FINAL GENER

Assignatura: **Disseny de Software**

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

UNIVERSITAT DE BARCELONA

Nom:_____

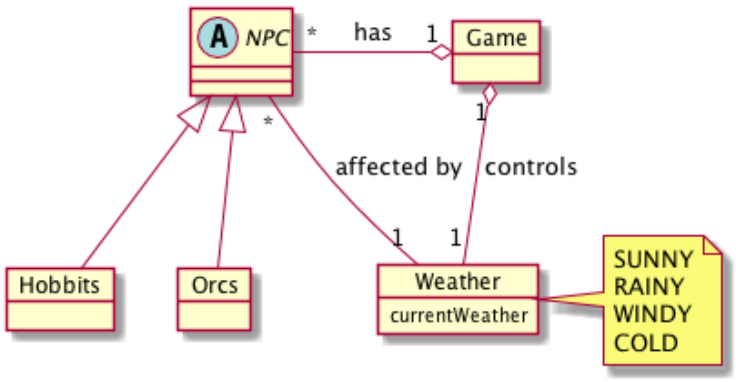
DNI:_____

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, 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)

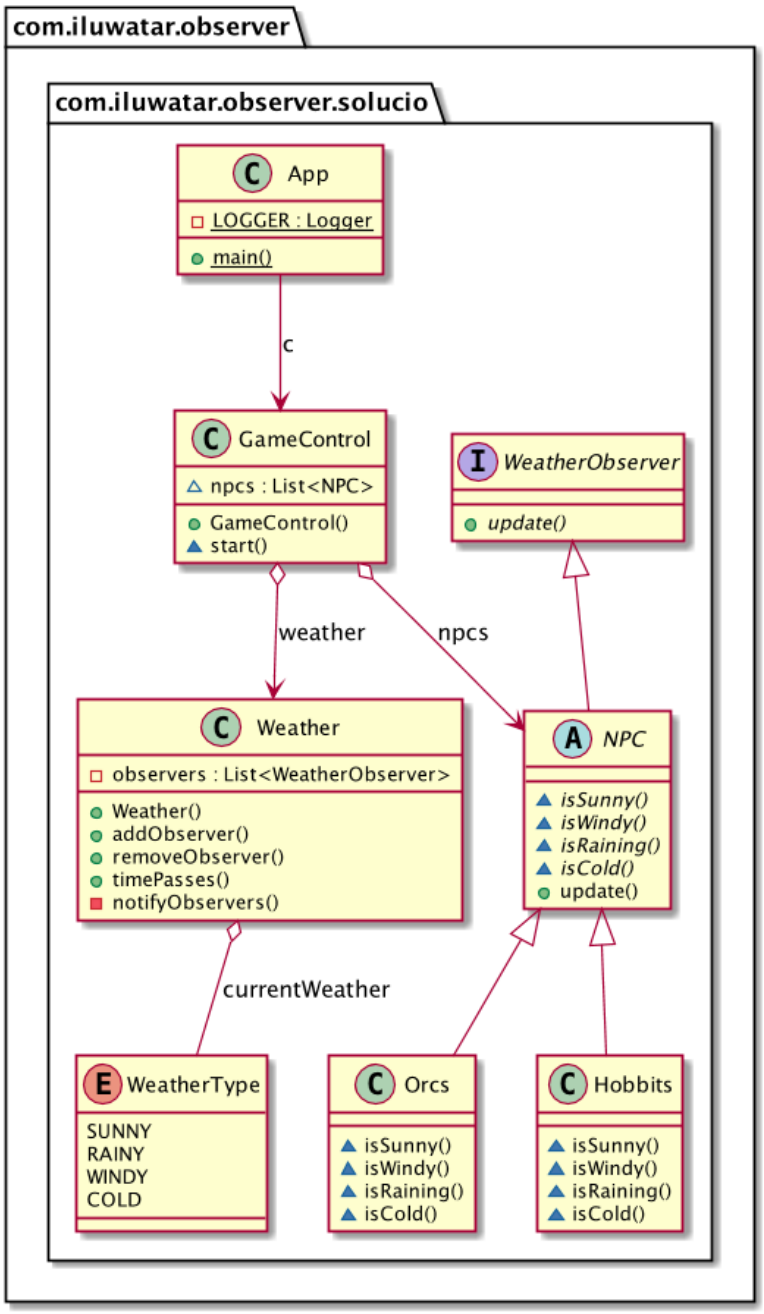
b) Com redissenyaries aquest disseny? Quin/s patró/ns de disseny faries servir? Per a contestar aquest apartat omple els apartats següents.

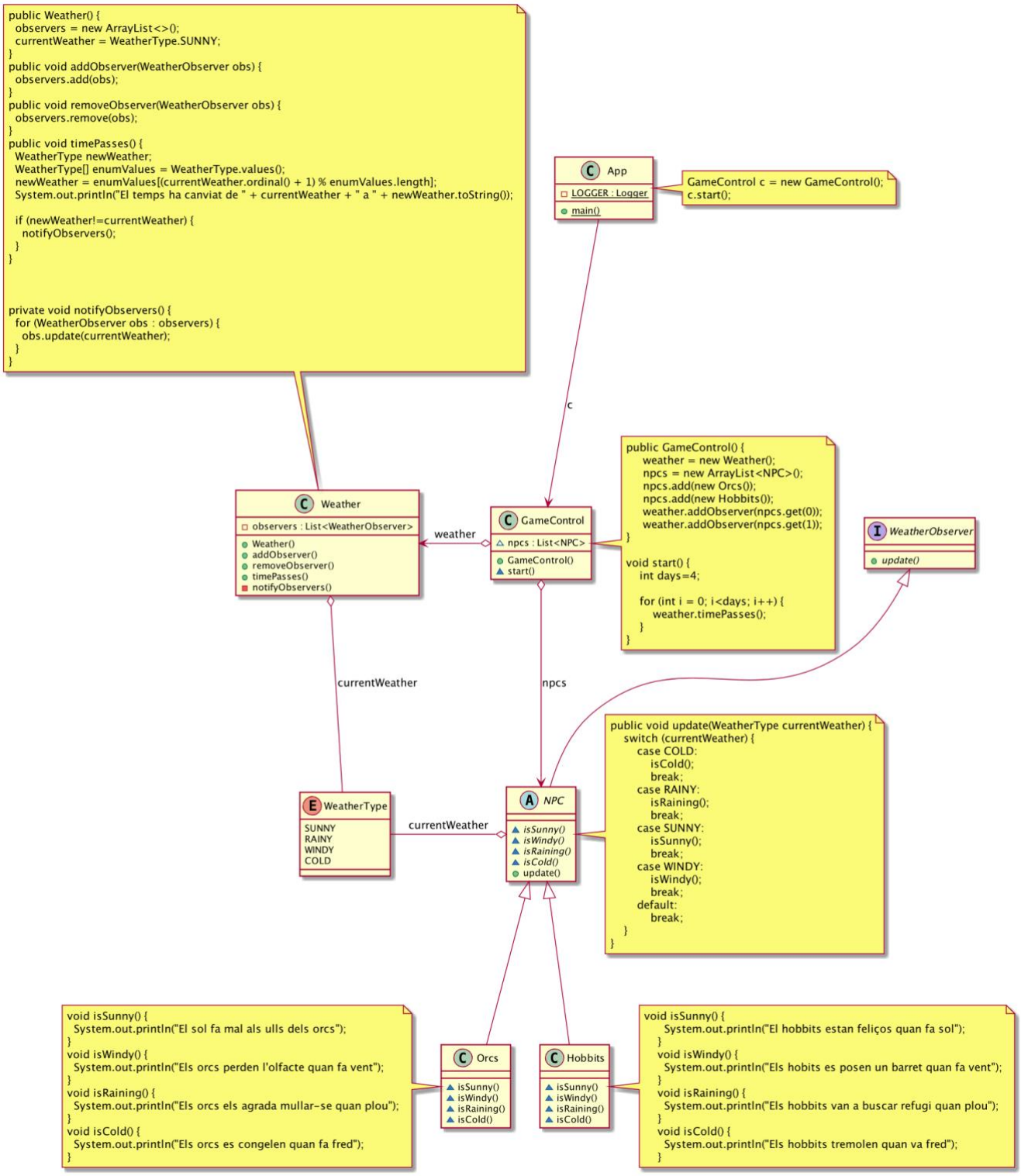
b.1. Nom del patró principal i tipus de patró: ___OBSERVER (comportament) _____

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:

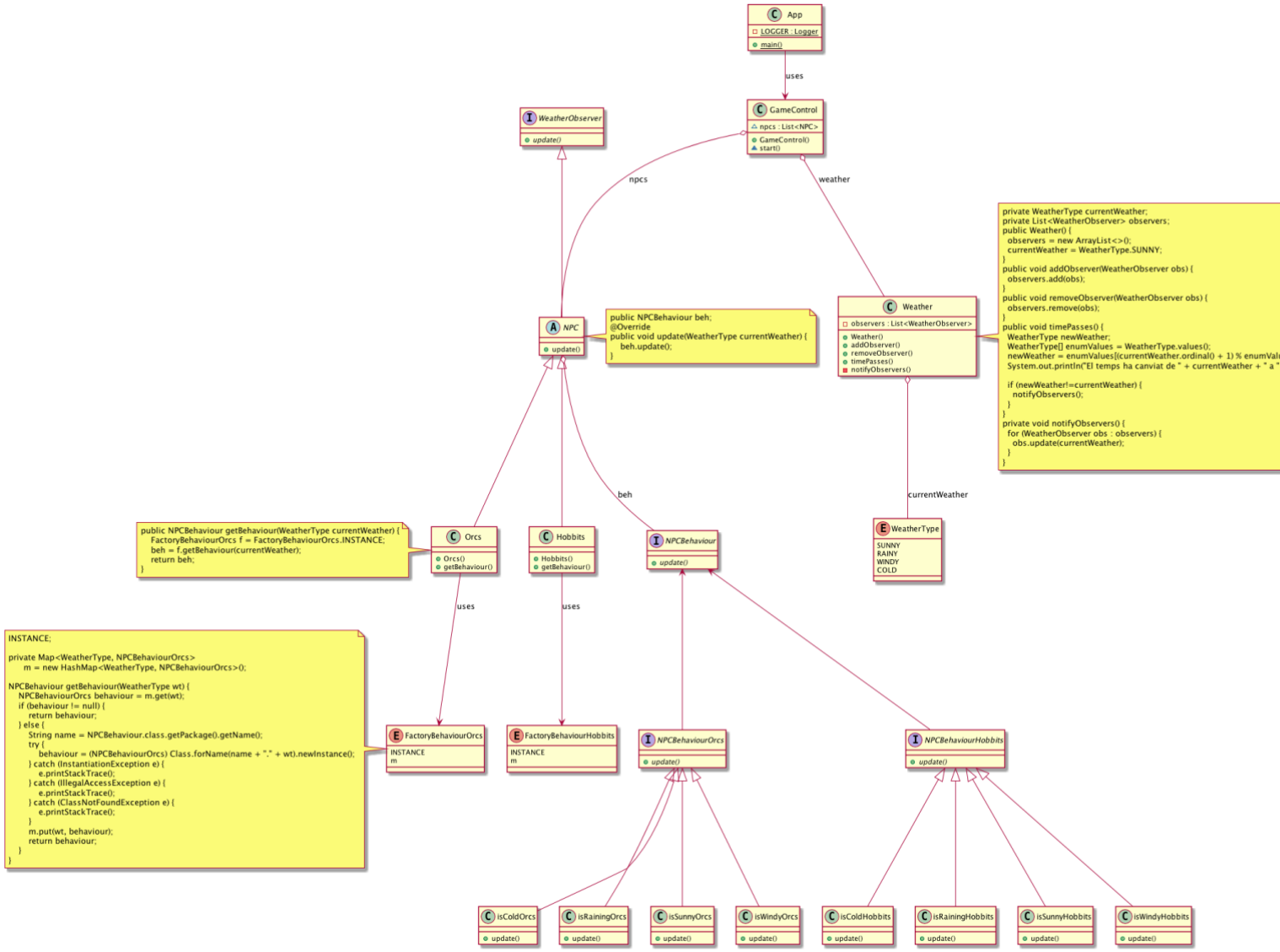
Com es fa crida l'update de cada NPC? a la classe Weather quan es detecta que el temps ha canviat:

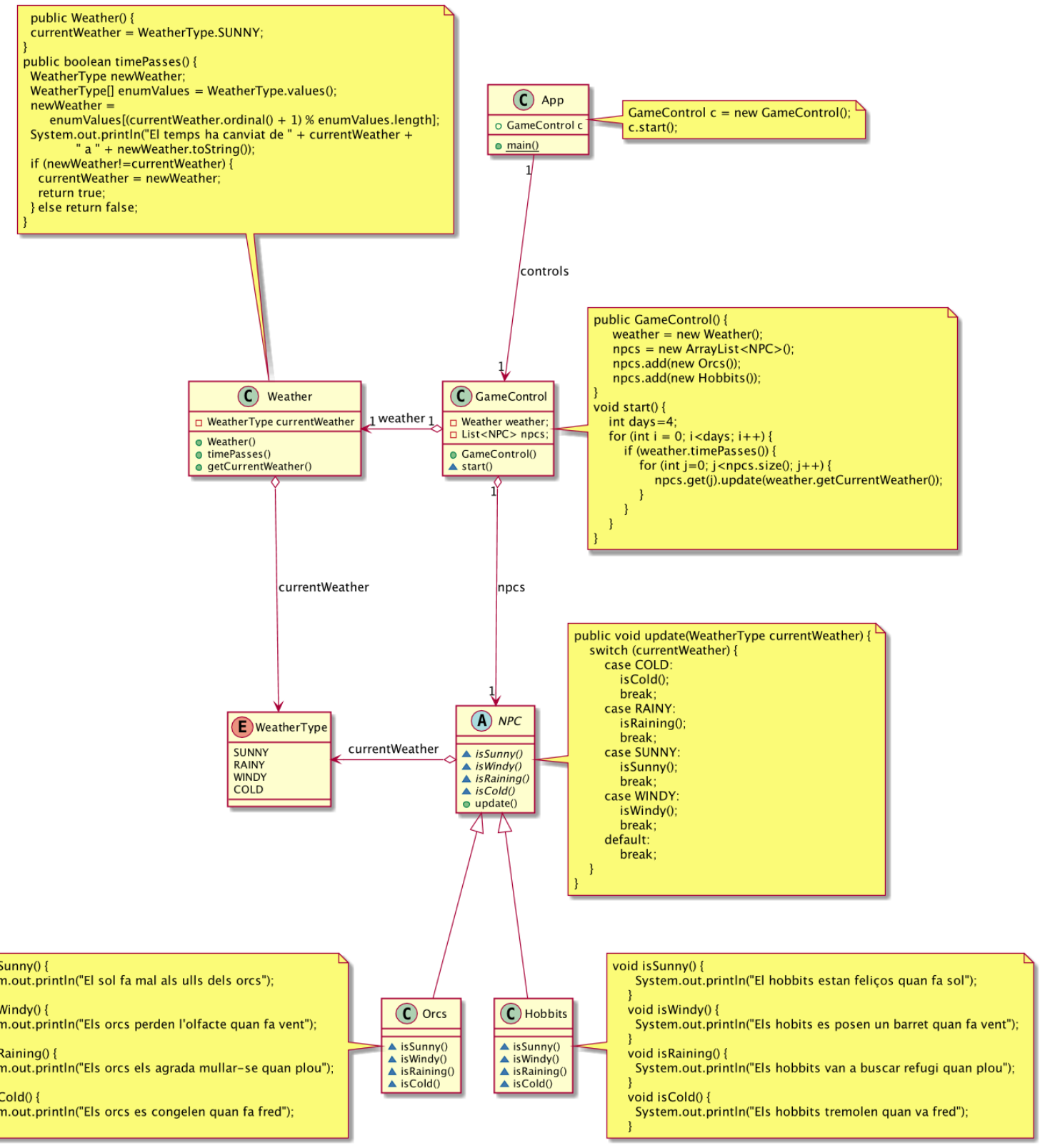
```
public Weather() {
  observers = new ArrayList<>();
  currentWeather = WeatherType.SUNNY;
}

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.


App
LOGGER : Logger
main()
uses
GameControl
npcs : List<NPC>
GameControl()
start()
WeatherObserver
update()
npcs
weather
NPC
update()
public NPCBehaviour beh;
@Override
public void update(WeatherType currentWeather) {
beh.update();
}
Weather
observers : List<WeatherObserver>
Weather()
addObserver()
removeObserver()
timePasses()
notifyObservers()
private WeatherType currentWeather;
private List<WeatherObserver> observers;
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.len
System.out.println("El temps ha canviat de " + currentWeather + " a " + new
if (newWeather!=currentWeather) {
notifyObservers();
}
}
private void notifyObservers() {
for (WeatherObserver obs : observers) {
obs.update(currentWeather);
}
}
beh
currentWeather
public NPCBehaviour getBehaviour(WeatherType currentWeather) {
FactoryBehaviourOrcs f = FactoryBehaviourOrcs.INSTANCE;
beh = f.getBehaviour(currentWeather);
return beh;
}
Orcs
Orcs()
getBehaviour()
Hobbits
Hobbits()
getBehaviour()
NPCBehaviour
update()
WeatherType
SUNNY
RAINY
WINDY
COLD
uses
uses
INSTANCE;
private Map<WeatherType, NPCBehaviourOrcs>
m = new HashMap<WeatherType, NPCBehaviourOrcs>();
NPCBehaviour getBehaviour(WeatherType wt) {
NPCBehaviourOrcs behaviour = m.get(wt);
if (behaviour != null) {
return behaviour;
} else {
String name = NPCBehaviour.class.getPackage().getName();
try {
behaviour = (NPCBehaviourOrcs) Class.forName(name + "." + wt).newInstance();
} catch (InstantiationException e) {
e.printStackTrace();
} catch (IllegalAccessException e) {
e.printStackTrace();
} catch (ClassNotFoundException e) {
e.printStackTrace();
}
m.put(wt, behaviour);
return behaviour;
}
}
FactoryBehaviourOrcs
INSTANCE
m
FactoryBehaviourHobbits
INSTANCE
m
NPCBehaviourOrcs
update()
NPCBehaviourHobbits
update()
isColdOrcs
update()
isRainingOrcs
update()
isSunnyOrcs
update()
isWindyOrcs
update()
isColdHobbits
update()
isRainingHobbits
update()
isSunnyHobbits
update()
isWindyHobbits
update()


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()
Sunny() {
m.out.println("El sol fa mal als ulls dels orcs");
Windy() {
m.out.println("Els orcs perden l'olfacte quan fa vent");
Raining() {
m.out.println("Els orcs els agrada mullar-se quan plou");
Cold() {
m.out.println("Els orcs es congelen quan fa fred");
Orcs
isSunny()
isWindy()
isRaining()
isCold()
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");
}