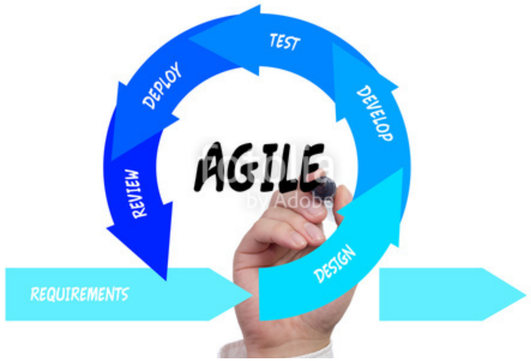


Classe Teoria Setmana 15: Disseny: Patrons de Disseny

Anna Puig

Enginyeria Informàtica

Facultat de Matemàtiques i Informàtica,

Universitat de Barcelona

Curs 2021/22

UNIVERSITAT DE BARCELONA

Com serà l'examen Final? 11 de gener 2022, 15:00h
Aules B3 i B5

Teoria:

2 ó 3 problemes: Model de Domini, SOLID/GRASP i Patrons de Disseny (Exemple)

Temps: 2 hores - 2:30h

Què es pot portar imprès ?: Resum de Patrons de Disseny

Teoria =
max (**Parcial** x 35% + **Final** x 65% + [**Problemes** x 10%],
Final x 100%)

Nota Final = Teoria + Pràctiques

AVALUACIÓ CONTÍNUA

L'avaluació es separa en dos blocs

- **Teoria i problemes**: (5,5 punts) **Nota sobre 10 >= 4**
  - Examen **Parcial**
  - Examen **Final**
  - **Problemes** durant el curs
- **Pràctiques**: (4,5 punts): **Nota sobre 10 >= 4**
  - Pràctica 1 (0,5 punt)
  - Pràctica 2 (1 punt)
  - Pràctica 3 (1,5 punts):
    - **Per fer ponderació: Nota sobre 10 >= 4**
  - Pràctica 4 (1,5 punts):
    - **Per fer ponderació: Nota sobre 10 >= 4**

Nota Final = Teoria + Pràctiques

AVALUACIÓ ÚNICA

L'avaluació es separa en dos blocs

- **Teoria (gener)** (5,5 punts) **Nota sobre 10 >= 4**
  - Examen final presencial

- **Pràctiques** (4,5 punts): **Nota sobre 10 >= 4**
  - Pràctica 1 (0,5 punt)
  - Pràctica 2 (1 punt)
  - Pràctica 3 (1,5 punts):
    - Per fer ponderació: **Nota sobre 10 >= 4**
  - Pràctica 4 (1,5 punts):
    - Per fer ponderació: **Nota sobre 10 >= 4**

Data d'avaluació única: 29.11.2021

RE-AVALUACIÓ

La re-avaluació es separa en dos blocs: Teoria i Pràctiques

Nota Final Teoria (5,5 punts) = Examen de re-avaluació

Nota Re-avaluació Pràctica 2 (puntuació màxima 6 punts sobre 10)

Nota Re-avaluació Pràctica 3 (puntuació màxima 6 punts sobre 10)

Cal superar un mínim de 4 **punts**

Nota Re-avaluació Pràctica 4 (puntuació màxima 6 punts sobre 10)

Cal superar un mínim de 4 **punts**

NOTA FINAL

Es separa en dos blocs: Teoria i Pràctiques: S'aprova amb 5 sobre 10

Nota Final Teoria (5,5 punts) =
màxim (Teoria , Examen de re-avaluació)

Cal obtenir més de **4**

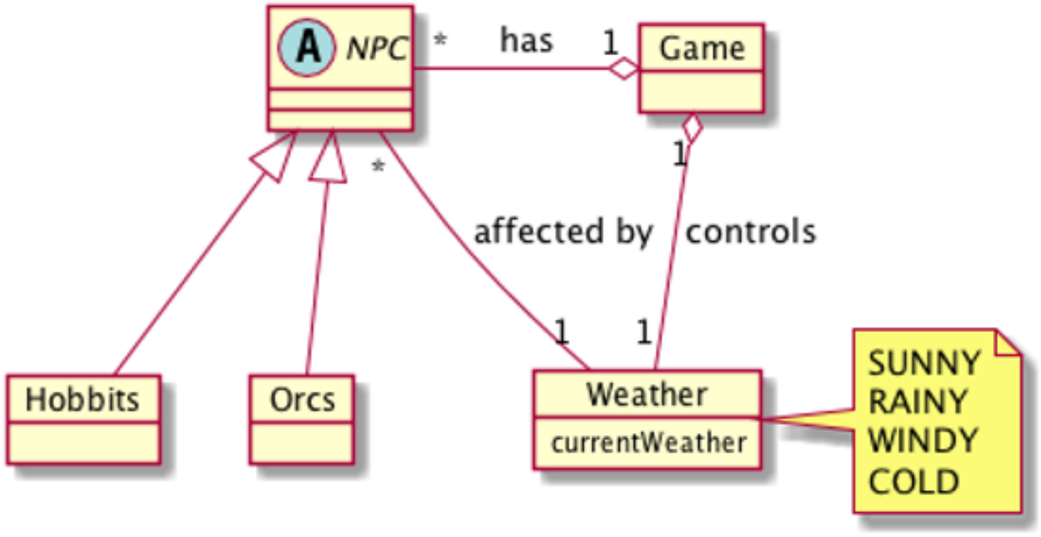
Nota Final de Pràctiques (4,5 punts) =
Pràctica 1 + màxim (Pràctica 2, Re-avaluació Pràctica 2)
+ màxim (Pràctica 3, Re-avaluació Pràctica 3)
+ màxim (Pràctica 4, Re-avaluació Pràctica 4)

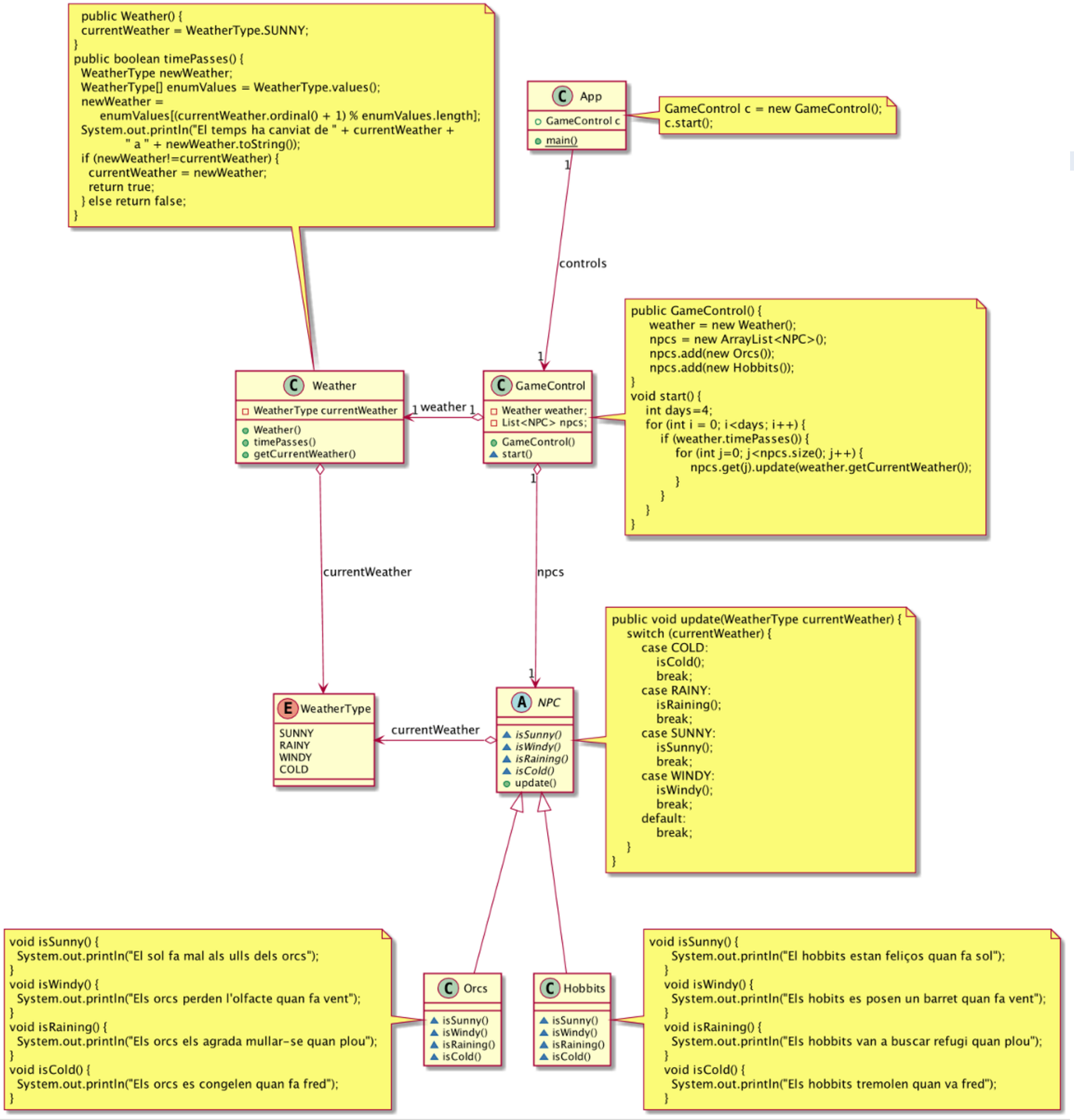
Cal superar un mínim de 4 **punts a les Pràctiques 3 i 4**

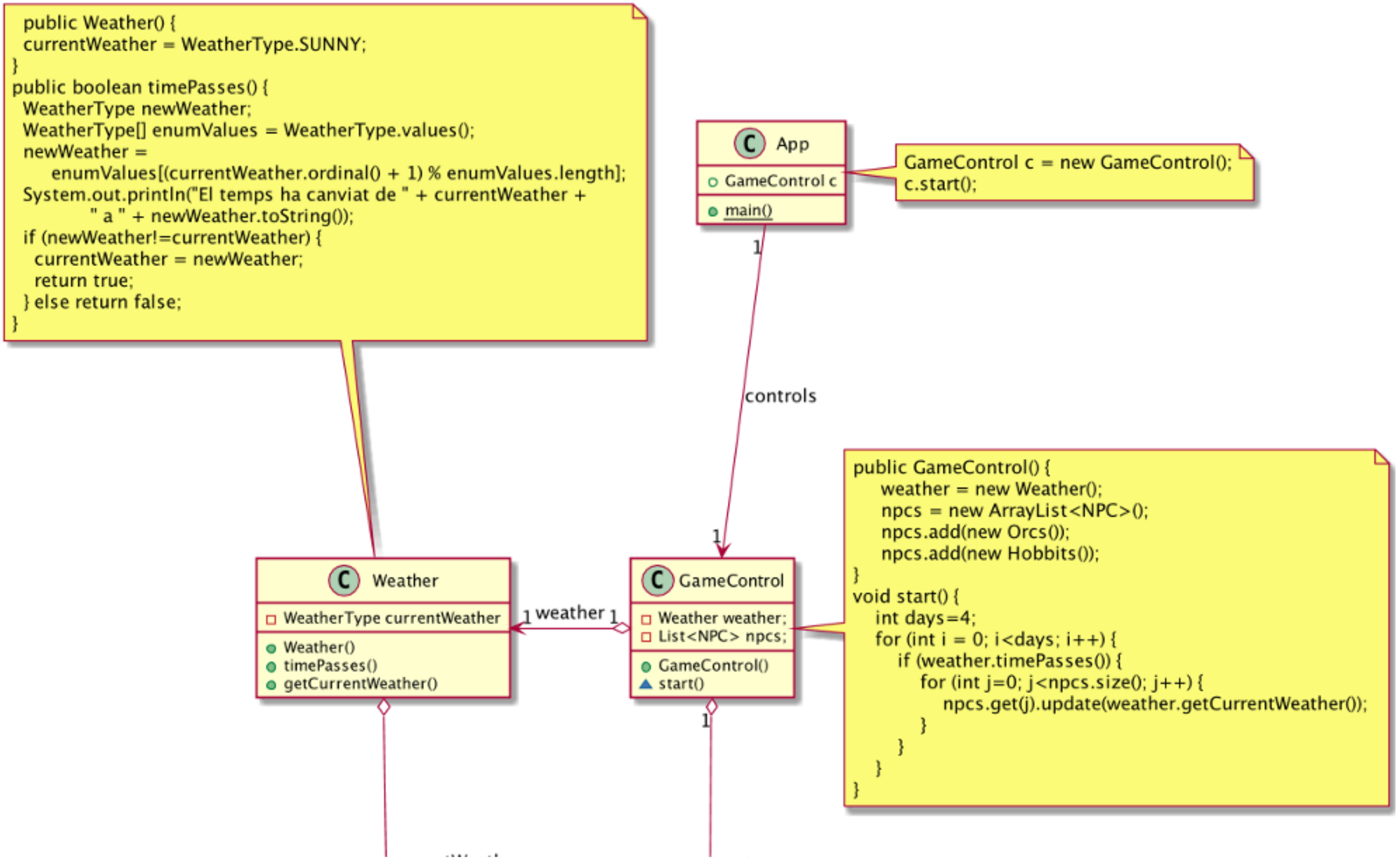
Problema: examen 2018-19

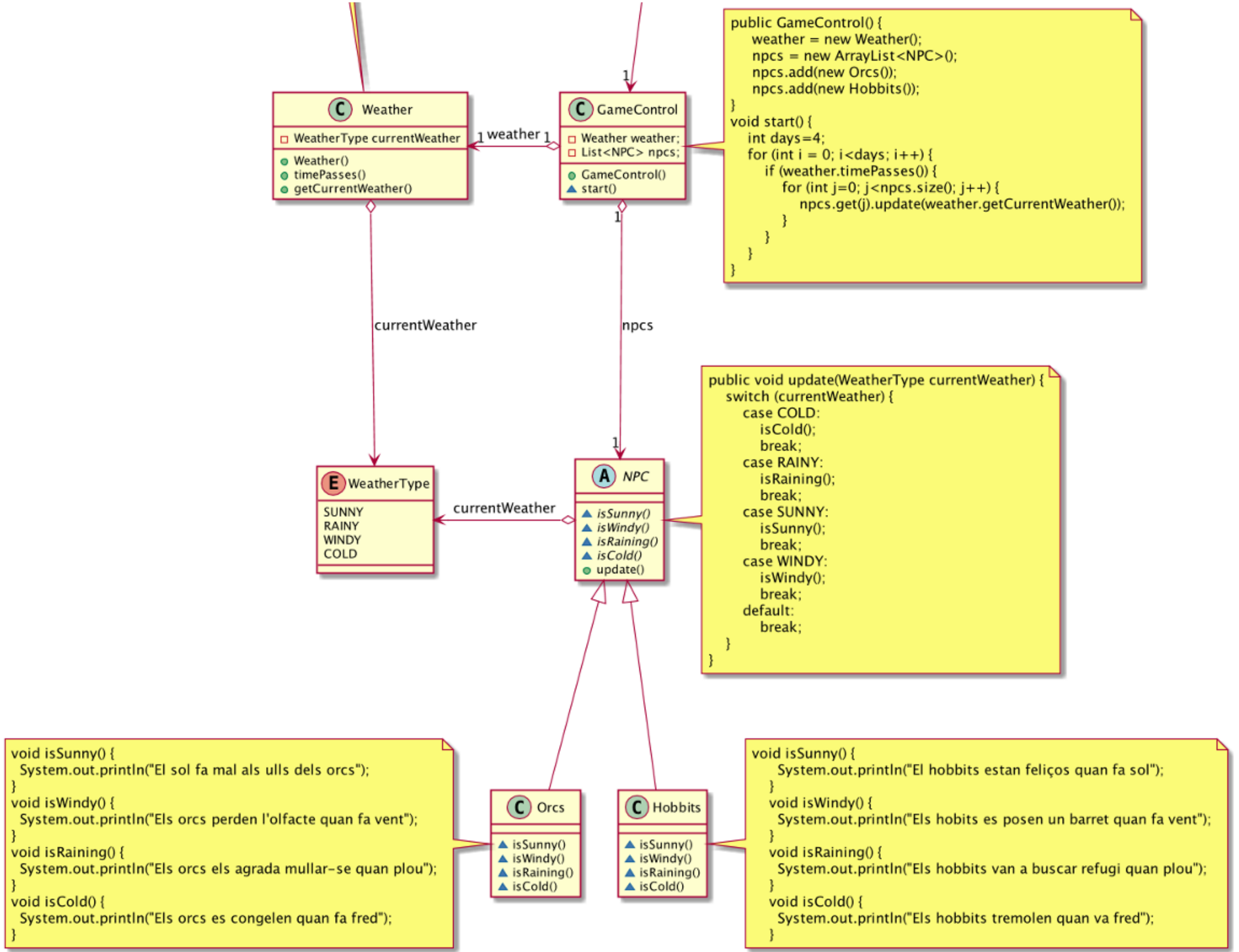
Es vol dissenyar un joc que té dos tipus de NPCs (els orcs i els hobbits) i que té diferents condicions meteorològiques que varien durant el temps del joc. Quan aquestes condicions varien, els orcs i els hobbits reaccionen de diferents maneres. Les condicions meteorològiques poden ser de 4 tipus (SUNNY, 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.




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()
void isSunny() {
System.out.println("El sol fa mal als ulls dels orcs");
}
void isWindy() {
System.out.println("Els orcs perden l'olfacte quan fa vent");
}
void isRaining() {
System.out.println("Els orcs els agrada mullar-se quan plou");
}
void isCold() {
System.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");
}


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()
1
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());
}
}
}
}


Weather
WeatherType currentWeather
Weather()
timePasses()
getCurrentWeather()
1
weather
1
GameControl
Weather weather;
List<NPC> npcs;
GameControl()
start()
1
1
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());
}
}
}
}
currentWeather
npcs
1
WeatherType
SUNNY
RAINY
WINDY
COLD
currentWeather
NPC
isSunny()
isWindy()
isRaining()
isCold()
update()
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;
}
}
void isSunny() {
System.out.println("El sol fa mal als ulls dels orcs");
}
void isWindy() {
System.out.println("Els orcs perden l'olfacte quan fa vent");
}
void isRaining() {
System.out.println("Els orcs els agrada mullar-se quan plou");
}
void isCold() {
System.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");
}

Problema: examen 2018-19

a) Quins principis S.O.L.I.D. vulnera aquest codi? Raona la resposta.

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: ___

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)

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

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

b.5. Observacions addicionals

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.

Problema: examen 2018-19

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 a la classe NPC i es seves derivades. Tota classe derivada es pot comportar com la classe mare

I: No es vulnera. No aplica

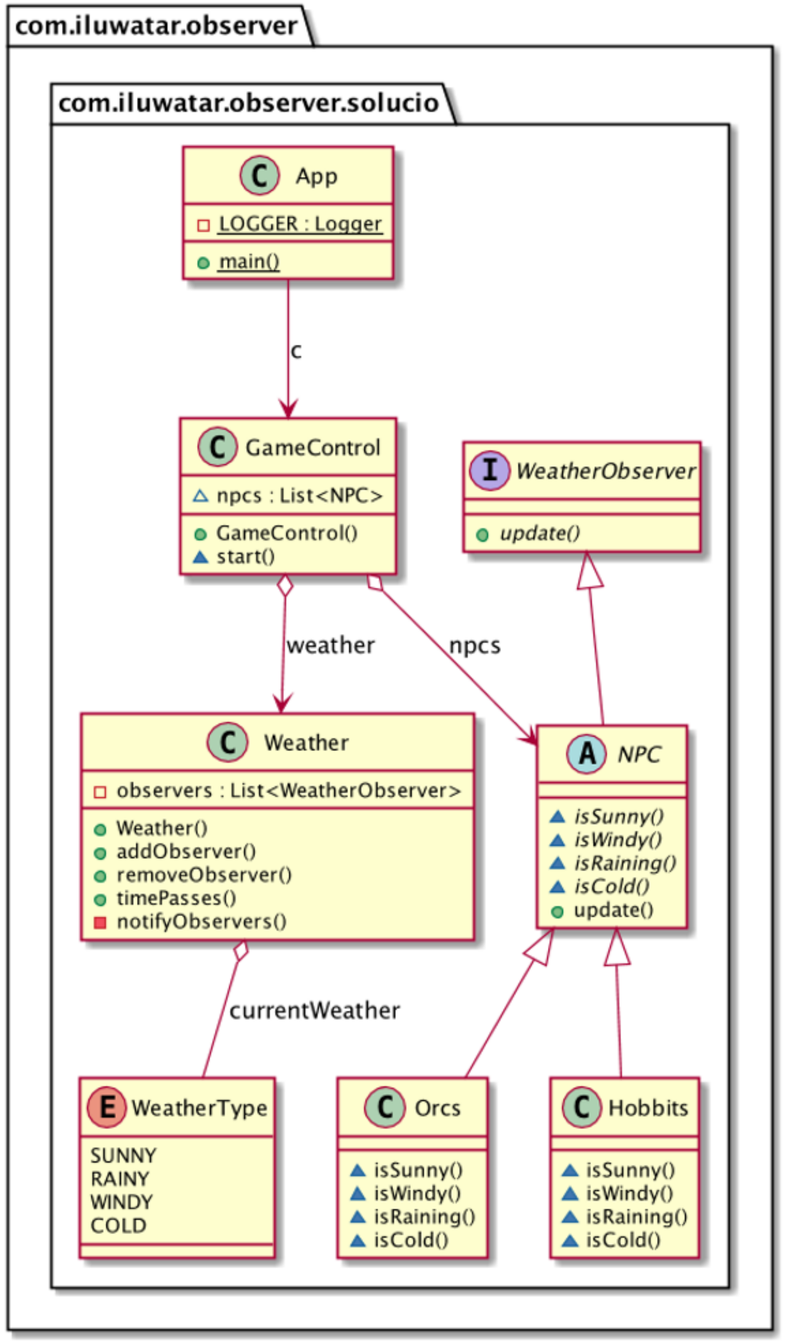
D: Es vulnera en tenir fixes les accions o comportaments (pero també es acceptable dir que no es vulnera)

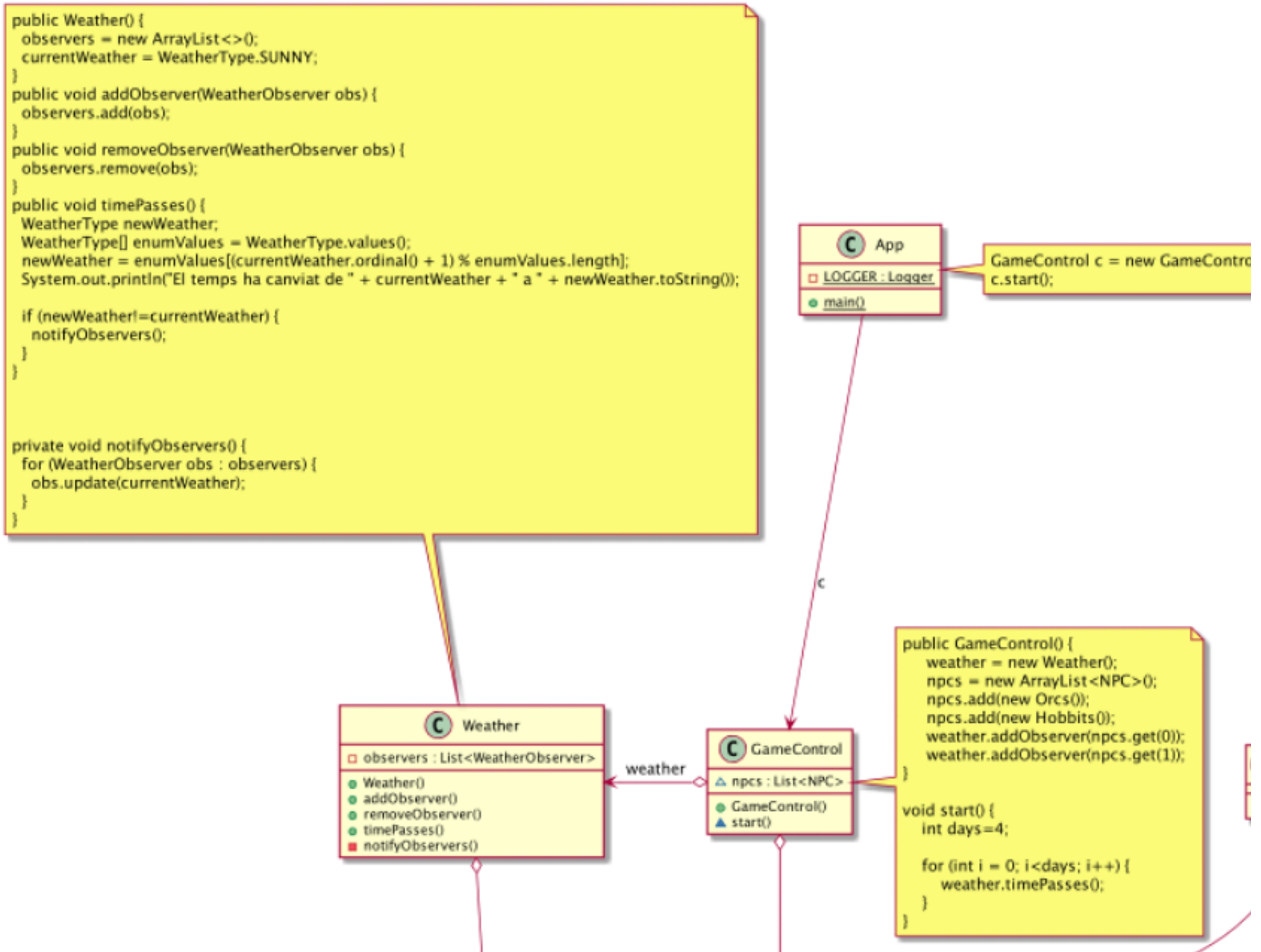
Problema: examen 2018-19

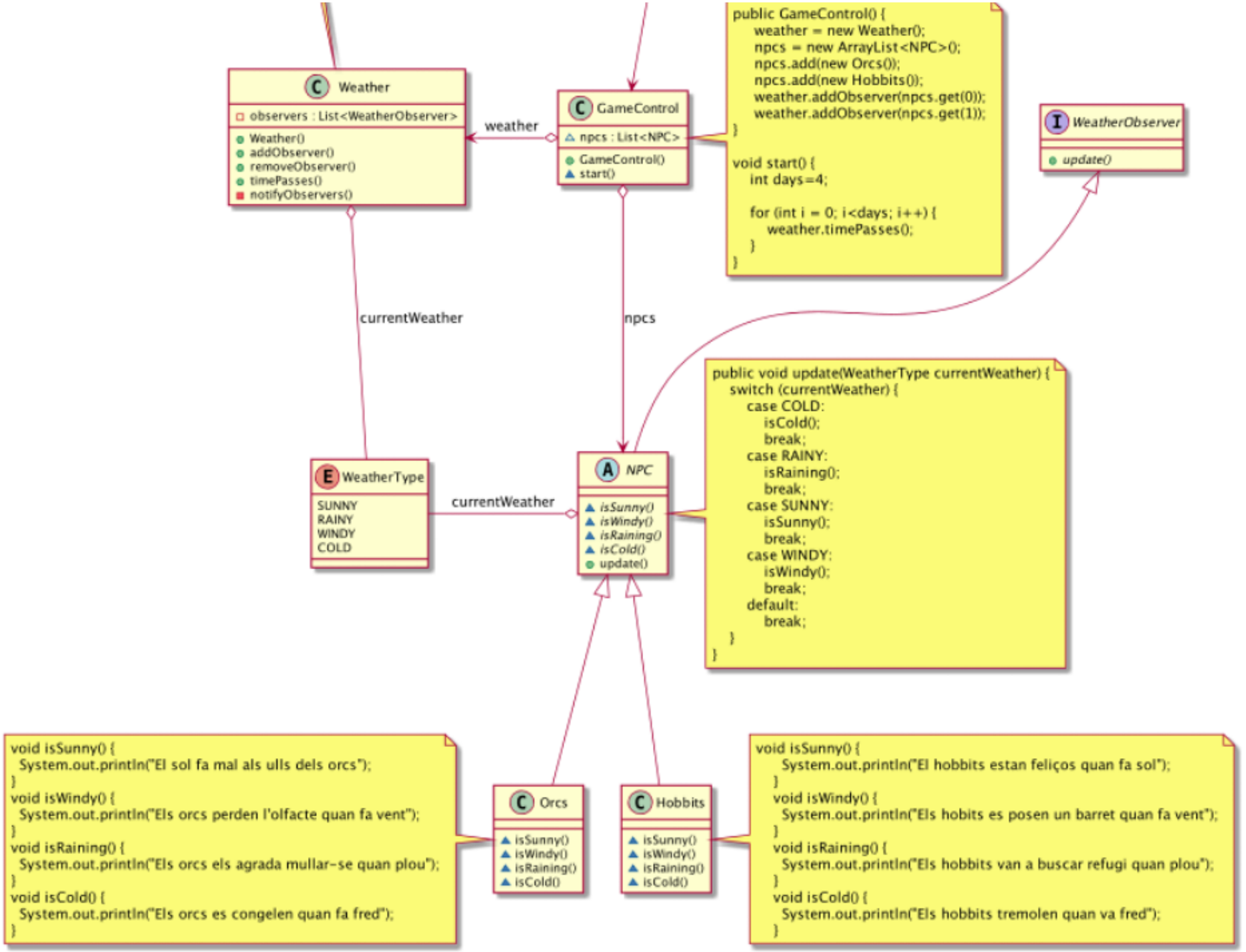
b.1. Nom del patró principal i tipus de patró: ___

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 (Observat) 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




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);
}
}
App
LOGGER : Logger
main()
GameControl c = new GameContro
c.start();
c
Weather
observers : List<WeatherObserver>
Weather()
addObserver()
removeObserver()
timePasses()
notifyObservers()
weather
GameControl
npcs : List<NPC>
GameControl()
start()
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();
}
}


Weather
observers : List<WeatherObserver>
Weather()
addObserver()
removeObserver()
timePasses()
notifyObservers()
weather
GameControl
npcs : List<NPC>
GameControl()
start()
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();
}
}
WeatherObserver
update()
currentWeather
npcs
WeatherType
SUNNY
RAINY
WINDY
COLD
currentWeather
NPC
isSunny()
isWindy()
isRaining()
isCold()
update()
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;
}
}
void isSunny() {
System.out.println("El sol fa mal als ulls dels orcs");
}
void isWindy() {
System.out.println("Els orcs perden l'olfacte quan fa vent");
}
void isRaining() {
System.out.println("Els orcs els agrada mullar-se quan plou");
}
void isCold() {
System.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");
}

Problema: examen 2018-19

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)

Problema: examen 2018-19

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();
    }
}
```

Problema: examen 2018-19

b.5. Observacions generals:

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);
  }
}
```

Problema: examen 2018-19

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.

SOUCIO 1: Des de **updateWeather** de la classe **NPCs**. Aqui caldira saber el tipus del NPC i el tipus de Weather per poder construir l'estratègia a aplicar.

Directament des de la classe NPC es podria cridar a l'accio creada pels Factory Method.

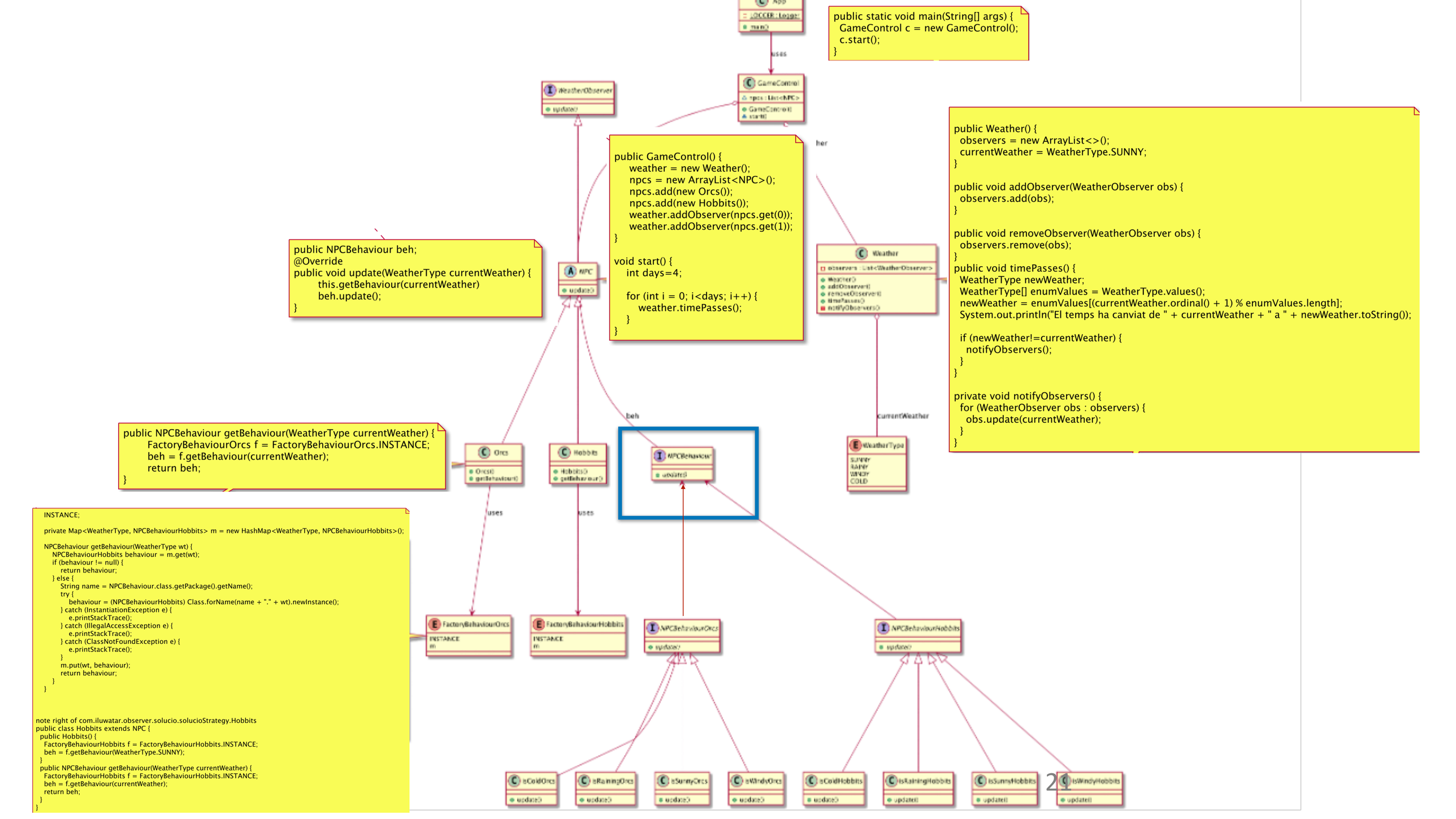
Per a implementar el Strategy es faria fet una interficie de behaviour.

Problema: examen 2018-19

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 dos **Factory Method** que encapsularia la creació dels diferents comportaments dels dos tipus de personatges.

SOLUCIO 2: Es tenen dues Factories de les Strategies segons el tipus de NPCs. Aquestes Factories s'usen des del propi Hobbit o des de l'Orc per a construir l'estratègia concreta en el moment del update. Les factories poden ser singletons que controles que nomes les estrategies s'instanciin només un cop.


public static void main(String[] args) {
GameControl c = new GameControl();
c.start();
}
App
LOGGER : Logger
main()
uses
GameControl
npcs : List<NPC>
GameControl()
start()
WeatherObserver
update()
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();
}
}
her
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);
}
}
public NPCBehaviour beh;
@Override
public void update(WeatherType currentWeather) {
this.getBehaviour(currentWeather)
beh.update();
}
NPC
update()
Weather
observers : List<WeatherObserver>
Weather()
addObserver()
removeObserver()
timePasses()
notifyObservers()
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, NPCBehaviourHobbits> m = new HashMap<WeatherType, NPCBehaviourHobbits>();
NPCBehaviour getBehaviour(WeatherType wt) {
NPCBehaviourHobbits behaviour = m.get(wt);
if (behaviour != null) {
return behaviour;
} else {
String name = NPCBehaviour.class.getPackage().getName();
try {
behaviour = (NPCBehaviourHobbits) 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;
}
}
note right of com.iluwatar.observer.solucio.solucioStrategy.Hobbits
public class Hobbits extends NPC {
public Hobbits() {
FactoryBehaviourHobbits f = FactoryBehaviourHobbits.INSTANCE;
beh = f.getBehaviour(WeatherType.SUNNY);
}
public NPCBehaviour getBehaviour(WeatherType currentWeather) {
FactoryBehaviourHobbits f = FactoryBehaviourHobbits.INSTANCE;
beh = f.getBehaviour(currentWeather);
return beh;
}
}
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()