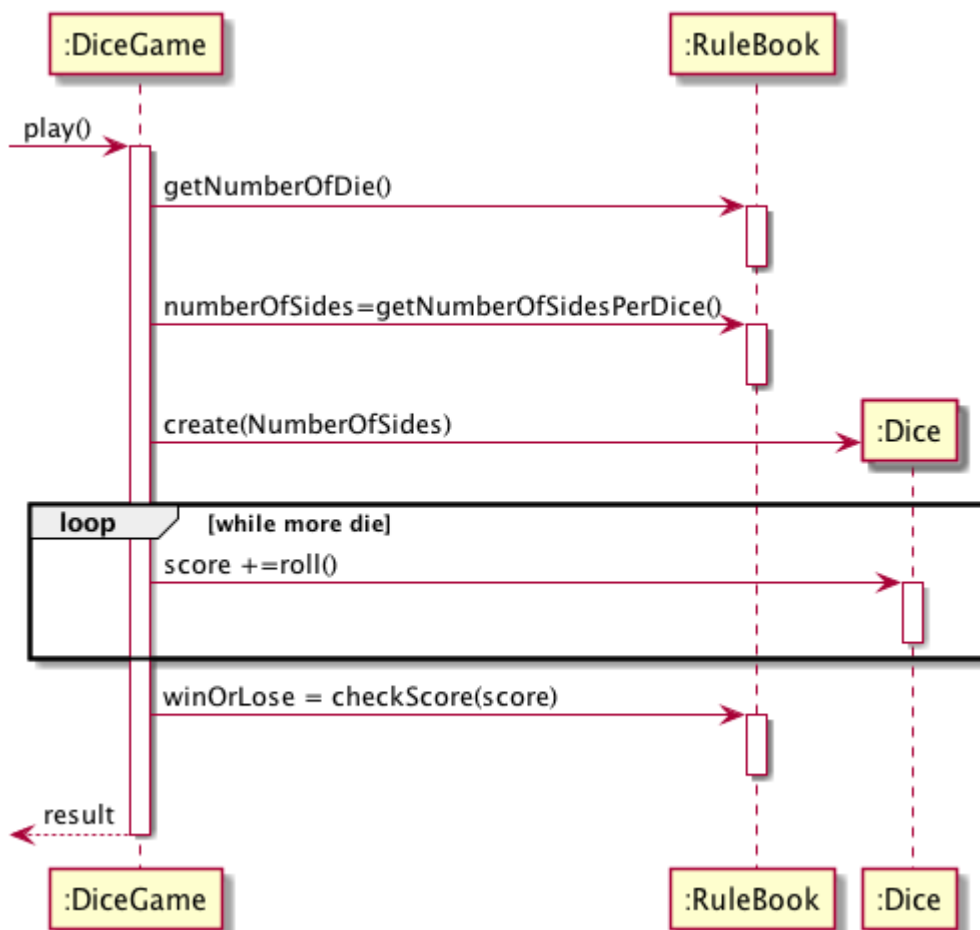
☐ Sant

☐ Falskt



Totalpoäng: 7

5 Interaktionsdiagram och GRASP



Sekvensdiagrammet beskriver ett enkelt tärningsspel.

Markera om följande påståenden är sanna eller falska:
(+1 för rätt svar, ingen förändring för fel svar)

:RuleBook har metoderna "getNumberOfDie()" och "getNumberOfSidesPerDice()"

- ☐ Sant
- ☐ Falskt



:DiceGame är en Controller

- ☐ Sant
- ☐ Falskt



:Dice är en Controller

☐ Sant

☐ Falskt



:RuleBook är en Controller

☐ Sant

☐ Falskt



:RuleBook är en Information Expert på hur tärningsspelet går till.

☐ Sant

☐ Falskt



:DiceGame är en Information Expert för en viss omgång av spelet.

☐ Falskt

☐ Sant



Man skulle få lägre Coupling om :RuleBook också blev en Creator av :Dice.

☐ Sant

☐ Falskt



:Dice är en Polymorfism

☐ Sant

☐ Falskt



Totalpoäng: 8

6 Design Patterns I

Markera om följande påståenden är sanna eller falska:
(+1 för rätt svar, ingen förändring för fel svar)

Designmönstret Abstract Factory är bara ett specialfall av Strategy Pattern

- ☐ Sant ✓
- ☐ Falskt

Observer Pattern handlar om att man som användare av systemet kan "följa med" och observera hur ens data hanteras.

- ☐ Falskt ✓
- ☐ Sant

Man får bara använda designmönstret Singleton en gång i ett system

- ☐ Falskt ✓
- ☐ Sant

Designmönstret Strategy Pattern handlar om att man vill kunna lösa en viss uppgift på olika sätt, så man behöver ha olika implementationer som kompilatorn kan hjälpa till att välja mellan.

- ☐ Sant ✓
- ☐ Falskt

I Strategy Pattern har en klass rollen <>, vilket innebär att den är ansvarig för att hålla koll på vilken strategi som är aktuell för stunden och skicka vidare anrop från resten av systemet till den aktuella strategin.

- ☐ Falskt
- ☐ Sant ✓

Singleton använder sig av Pure Fabrication

- ☐ Sant
- ☐ Falskt ✓

Designmönstret State Pattern handlar om att man vill separera kontroll och styrning av ett program från alla objekt.

- ☐ Falskt
- ☐ Sant



Totalpoäng: 7

7 Design Patterns II

a) Om man bromsar kraftigt med en bil så skall en massa olika saker hända; säkerhetsbälten skall sträckas, nackstöd skall skjutas fram något, radion skall pausas, osv. Bromsarna kan använda sig av designmönstret (State, Singleton, Strategy, **Observer**, Factory) för att meddela resten av systemet när bilen bromsar kraftigt.

b) I ordbehandlaren MyWord så kan man spara dokument i flera olika filformat. Systemet använder sig av designmönstret (Singleton, State, Observer, Factory, **Strategy**) för att hantera detta.

c) I ordbehandlaren MyWord finns det ett status-fält längst ner på skärmen. Oavsett vad man gör så finns det bara ett sådant status-fält, men alla delar av systemet kan ha information som skall visas där. För att systemet skall hitta till rätt instans av Statusfält-klassen använder det sig av designmönstret (State, Factory, Strategy, Observer, **Singleton**)

d) I Konfigurationshanteringsverktyget git kan en fil vara *Untracked*, *Modified*, *Staged*, *Committed*, och *Pushed*. Beroende på vilken status en fil har just nu så kan man göra olika saker med den, och olika information skall visas för den. Dessutom så finns det regler som t.ex. säger att man måste först sätta status till *Staged* innan en fil kan bli *Committed*, och en fil måste vara *Committed* för att kunna bli *Pushed*. För att lösa detta kan man använda designmönstret (Factory, Observer, **Singleton**, State, Strategy)

Totalpoäng: 4

8 GRASP Patterns

Markera om följande påståenden är sanna eller falska:
(+1 för rätt svar, ingen förändring för fel svar)


Enligt Information Expert skall man sprida kunskap om ett visst fenomen mellan så många klasser som möjligt så att alla kan lösa alla uppgifter tillsammans.

- ☐ Falskt 
- ☐ Sant

En klass kan vara både en Creator och en Controller

- ☐ Falskt
- ☐ Sant 

Controller är egentligen bara en tillämpning av Pure Fabrication

- ☐ Falskt 
- ☐ Sant

Vilken klass som helst kan vara en Controller

- ☐ Falskt
- ☐ Sant 

Enligt High Cohesion bör en klass inte vara Controller för mer än en sak

- ☐ Sant 
- ☐ Falskt

Enligt Low Coupling skall man ha så många små objekt som möjligt så man måste växla mellan dem så ofta som möjligt

- ☐ Sant
- ☐ Falskt 

Polymorphism gör det möjligt för olika klasser i en arvshierarki att ge olika svar på samma fråga.

- ☐ Falskt
- ☐ Sant



Enligt High Cohesion skall varje klass ha så få och välvgränsade ansvarsområden som möjligt.

- ☐ Sant
- ☐ Falskt



Totalpoäng: 8

i Betygsgränser

Betygsgränserna för den här tentan är:

Betyg	Procent	Poäng
MAX	100%	58
A	90%	52
B	80%	46
C	70%	40
D	65%	37
E	60%	34

Lycka till!