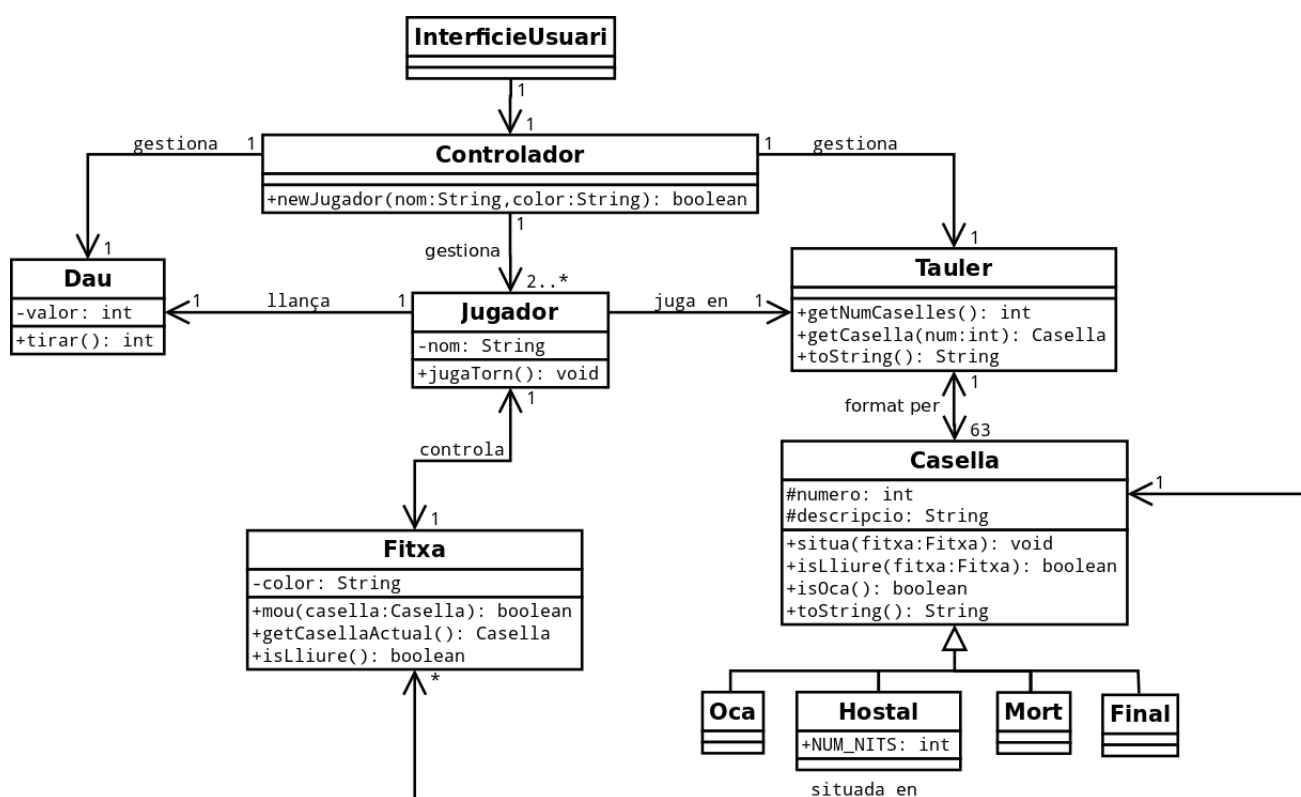


Per tal de posar en pràctica els coneixements adquirits a l'assignatura de Metodologia de Programació Orientada a Objectes (MOO), ens hem plantejat la implementació d'una versió simplificada de l'arxiconegut **Joc de la Oca**. El Joc de la Oca és un joc de tauler, el qual conté 63 caselles, numerades de la 1 a la 63, disposades seqüencialment en forma d'espiral. En aquest tauler, els jugadors controlen cada un d'ells una fitxa, inicialment situada a la casella número 1. Durant el seu torn, cada jugador llança un dau i avança la fitxa que controla un nombre de caselles igual al valor del dau. El primer jugador que arriba a la casella 63 guanya.

Un cop finalitzades les etapes d'anàlisi i disseny del programari de l'aplicació, hem arribat al següent diagrama de classes UML:



La gràcia del Joc de la Oca recau en el fet que algunes de les caselles del tauler tenen característiques especials, les quals poden fer avançar o retrocedir la fitxa d'un jugador quan cau en una d'aquestes caselles. Al llarg dels anys s'han anat introduint diverses caselles especials en el Joc de la Oca. Tal i com mostra el diagrama de classes UML, en la nostra versió del joc n'hem considerat les següents:

- **Oca:** és la casella especial més característica del joc. Quan la fitxa d'un jugador cau en una oca, la fitxa salta a la següent oca del tauler i el jugador pot tornar a tirar.
- **Hostal:** quan la fitxa d'un jugador cau a l'hostal, aquest ha d'estar un cert nombre de torns sense jugar, típicament 3 (el jugador passa 3 nits a l'hostal).
- **Mort:** quan la fitxa d'un jugador cau a aquesta casella, el jugador ha de començar altre cop des de la casella número 1.
- **Final:** també coneguda com a "jardí de la oca", és la casella final del tauler, és a dir, la 63. Quan la fitxa d'un jugador cau en aquesta casella, el jugador guanya la partida.

A partir de la informació de l'enunciat, així com del diagrama de classes UML proporcionat, respon a les següents preguntes de forma justificada:

1. **(0,5 punts)** La relació entre les classes Tauler i Casella s'ha definit mitjançant una associació bidireccional, doncs el comportament inclòs en algun dels mètodes de la classe Casella pot requerir l'accés al Tauler. Creus que aquesta relació es podria haver definit com a relació de composició, on el Tauler fos una composició de 63 Caselles?

Una de les característiques clau de les relacions de composició és la seva navegabilitat unidireccional del tot (en aquest cas seria el Tauler) cap a les parts (Caselles). Aleshores, en el supòsit que els mètodes definits a la classe Casella hagin d'accedir al Tauler (necessitin invocar mètodes sobre l'objecte Tauler que conté a les Caselles), no podrem utilitzar una relació de composició, doncs la seva navegabilitat unidireccional ens prohibiria aquesta operació.

2. **(0,5 punts)** La classe Casella s'ha definit com a classe convencional, la qual pot ser instanciada per a representar qualsevol casella NO especial del Tauler. En cas d'haver definit la classe Casella com a abstracta, quins canvis hauríem d'efectuar en el diagrama de classes UML proporcionat?

En principi, una classe abstracta s'utilitza per representar un concepte tant genèric que no té sentit instanciar-lo durant l'execució del programa. En efecte, el compilador de Java detecta i prohibeix qualsevol intent de creació d'un objecte d'una classe abstracta. Per tant, si s'opta per definir la classe Casella com a abstracta, no la podrem instanciar per crear les Caselles no especials del Tauler. Això ens forçaria a haver de definir una nova subclasse de Casella (p. ex., anomenada "Normal"), la qual representaria les Caselles no especials del Tauler, i que podríem instanciar-la amb aquest objectiu.

3. **(0,5 punts)** Alguns dels mètodes mostrats en el diagrama de classes UML són polimòrfics. Indica'ls. Indica també quines classes haurien de redefinir-los per tal que el codi polimòrfic pugui funcionar de forma adequada.

Els mètodes polimòrfics mostrats al diagrama de classes UML són:

`+situa(fitxa:Fitxa):void` de la classe Casella -> Aquest mètode implementaria el comportament de la Casella quan la Fitxa fitxa es situa en ella. L'haurien de redefinir les subclasses Oca, Hostal, Mort i Final, ajustant-lo al seu comportament especial.

`+isLliure(fitxa:Fitxa):boolean` de la classe Casella -> Aquest mètode retorna true/false en funció de si la Fitxa fitxa és lliure per a ser jugada o no. En general, el mètode sempre retorna true, excepte en el cas de la subclasse Hostal, la qual hauria de redefinir-lo per que retorni false en aquells casos en que la fitxa romanguí bloquejada.

`+isOca():boolean` de la classe Casella -> Aquest mètode retorna true/false en funció de si la Casella sobre la qual s'invoca el mètode és una Oca o no. En general el mètode sempre retorna false, excepte en el cas de la subclasse Oca, que l'ha de redefinir per tal que retorni true.

`+toString():String` de la classe Casella -> L'han de redefinir totes les seves subclasses per a mostrar la informació específica més rellevant d'aquestes.

4. **(2.5 punts)** Escriu la part de codi de la definició de totes les classes del diagrama UML, començant per `public class`. Inclou només la declaració dels atributs de les classes, fins i tot els que implementin relacions.

```
public class InterficieUsuari {  
    private Controlador controlador;  
}
```

```

public class Controlador {
    private Tauler tauler;
    private Dau dau;
    private List<Jugador> jugadors;
}

public class Tauler {
    private Casella[] caselles;
}

public class Dau {
    private int valor;
}

public class Jugador {
    private String nom;
    private Tauler tauler;
    private Dau dau;
    private Fitxa fitxa;
}

public class Fitxa {
    private String color;
    private Jugador jugador;
    private Casella casella;
}

public class Casella {
    protected int numero;
    protected String descripcio;
    protected Tauler tauler;
    protected Map<String, Fitxa> fitxes;
}

public class Oca extends Casella {
}

public class Hostal extends Casella {
    public static final int NUM_NITS = 3;
}

public class Mort extends Casella {
}

public class Final extends Casella {
}

```

Les preguntes a continuació sol·liciten la implementació d'alguns dels mètodes definits per a les classes de l'aplicació. En alguns casos, pot succeir que la correcta implementació del mètode sol·licitat requereixi la invocació de **mètodes addicionals** d'altres classes de l'aplicació (p.ex., getters, setters...). No cal que implementeu aquests mètodes addicionals, però **si que cal esmentar breument quin n'és el seu comportament esperat**.

5. **(1 punt)** Defineix el mètode constructor de la classe Jugador. La llista de paràmetres del mètode ha de permetre la inicialització correcta de TOTS els atributs d'un nou objecte Jugador quan el creem. Tingues en compte que la capçalera del mètode constructor de la

classe Fitxa és la següent: `public Fitxa (String color, Casella casella);` on color és el color de la nova Fitxa i casella la Casella on es situa inicialment aquesta.

```
/* A l'hora de definir el constructor de Jugador, hem de tenir en compte quins dels seus atributs ja estaran creats a l'hora de crear el nou Jugador i, per tant, ens els hauran de passar com a paràmetres. El dau i el tauler de la partida són atributs de Controlador. L'objecte controlador ja els haurà inicialitzat (creat) prèviament a la creació del nou Jugador, i els hi passarà com a paràmetre al constructor de Jugador */

public class Jugador {

    public Jugador (String nom, Tauler tauler, Dau dau, String color) {
        this.nom = nom;
        this.tauler = tauler;
        this.dau = dau;
        this.fitxa = new Fitxa (color, this.tauler.getCasella(1));
    }
}

/* Hem utilitzat el mètode +getCasella(num:int) de Tauler per situar de bon inici la fitxa del nou Jugador que hem creat a la casella número 1 del Tauler */
```

6. **(1.5 punts)** Implementa el mètode `+newJugador(nom:String, color:String):boolean` de la classe Controlador. Aquest mètode crea un nou objecte Jugador del Joc de la Oca, amb el nom i color de fitxa passats com a paràmetre. Tingues en compte que NO pot existir més d'un Jugador controlant una Fitxa del mateix color. En aquest cas, el Jugador no s'ha de crear i el mètode retorna false (retorna true si el Jugador es crea).

```
public class Controlador {

    public boolean newJugador (String nom, String color) {

        Iterator<Jugador> it = this.jugadors.iterator();
        while (it.hasNext())
        {
            Jugador jugador = it.next();
            if (jugador.getFitxa().getColor().equals(color))
            {
                return false;
            }
        }

        Jugador nou = new Jugador (nom, this.tauler, this.dau, color);
        this.jugadors.add (nou);
        return true;
    }
}

/*Hem utilitzat els getters +getFitxa() de Jugador i +getColor() de Fitxa per poder obtenir el valor dels respectius atributs privats fitxa i color de Jugador i Fitxa, així com el mètode constructor de la classe Jugador tal i com l'hem definit a la pregunta anterior */
```

7. **(2 punts)** Implementa els mètodes `+toString():String` de Casella i Tauler. En el cas de Casella, aquest mètode ha de retornar un String amb el número i descripció de la Casella, així com la llista de Fitxes ubicades sobre d'aquesta, amb el seu color i nom del Jugador

que les controlen. En el cas de Tauler, el mètode ha de retornar un String amb la informació de totes Caselles del Tauler.

```
public class Casella {

    public String toString() {

        String s = "Número: " + this.num + "\n";
        s = s + "Descripció: " + this.descripcio + "\n";

        Iterator<Fitxa> it = this.fitxes.values().iterator();
        while (it.hasNext())
        {
            Fitxa fitxa = it.next();
            s = s + "\t Color: " + fitxa.getColor();
            s = s + "; jugador: " + fitxa.getJugador().getNom() + "\n";
        }
        return s;
    }
}

public class Tauler {

    public String toString() {

        String s = new String();
        for (int i = 0; i < this.caselles.length; i++)
        {
            s = s + this.caselles[i].toString();
        }
        return s;
    }
}

/* Hem utilitzat els getters +getColor() i +getJugador() de Fitxa per obtenir el
valor dels atributs privats color i jugador de Fitxa, així com el getter
+getNom() de Jugador per obtenir el valor de l'atribut privat nom de Jugador */
```

8. **(1.5 punts)** Implementa el mètode +jugarTorn():void de la classe Jugador. En aquest mètode: **1)** El Jugador comprova que la seva fitxa és lliure per a poder ser jugada (no està, p.ex., a l'Hostal). Amb aquest objectiu, podeu usar el mètode +isLliure():boolean de Fitxa, el qual retorna true/false en funció de si la Fitxa és lliure o no. Si la fitxa és lliure, **2)** el Jugador llança el Dau, recull el seu valor i mou la Fitxa a la Casella corresponent. Per simplicitat, assumirem que si la Fitxa del Jugador sobrepassa l'última Casella amb el valor del Dau, es queda directament a l'última Casella (no rebota i torna enrere). Per moure la Fitxa, pots usar el mètode +mou(casella:Casella): boolean de Fitxa, el qual mou la Fitxa a la Casella que li passem com a paràmetre. Aquest mètode retorna true/false en funció de si el Jugador pot tornar a llençar el Dau o no (p.ex., ha caigut en una Oca). Mentre el retorn és true, es tornen a efectuar les accions del pas 2.

```
public class Jugador {

    public void jugarTorn() {

        if (this.fitxa.isLliure())
        {
            boolean continua = true;
            while (continua)
```

```

        {
            int valor = this.dau.tirar();
            int numCD = this.fitxa.getCasellaActual().getNumero() + valor;

            Casella desti = null;
            if (numCD > this.tauler.getNumCaselles())
            {
                desti = this.tauler.getCasella(this.tauler.getNumCaselles());
            }
            else
            {
                desti = this.tauler.getCasella(numCD);
            }

            continua = this.fitxa.mou(desti);
        }
    }
}

```

/\* A part dels mètodes suggerits a l'enunciat de la pregunta, hem usat el mètode +tirar() de Dau per obtenir el número de caselles que hem d'avançar la Fitxa del Jugador, +getCasellaActual() de Fitxa per obtenir la Casella on estava ubicada la Fitxa abans de tirar, +getNumero() per aconseguir el valor de l'atribut protected número de Casella, així com +getCasella(num:int) i +getNumCaselles() de Tauler per a gestionar el moviment de la Fitxa un cop tirem el Dau \*/