

# 1 Teori - I

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

**Ett klassdiagram visar alla klasser, metoder, och attribut för de objekt som används i alla interaktionsdiagram som gjorts för systemet.**

☐ Sant



☐ Falskt

**Ett use case diagram visar hur man använder en viss klass.**

☐ Sant



☐ Falskt



**En domänmodell är en modell av alla domäner som en viss klass behöver känna till.**

☐ Sant

☐ Falskt



**Att klassen "Äpple" ärver från klassen "SpelElement" innebär att alla metoder och attribut i "SpelElement" också finns i "Äpple".**

☐ Sant



☐ Falskt

**man använder systemsekvensdiagram för att visa vilka händelser användare i ett visst use case genererar mot systemet, och vad systemet skall svara med.**

☐ Sant



☐ Falskt

**Ett samarbetsdiagram och ett klassdiagram visar båda vilka objekt som samarbetar med varandra.**

☐ Sant

☐ Falskt



**Ett Design Pattern är ett generellt förslag på lösning till ett vanligt förekommande programvarudesignproblem.**

☐ Sant



☐ Falskt

**I ett klassdiagram visar man värdet på alla attribut i klasserna.**

☐ Sant

☐ Falskt

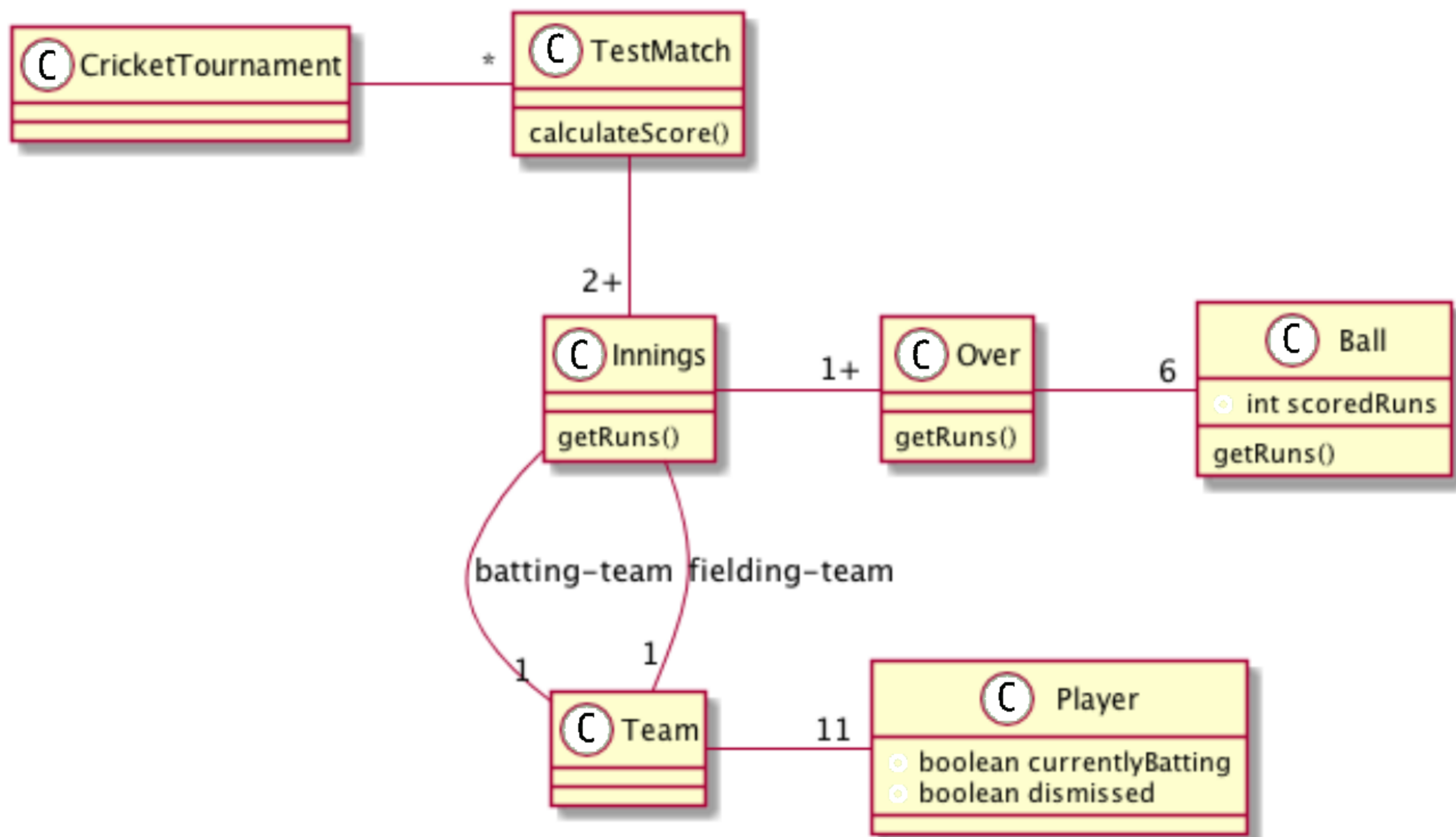


---

Totalpoäng: 8

## 2 Klassdiagram

Betrakta följande klassdiagram:



För varje påstående nedan, markera om klassdiagrammet stödjer påståendet (sant) eller inte stödjer påståendet (falskt) (+1 för rätt svar, ingen förändring för fel svar)

**En Cricket-turnering behöver inte ha några testmatcher (TestMatch).**

☐ Sant

☐ Falskt

**Varje TestMatch måste ha minst två Innings.**

☐ Sant

☐ Falskt

**En Innings består av minst en Over.**

☐ Sant

☐ Falskt

**En Innings får som mest ha 256 Overs.**

☐ Sant

☐ Falskt

**För en Innings så blir det 6, 12, 18, 24, ...,  $6 \cdot n$  Balls som spelas.**

☐ Sant

☐ Falskt



**Ett Team kan bestå av upp till 11 spelare.**

☐ Sant

☐ Falskt



**Klassen Innings vet inte vad poängen är (scored runs), men vet hur den kan ta reda på det.**

☐ Sant

☐ Falskt



**11 spelare (Player) är inblandade i varje Innings.**

☐ Sant

☐ Falskt

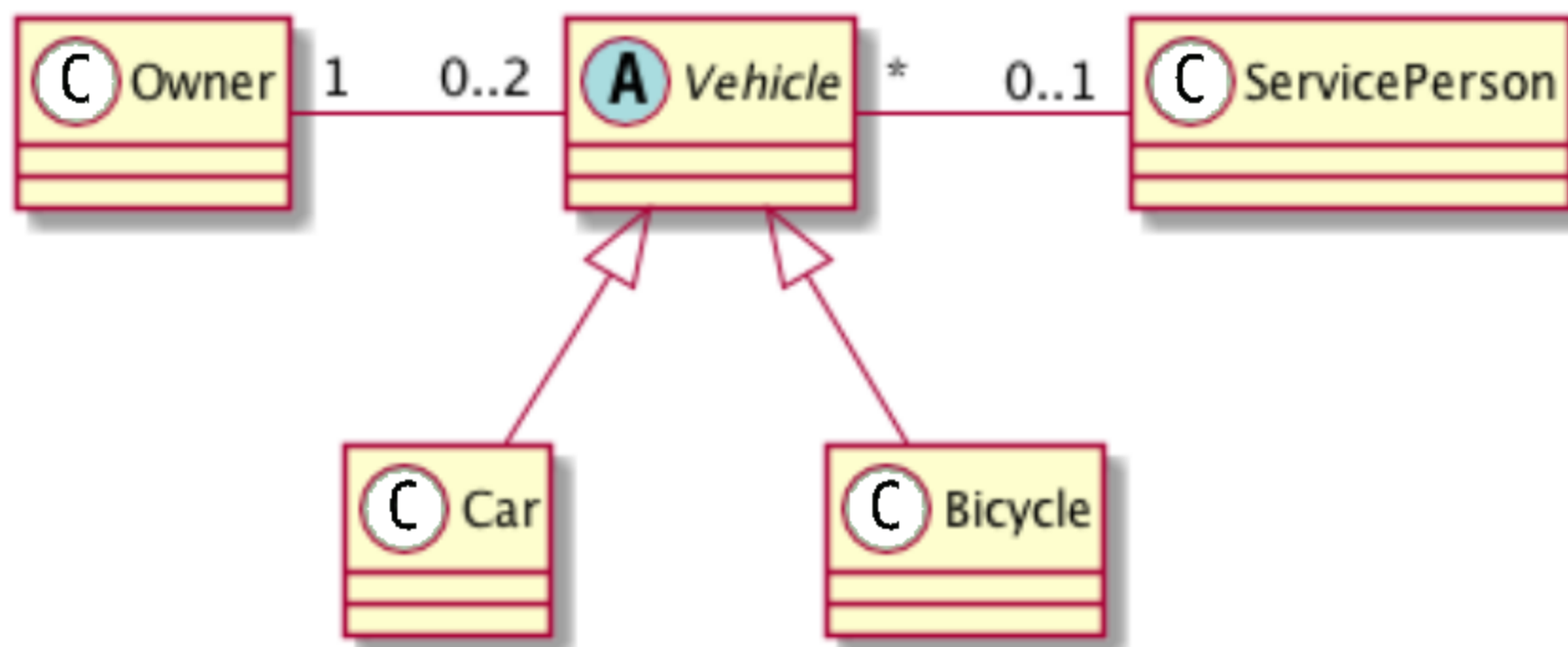


---

Totalpoäng: 8

### 3 Relationer mellan Klasser

Betrakta följande relationer mellan klasser:



Notera att detta diagram fokuserar på relationerna mellan klasserna, och går inte in på detaljer i vilka metoder eller attribut som finns.

För varje påstående nedan, markera om relationerna mellan klasserna stödjer påståendet (sant) eller inte stödjer påståendet (falskt) (+1 för rätt svar, ingen förändring för fel svar).

**astrid:Owner äger c1:Car och en b1:Bicycle**

☐ Sant



☐ Falskt

**xerxes:ServicePerson servar bara b1:Bicycle**

☐ Sant



☐ Falskt

**bengt:Owner äger c2:Car. Han servar bilen själv.**

☐ Sant



☐ Falskt

**cecilia:Owner har en boat:Vehicle**

☐ Sant

☐ Falskt



**david:Owner och ellen:Owner äger gemensamt c3:Car**

- ☐ Sant
- ☐ Falskt



**barbara:ServicePerson servar alla objekt av typen :Car som inte servas av någon annan**

- ☐ Sant
- ☐ Falskt



**fredrik:Owner äger c4:Car, c5:Car, b2:Bicycle och b3:Bicycle**

- ☐ Sant
- ☐ Falskt



**greta:Owner servar sin egen b4:Bicycle, men ibland måste hon ta hjälp av hans:ServicePerson**

- ☐ Sant
- ☐ Falskt



**irene:ServicePerson brukar ta hjälp av john:ServicePerson för att serva c6:Car**

- ☐ Sant
- ☐ Falskt



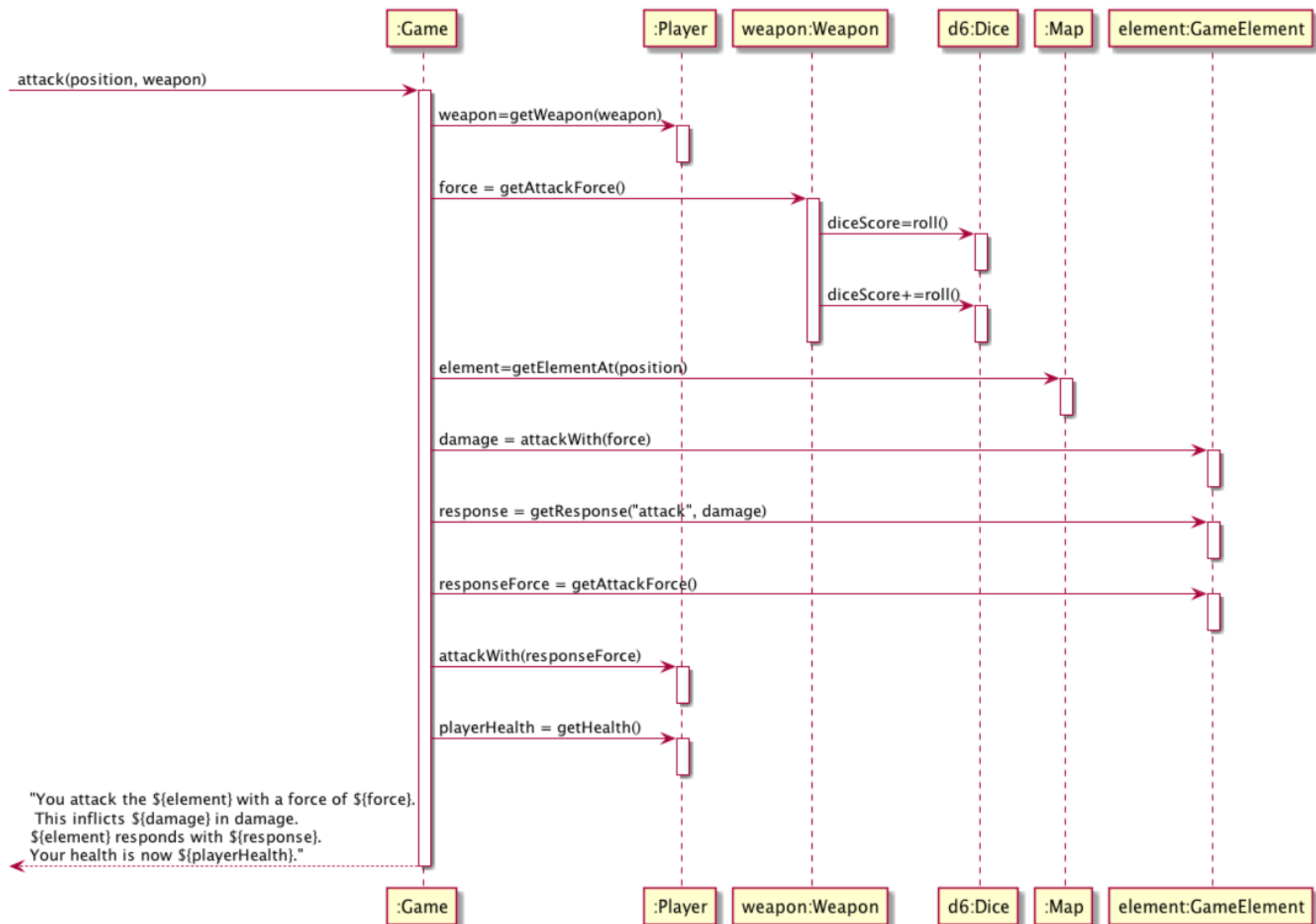
---

Totalpoäng: 9



## 4 \*COPY\_OF\* Interaktionsdiagram

Betrakta följande interaktionsdiagram:



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

**:Player** har metoderna `"getWeapon()"`, `"attackWith()"`, och `"getHealth()"`

☐ Sant

☐ Falskt

**:GameElement** har metoderna `"attackWith()"`, `"getResponse()"`, och `"getAttackForce()"`

☐ Sant

☐ Falskt

**:Game** är en Controller

☐ Sant

☐ Falskt

**:Map är en Polymorfism**

- ☐ Sant
- ☐ Falskt



**Man skulle få bättre Coupling om :Player själv hanterade sitt weapon:Weapon i stället för att ge det till :Game**

- ☐ Sant
- ☐ Falskt



**:Player är en :GameElement**

- ☐ Sant
- ☐ Falskt



**Om :Player inte har vapnet man försöker använda så kommer :Player returnera null, och :Game avbryter metoden "attack()" i förtid.**

- ☐ Sant
- ☐ Falskt



**:Player har bara ett objekt av typen :Weapon, och det ligger lagrat i attributet "weapon" hos :Player.**

- ☐ Sant
- ☐ Falskt




---

Totalpoäng: 8



## 5 Design Patterns I

Markera om följande påståenden är sanna eller falska:

(+1 för rätt svar, ingen förändring för fel svar)

**I Strategy Pattern har en klass rollen «context», 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.**

☐ Sant



☐ Falskt

**Observer Pattern består av Observers som regelbundet letar efter förändringar i klasser av typen Observable**

☐ Sant

☐ Falskt



**När det är dags att byta strategi i Strategy Pattern är varje konkret strategi ansvarig för att berätta för «Context»-klassen vilken annan konkret strategi som det är dags att byta till.**

☐ Sant

☐ Falskt



**Singleton använder sig av Polymorfism**

☐ Sant

☐ Falskt



**Ett State pattern har en klass med rollen «context», en «abstract State»-klass, och en klass per tillstånd.**

☐ Sant



☐ Falskt

**I State pattern är det klassen med rollen «abstract state» som ansvarar för vilket tillstånd man skall byta till.**

☐ Sant

☐ Falskt



---

Totalpoäng: 6

## 6 Design Patterns II

Välj rätt designmönster ur listan för varje påstående:  
(+1 för rätt svar, ingen förändring för fel svar)

Om klassen Warehouse vill berätta för olika delar av resten av systemet att lagret är uppdaterat

kan man använda Designmönstret:  (Factory, State, Strategy, **Observer**).

Ett system har olika regler för hur man skall beställa varor, till exempel kan man "beställa via mail", "beställa via ett REST-api", "beställa via ett eget protokoll", eller "beställa genom att be användaren ringa". För att implementera dessa olika sätt att beställa kan man använda

designmönstret:  (**Strategy**, State, Factory, Observer).

En viss vara kan vara "slut", "tillgänglig", "förbokad", eller "såld". För att implementera detta kan

man använda designmönstret:  (Strategy, **State**, Factory, Observer)

---

Totalpoäng: 3

## 7 GRASP Patterns

Markera om följande påståenden är sanna eller falska:

(+1 för rätt svar, ingen förändring för fel svar)

**Enligt High Cohesion skall varje klass göra så mycket som möjligt**

☐ Sant

☐ Falskt



**Enligt Low Coupling skall man se till att bara klasser så långt ner som möjligt i klassdiagrammet är kopplade**

☐ Sant

☐ Falskt



**En klass kan vara både en Information Expert och en Controller**

☐ Sant

☐ Falskt



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

☐ Sant

☐ Falskt



**En Controller kan anropa Information Experts**

☐ Sant

☐ Falskt



**Controller kräver Polymorfism för att fungera**

☐ Sant

☐ Falskt



---

Totalpoäng: 6

## i Betygsgränser

Betygsgränserna för denna tenta är:

Betyg	Procent	Poäng
MAX	100%	48
A	90%	43
B	80%	38
C	70%	33
D	65%	31
E	60%	29

**Lycka till!**