FINAL GENER

Assignatura: **Disseny de Software**

Data: 15 de Gener de 2019 Curs: **2018/2019**

UNIVERSITAT DE BARCELONA

Nom:_____

DNI:_____

Aula: _____ **Fila:**_____**Columna:**_____

- Per marcar una resposta vàlida poseu ×
- Per rectificar una resposta ja marcada poseu un ◯ sobre la × i marqueu la correcta amb una ×
- **Puntuació**: No contestada: 0 punts. Correcta: 1 punt. Incorrecta: -0.1 punts.

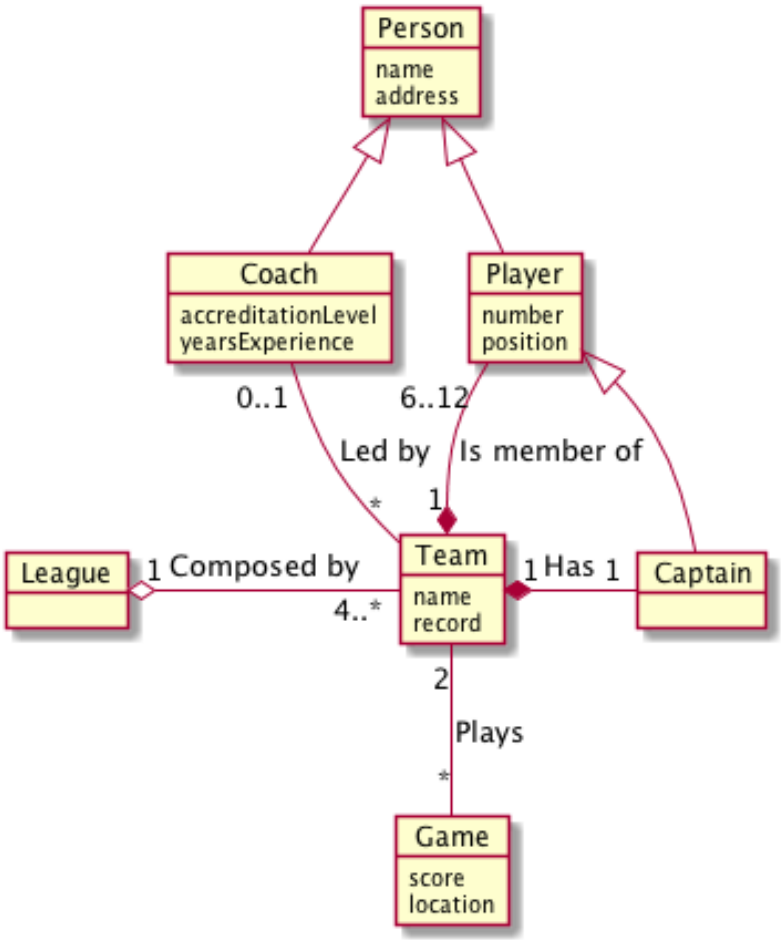
Test (40 punts): Temps ESTIMAT 45 min.

1. El disseny conduït per tests (Test-Driven Development) utilitzant Concordion i seguint el patró Model-Vista-Controlador:
   - **a.** Serveix per a desenvolupar el Model de forma incremental, basant-se en una història d'usuari a cada iteració.
   - **b.** No cal refactoritzar ja que Concordion ja se n'ocupa de fer-ho a cada increment.
   - **c.** Serveix per a testejar les funcionalitats del Controlador i el Model, basant-se en un criteri d'acceptació a cada iteració.
   - **d.** Des de la part de Concordion, és necessari definir a cada iteració un nou fitxer html amb un nou criteri d'acceptació.

2. En relació al Model-Vista-Controlador és CERT que:
   - **a.** El Controlador controla la lògica de la Vista per activar o desactivar botons
   - **b.** El Controlador s'enregistra com observador del Model, usant el patró Observer, per a monitoritzar els seus possibles canvis i així poder-los mostrar en la Vista.
   - **c.** La Vista només es pot comunicar amb el Model via el Controlador, per així no acoblar la Vista amb el Model.
   - **d.** El Controlador es dissenya seguint el patró Facade per a donar l'accés al Model.

3. En relació als Casos d'Ús es pot afirmar que:
   - **a.** S'inclouen tots els requeriments FURPS+ de l'aplicació.
   - **b.** Defineixen el diàleg entre els actors i el sistema per a poder definir i acotar les històries d'usuari de l'aplicació a desenvolupar.
   - **c.** Cada cas d'ús dóna lloc a una única història d'usuari.
   - **d.** Els actors que es defineixen en el cas d´ús no han de sortir en el Model de Domini, ja que son els que inicien les peticions al sistema però no formen part del sistema.

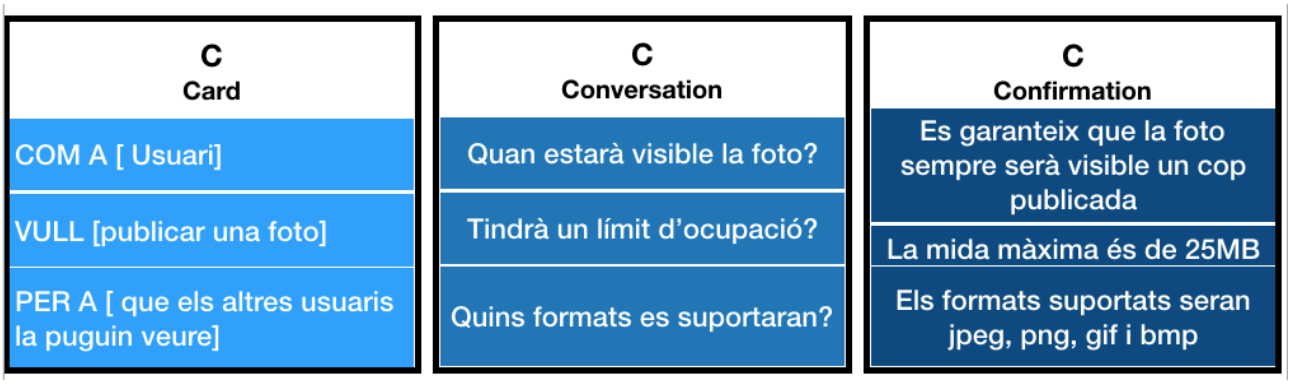
Donat el següent model de domini que modela una lliga de hoquei, contesta les preguntes següents:



4. Quina de les següents afirmacions és **FALSA?**:
   - **a.** L'atribut *location* de la classe *Game* hauria de ser una classe a part, ja que és un lloc físic i s'evitarien repeticions.
   - **b.** L'atribut *address* de la classe *Person* hauria de ser una classe diferent ja que el seu tipus no és elemental.
   - **c.** En aquest Model de Domini, un entrenador (*Coach*) té opció a entrenar a més d'un equip (*Team*).
   - **d.** En aquest Model de Domini, gràcies a les cardinalitats de les associacions "*Has*" i "*Is member of*", quan s'esborren els jugadors (*Players*) i el capità (*Captain*) d'un equip, s'esborra l'equip (*Team*)

5. Si es volgués saber el nombre de gols que ha marcat un jugador en un partit, quina de les següents opcions ho modelaria millor en el Model de Domini anterior?
   - **a.** S'hauria d'afegir una associació entre *Player* i *Game* (de cardinalitat "*"--"*") i un atribut a *Player* que indiqués els partits on ha marcat.
   - **b.** S'hauria de posar a la classe *Game* una llista de Jugadors que han marcat gols i l'associació entre *Player* i *Game* (de cardinalitat "*"--"*").
   - **c.** S'hauria d'afegir una classe anomenada "*Gol*" que contingués un atribut amb el número de gols marcats per un jugador en un partit i relacionar-la amb *Game* i *Player* amb cardinalitats (*Game* "1"--"*" *Gol* i *Player*"1"--"*" *Gol*)
   - **d.** Les tres opcions anteriors són correctes en el Model de Domini.

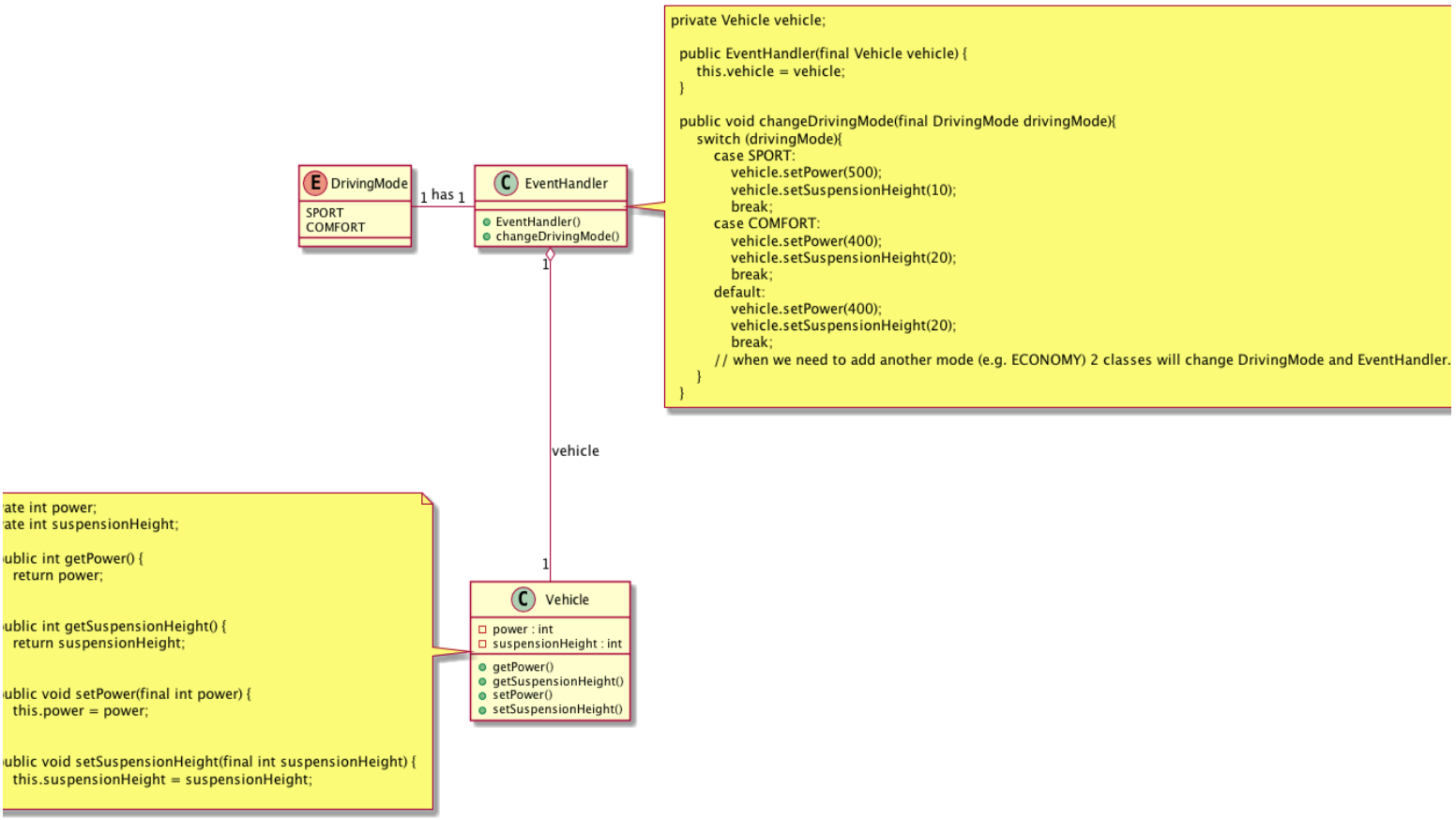
6. Un dissenyador, en la fase de requeriments ha construït aquesta Card-Conversation-Confirmation:



Quina de les següents afirmacions és **FALSA**?

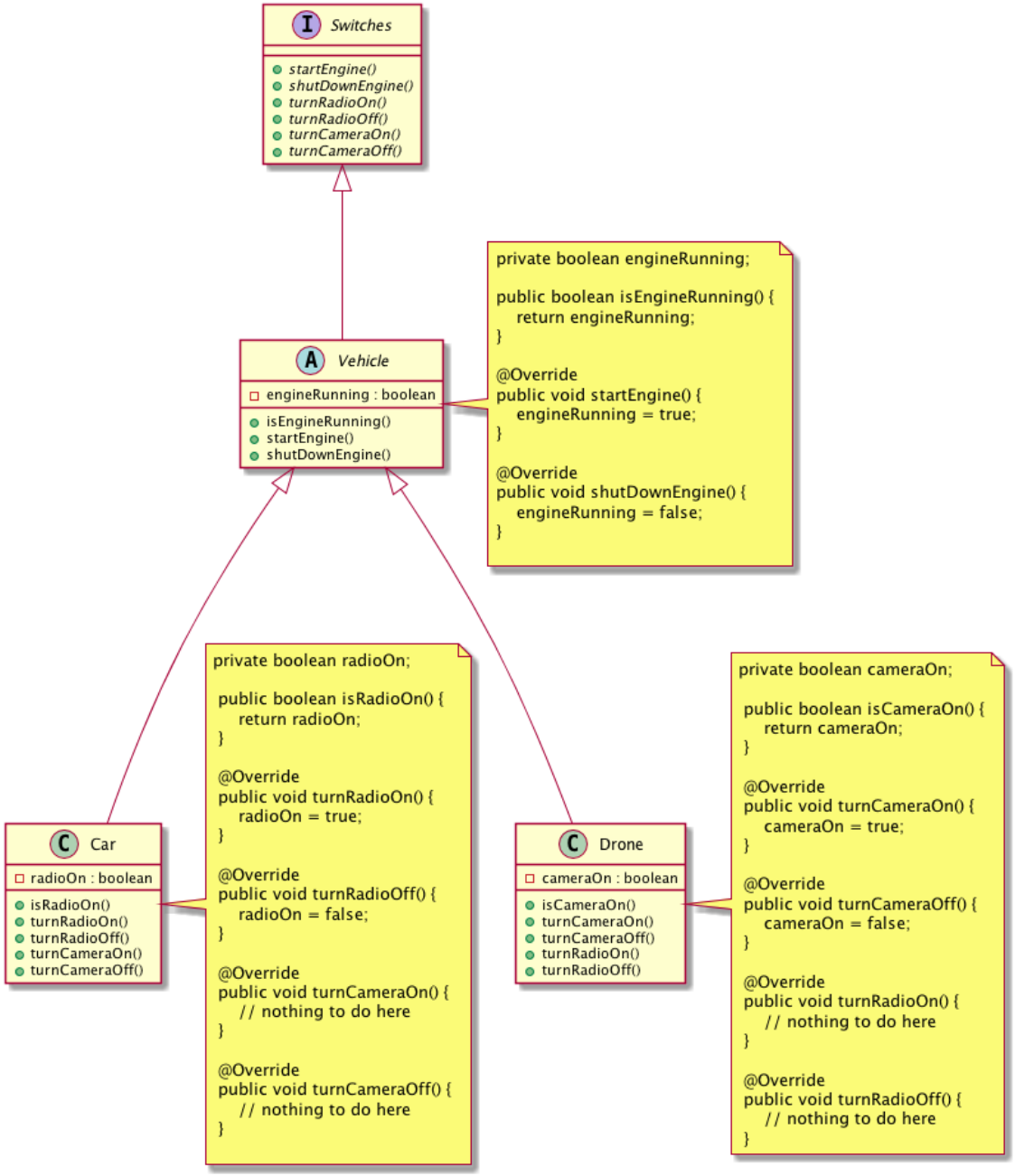
a. La carta CARD és directament la història d'usuari.
b. Segons la carta CONFIRMATION, el dos darrers criteris d'acceptació seran els que podran donar lloc a test d'acceptació funcionals.
c. Els criteris I.N.V.E.S.T. només s'apliquen entre històries d'usuari i no en criteris d'acceptació i , per tant, no es poden avaluar en aquesta història d'usuari.
d. Un possible test d'acceptació que s'extreu de la carta Confirmation és "EN EL CAS QUE l'usuari entri en el seu perfil QUAN l'usuari hagi publicat la foto anteriorment EL SISTEMA garanteix que sempre serà visible pels altres usuaris".

7. Seguint els principis S.O.L.I.D., un dissenyador ha fet el següent diagrama de classes en el què vol dissenyar diferents tipus de conduccions d'un vehicle. Quin/s principi/s de disseny vulnera/en?



a. Single Responsability Principle
b. Open-Closed Principle
c. Dependency Inversion Principle
d. No es vulnera cap principi S.O.L.I.D

8. Seguint els principis S.O.L.I.D., un dissenyador ha fet el següent diagrama de classes en el què volen modelar diferents accions sobre Vehicles. Quin/s principi/s de disseny vulnera?



a. Interface Segretation Principle
b. Liskov Substitution Principle
c. Dependency Inversion Principle
d. No se vulnera cap principi S.O.L.I.D

9. Segons els patrons GRASP es pot afirmar que:
a. S'aplica el patró Creador per a donar a la classe B la responsabilitat de crear una instància d'A si B agrega objectes de classe A
b. S'aplica el patró Expert per a especialitzar classes utilitzant herència.
c. És bo que la cohesió sigui baixa
d. El patró Creador s'aplica només en el cas d'utilitzar el patró Factory

10. Quina de les següents afirmacions és **CERTA**?

a. El patró Factory Method és un patró estructural.

b. En el patró Singleton cal definir un constructor privat per així evitar el problema de la *eager initialization.*

c. En el patró Strategy el comportament d'una classe o el seu algorisme pot canviar en temps d'execució.

d. El patró Factory Method permet només una única Factory concreta (Concrete Creator) que serà la que encapsularà la creació dels productes concrets.

Respostes:

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
a										
b										
c										
d										

Nom:_____

FINAL GENER

Assignatura: **Disseny de Software**

Data: 15 de Gener de 2019 Curs: **2018/2019**

UNIVERSITAT DE BARCELONA

Nom:_____

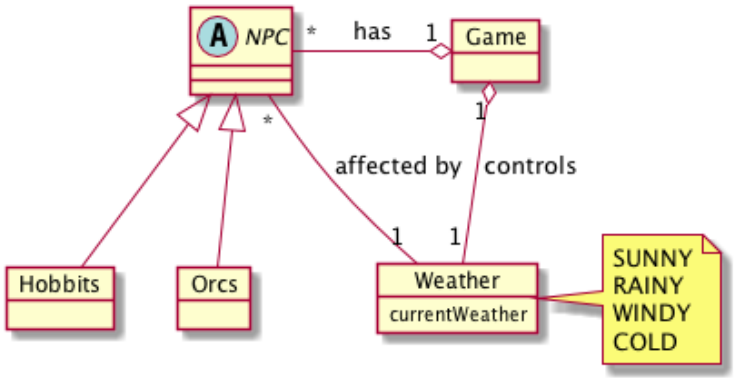
DNI:_____

Aula: _____ **Fila:**_____**Columna:**_____

Problema (60 punts): Temps estimat: 2 hores

Es vol dissenyar un joc que té dos tipus de NPCs (els orcs i els hobbits) i que té diferents condicions meteorològiques que varien durant el temps del joc. Quan aquestes condicions varien, els orcs i els hobbits reaccionen de diferents maneres. Les condicions meteorològiques poden ser de 4 tipus (SUNNY, WINDY, RAINY i COLD). en aquesta versió inicial del joc, i canvien de forma aleatòria entre elles. No obstant, es vol que en un futur, aquestes condicions meteorològiques vagin lligades al temps que fa en la ubicació del jugador en el món real i segons canviïn a la realitat, afectin als NPCs.

Davant d'aquest problema, un dissenyador de software, preveient aquest funcionament del joc, ha fet el següent Model de Domini on la classe Game, entre d'altres relacions i atributs d'altres parts del joc, té els NPCs i les condicions meteorològiques.



I proposa el següent disseny de classes, on es té un main en la classe App que crea al controlador del joc (veure el full del final de l'examen) i respon a les següents preguntes:

a) Què en pots dir dels principis S.O.L.I.D.? Raona si se'n vulnera algun/s.

S: El GameControl té mes d'una responsabilitat, ja que controlar tant el canvi del temps com la reacció dels NPCs quan canvia el temps
O: Clarament es vulnera en la classe NPCs que no es tancada a extensions ja que si s'afegeix un nou tipus de temps, caldria modificar el update
L: No es vulnera
I: No es vulnera
D: Es vulnera en tenir fixes les accions o comportaments (pero també es acceptable dir que no es vulnera)

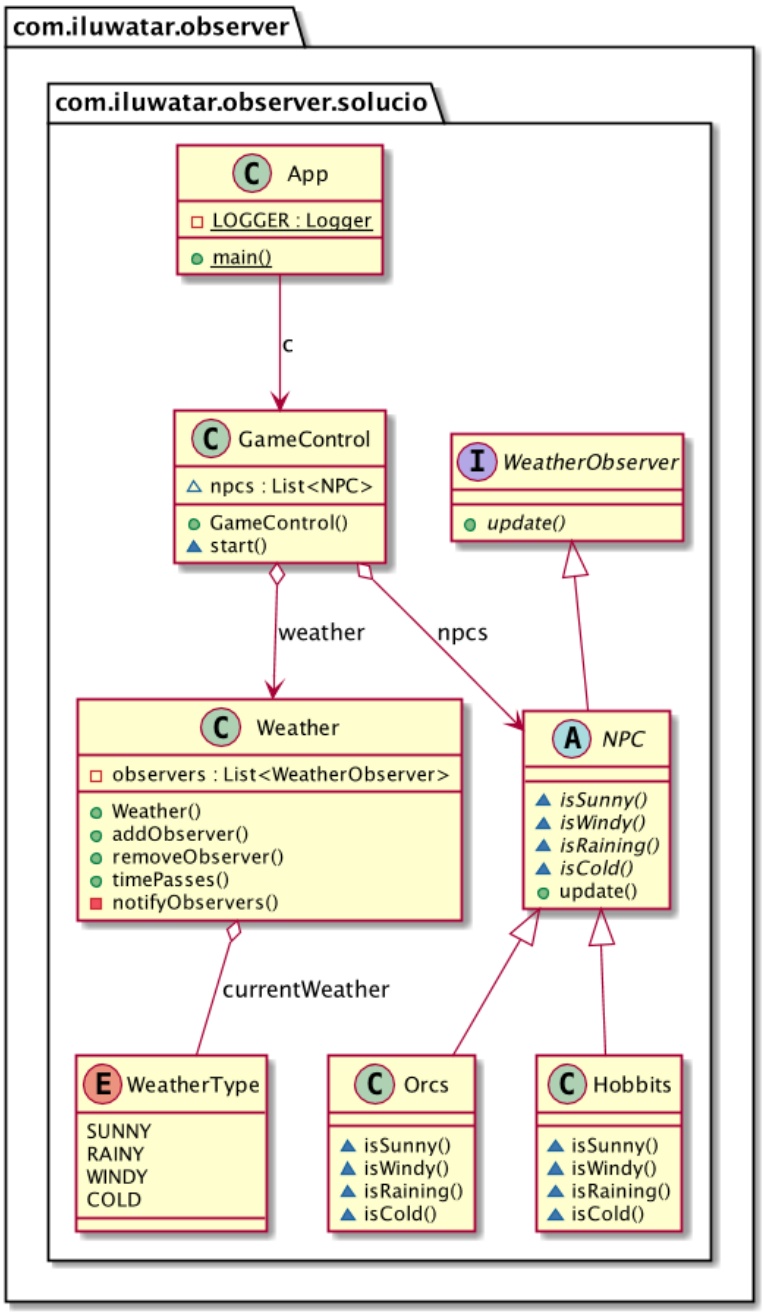
b) Com redissenyaries aquest disseny? Quin/s patró/ns de disseny faries servir? Per a contestar aquest apartat omple els apartats següents.

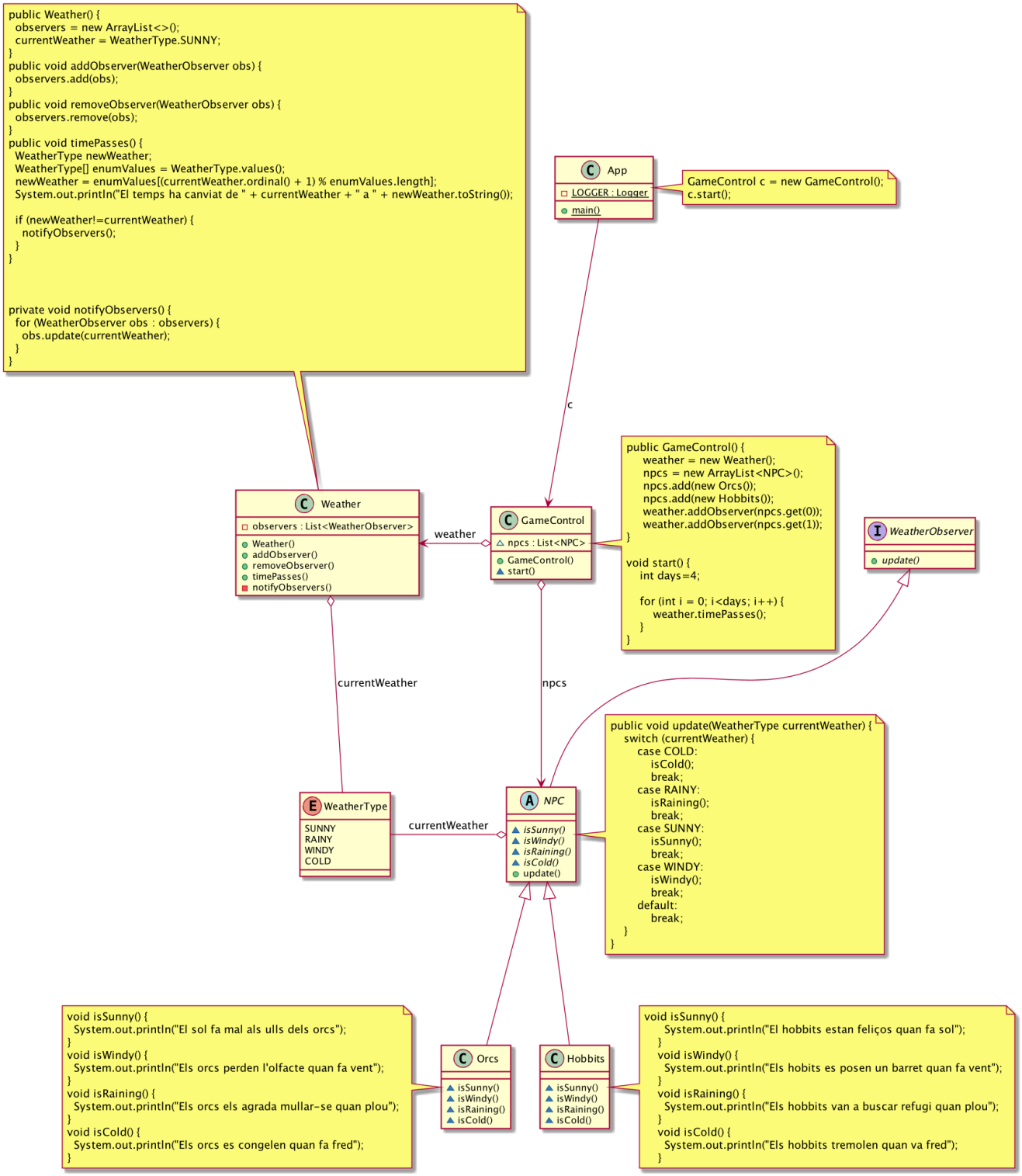
b.1. Nom del patró principal i tipus de patró: __OBSERVER (comportament) _____

b.2. Aplicació del patró (Dibuixa el diagrama de classes obtingut després d'aplicar el patró, quines classes es corresponent en el patró original i explica els detalls més rellevants del teu disseny)

El patró a aplicar és l'observer ja que qualsevol canvi en el temps ha d'estar notificat als NPCs. Així l'Observer és el NPC i el Subject és el Weather. Quan el Weather canviï, ho notificarà directament als observers. La llista d'observers estarà a la classe Weather per a poder enregistrar tots els observers a qui s'ha de notificar i fer update. El mètode update del NPC serà cridat des del notify del Weather sense passar per Game Control

SOLUCIO's Class Diagram





b.3. Anàlisi del patró aplicat en relació als principis S.O.L.I.D.

S: Ja no es vulnera, ja que el GameControl ara com a molt fa la part d'actualitzar el temps però el comportament dels NPCs s'ha delegat amb el patró observer
O: Clarament es segueix vulnerant en la classe NPCs que no es tancada a extensions ja que si s'afegeix un nou tipus de temps, caldria modificar el update
L: No es vulnera ja que totes les classes derivades de NPC implementen tot el comportament esperat per la superclasse NPC.
I: No es vulnera ja que no hi han implementacions d'interficies que no fan els mètodes que es defineixen en la interfície.

D: Es vulnera en tenir fixes les accions o comportaments (pero també es acceptable dir que no es vulnera)

b.4. El constructor i el mètode start() de la classe GameControl que mostra l'ús del patró utilitzat:

```
public GameControl() {
    weather = new Weather();
    npcs = new ArrayList<NPC>();
    npcs.add(new Orcs());
    npcs.add(new Hobbits());
    weather.addObserver(npcs.get(0));
    weather.addObserver(npcs.get(1));
}

void start() {
   int days=4;

    for (int i = 0; i<days; i++) {
       weather.timePasses();
    }
}
```

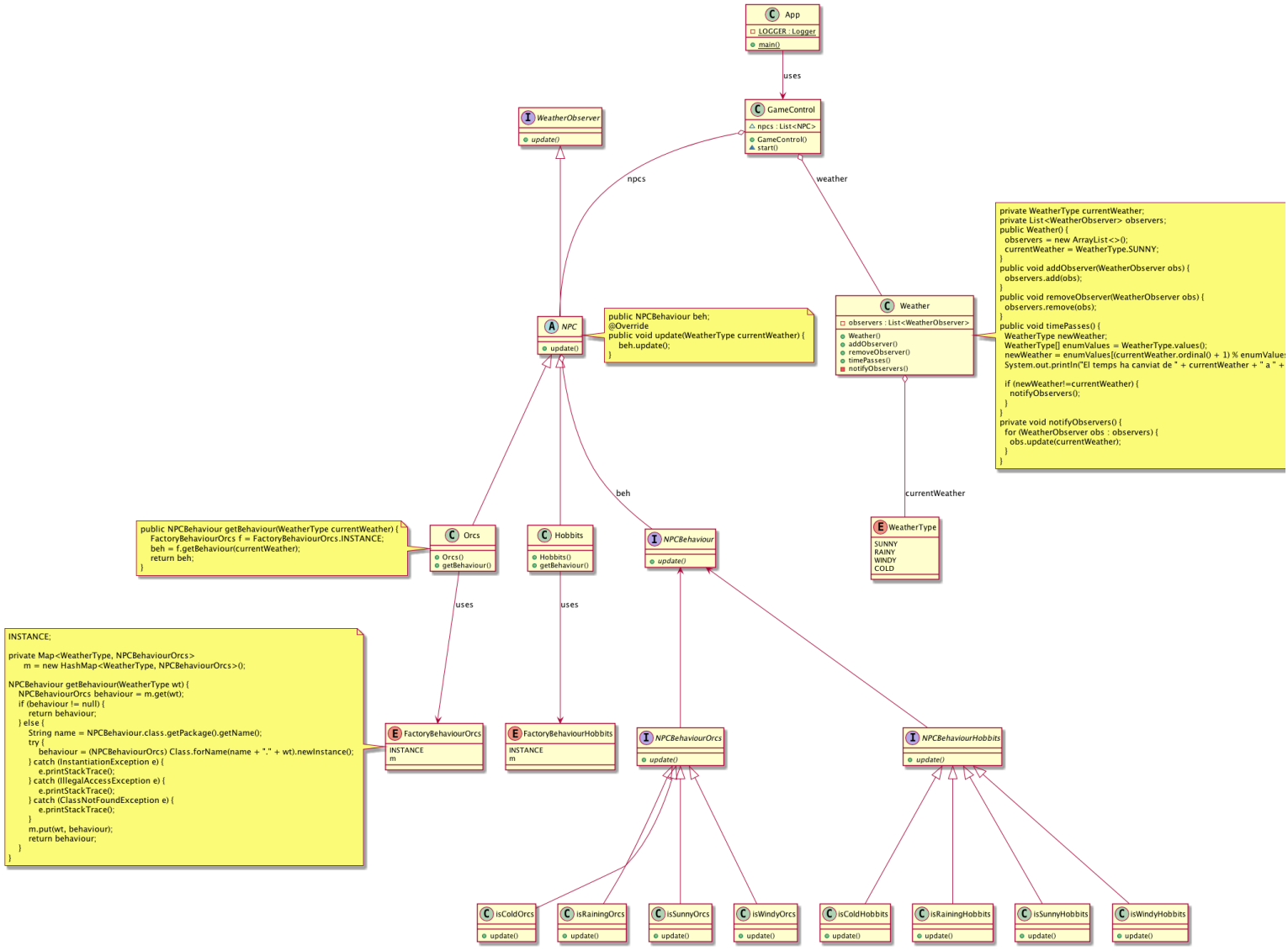
b.5. Observacions addicionals:

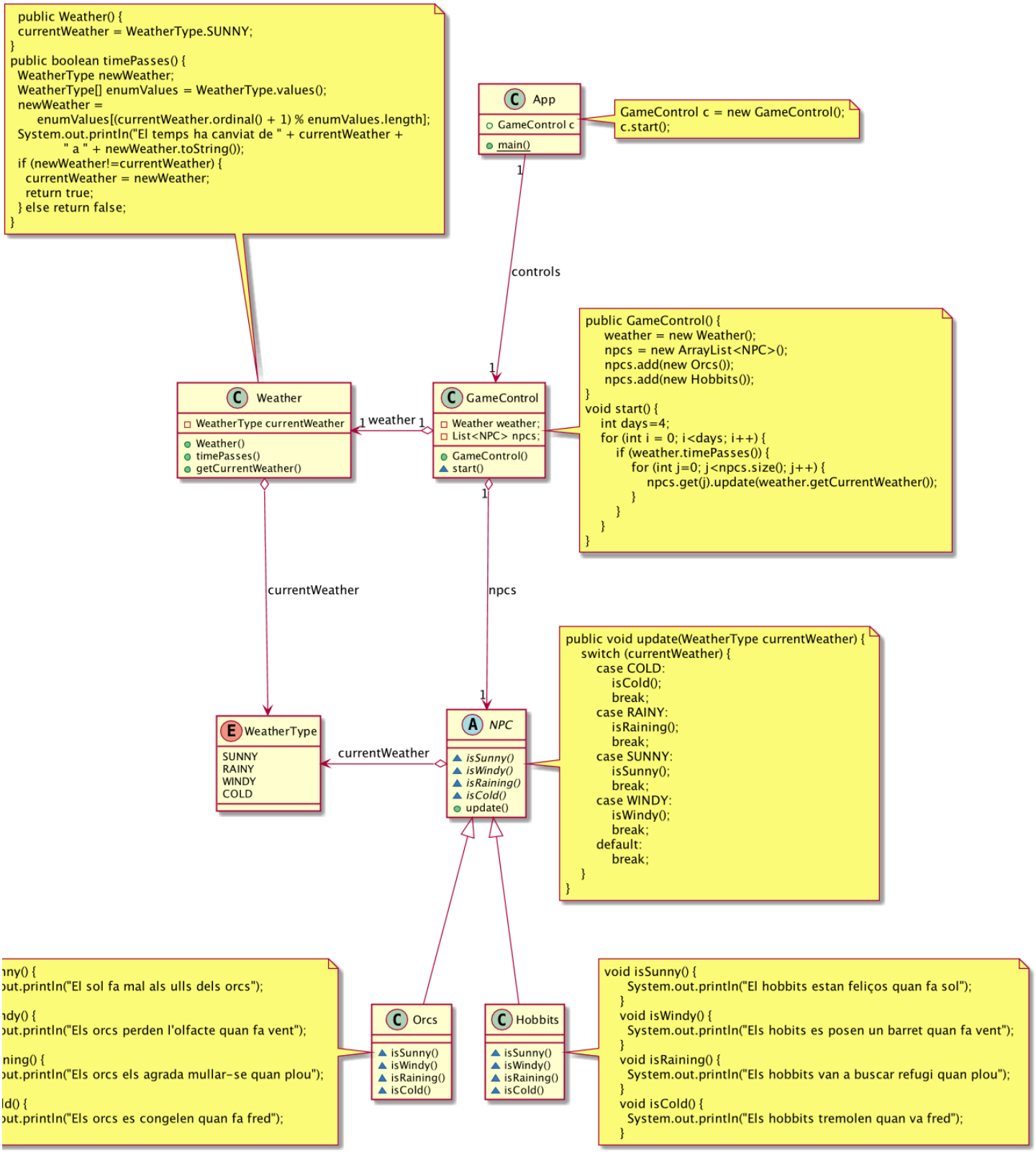
Com es fa crida l'update de cada NPC? a la classe Weather quan es detecta que el temps ha canviat:

```
public Weather() {
  observers = new ArrayList<>();
  currentWeather = WeatherType.SUNNY;
}

public void addObserver(WeatherObserver obs) {
  observers.add(obs);
}

public void removeObserver(WeatherObserver obs) {
  observers.remove(obs);
}

public void timePasses() {
  WeatherType newWeather;
  WeatherType[] enumValues = WeatherType.values();
  newWeather = enumValues[(currentWeather.ordinal() + 1) % enumValues.length];
  System.out.println("El temps ha canviat de " + currentWeather + " a " + newWeather.toString());

  if (newWeather!=currentWeather) {
    notifyObservers();
  }
}

private void notifyObservers() {
  for (WeatherObserver obs : observers) {
    obs.update(currentWeather);
  }
}
```

c) En una segona versió del joc es vol afegir un nou tipus de temps SNOWY per a modelar que els hobbits quan neva fan ninots de neu i els orcs fan batalles de neu. Com afectaria al teu disseny? Quin/s patró podries usar per canviar dinàmicament el comportament dels NPCs? Raona la teva resposta en 10 línies com a màxim.

Es podria fer un patró Strategy per poder instanciar els comportaments dels NPCs de forma dinàmica lligat amb un Factory Method que encapsularia la creació dels diferents comportaments.
Per a implementar el Strategy es faria fet una derivada de behaviour per a cada NPC.




public Weather() {
currentWeather = WeatherType.SUNNY;
}
public boolean timePasses() {
WeatherType newWeather;
WeatherType[] enumValues = WeatherType.values();
newWeather =
enumValues[(currentWeather.ordinal() + 1) % enumValues.length];
System.out.println("El temps ha canviat de " + currentWeather +
" a " + newWeather.toString());
if (newWeather!=currentWeather) {
currentWeather = newWeather;
return true;
} else return false;
}
App
GameControl c
main()
GameControl c = new GameControl();
c.start();
1
controls
1
Weather
WeatherType currentWeather
Weather()
timePasses()
getCurrentWeather()
1 weather 1
GameControl
Weather weather;
List<NPC> npcs;
GameControl()
start()
public GameControl() {
weather = new Weather();
npcs = new ArrayList<NPC>();
npcs.add(new Orcs());
npcs.add(new Hobbits());
}
void start() {
int days=4;
for (int i = 0; i<days; i++) {
if (weather.timePasses()) {
for (int j=0; j<npcs.size(); j++) {
npcs.get(j).update(weather.getCurrentWeather());
}
}
}
}
1
currentWeather
npcs
1
public void update(WeatherType currentWeather) {
switch (currentWeather) {
case COLD:
isCold();
break;
case RAINY:
isRaining();
break;
case SUNNY:
isSunny();
break;
case WINDY:
isWindy();
break;
default:
break;
}
}
E WeatherType
SUNNY
RAINY
WINDY
COLD
currentWeather
A NPC
isSunny()
isWindy()
isRaining()
isCold()
update()
nny() {
ut.println("El sol fa mal als ulls dels orcs");
ndy() {
ut.println("Els orcs perden l'olfacte quan fa vent");
ning() {
ut.println("Els orcs els agrada mullar-se quan plou");
ld() {
ut.println("Els orcs es congelen quan fa fred");
C Orcs
isSunny()
isWindy()
isRaining()
isCold()
C Hobbits
isSunny()
isWindy()
isRaining()
isCold()
void isSunny() {
System.out.println("El hobbits estan feliços quan fa sol");
}
void isWindy() {
System.out.println("Els hobits es posen un barret quan fa vent");
}
void isRaining() {
System.out.println("Els hobbits van a buscar refugi quan plou");
}
void isCold() {
System.out.println("Els hobbits tremolen quan va fred");
}