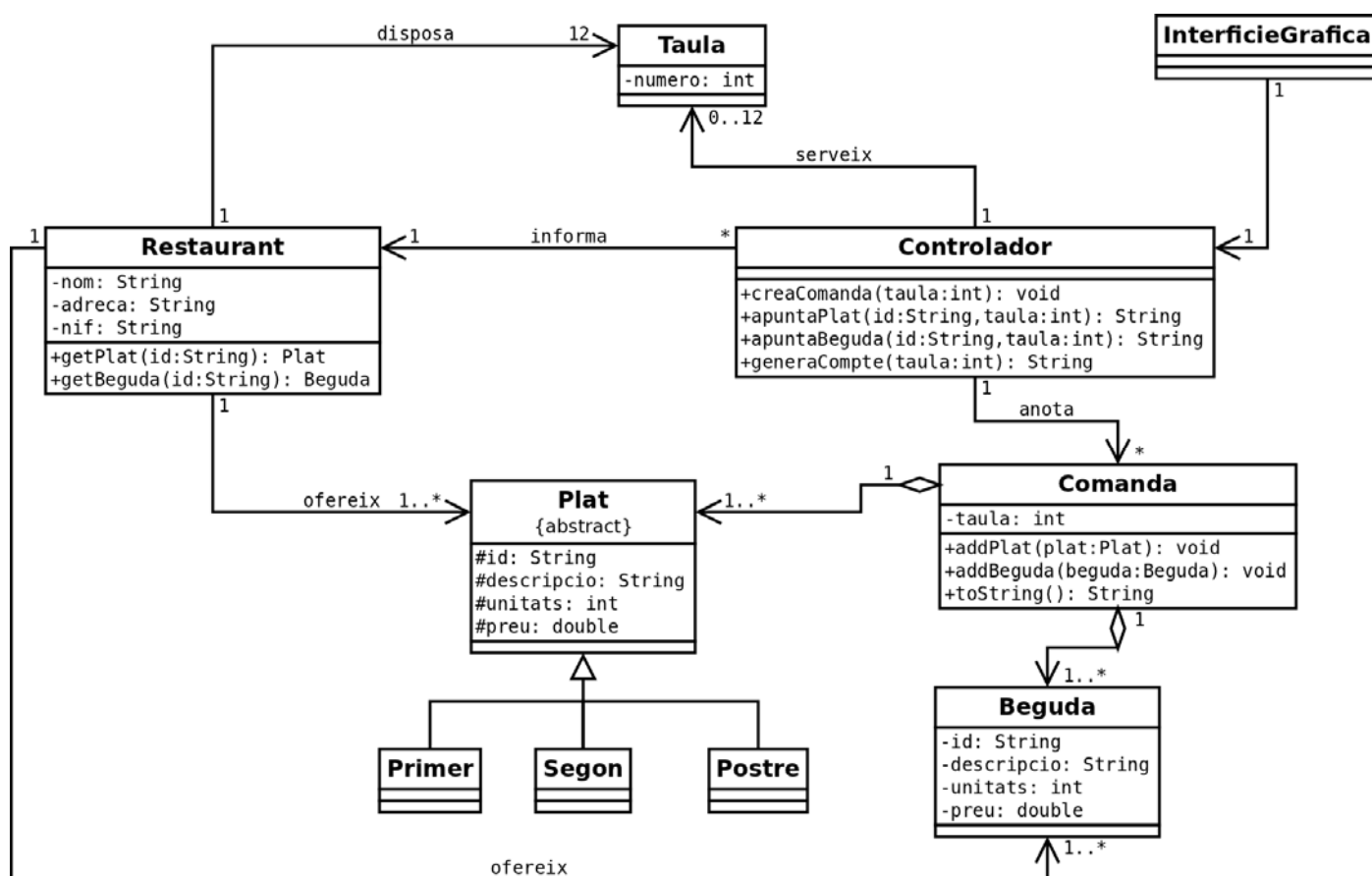


La cadena de restaurants MenjaGuai® ha encarregat una senzilla aplicació per assistir els cambrers dels seus establiments a l'hora de prendre nota i generar els comptes de les taules que serveixen. Aquesta aplicació funcionarà en un petit dispositiu mòbil que durà cada cambrer, i que aquest utilitzarà per introduir les comandes dels clients. L'aplicació, llavors, es connectarà al servidor centralitzat del restaurant via Wi-Fi, notificant les demandes introduïdes al servei de cuina (els plats sol·licitats) i barra (les begudes). Cal mencionar que el servidor centralitzat del restaurant gestiona la disponibilitat de cada plat i beguda anunciada a la carta. Així, el cambrer pot detectar ràpidament la manca d'existències d'un plat o beguda sol·licitada pel client i informar-lo al respecte.

Un cop finalitzades les etapes d'anàlisi i disseny del programari que ens han encarregat, hem arribat al següent diagrama de classes UML. Per simplicitat, hem assumit que podem representar el servidor centralitzat del restaurant com a un únic objecte de tipus Restaurant, directament accessible des de l'objecte Controlador de l'aplicació:



A partir del diagrama de classes UML mostrat es demana:

1. La classe Plat s'ha definit com a abstracta. Creus que és realment necessari fer-ho d'aquesta forma? Raona la teva resposta **(0.5 punts)**.

En el diagrama de classes UML de l'enunciat, la classe Plat s'ha introduït com a classe fonamental a partir de la qual se'n volen crear especialitzacions (Primer, Segon, Postre). Definint aquesta classe com a abstracta, el dissenyador de l'aplicació s'assegura que es prohibirà qualsevol intent de creació d'un objecte de tipus Plat, tenint en compte que el restaurant sempre acabarà servint tipus específics de plats: primers, segons i postres.

2. Quan definim classes en un llenguatge de programació orientat a objectes, com per exemple Java, és necessari definir, com a mínim, un mètode constructor per cadascuna d'elles. Tenint en compte, però, que Plat és una classe abstracta, creus que és necessari proporcionar-li algun mètode constructor? Raona la teva resposta **(0.5 punts)**.

Sempre hem de proporcionar com a mínim un mètode constructor per a cada una de les classes Java que definim per a les nostres aplicacions, i les classes abstractes no en són pas una excepció. Encara que no puguem instanciar directament una classe abstracta, el mètode constructor d'aquesta el continuem necessitant per que pugui ser invocat des dels mètodes constructors de les seves subclasses -emprant la crida `super()`- per tal d'inicialitzar tots els atributs que aquestes subclasses puguin haver heretat d'ella.

3. Per tal d'aplicar correctament el principi d'encapsulació, sempre definim els atributs de les classes com a privats (-). No obstant això, a la classe Plat s'ha elegit una visibilitat protegida per aquests (#). Quines implicacions té això? Què hauria passat si s'hagués definit visibilitat privada també per als atributs de la classe Plat? **(0.5 punts)**.

Definint els atributs de la classe Plat com a protegits estem permetent que les seves subclasses puguin accedir directament a ells, quan els heretin a través del mecanisme d'herència. En cas que els haguéssim definit amb visibilitat privada, les subclasses de Plat, tot i heretar aquests atributs, no podrien accedir-hi directament. Ho haurien de fer a través dels mètodes accessors (getters i setters públics) heretats de la mateixa classe Plat.

4. Escribeu la part de codi de la definició de totes les classes del diagrama de classes UML, començant per `public class`. Inclou només la declaració dels atributs de les classes, fins i tot aquells que implementin relacions **(3 punts)**.

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

public class Controlador {
    private Restaurant restaurant;
    private List<Taula> taules;
    private Map<Integer, Comanda> comandas;
}

public class Restaurant {
    private String nom, adreça, nif;
    private Taula taules[];
    private Map<String, Plat> plats;
    private Map<String, Beguda> begudes;
}
```

```

public class Taula {
    private int numero;
}

public class Comanda {
    private int taula;
    private List<Plat> plats;
    private List<Beguda> begudes;
}

public class Beguda {
    private String id, descripcio;
    private int unitats;
    private double preu;
}

public abstract class Plat {
    protected String id, descripcio;
    protected int unitats;
    protected double preu;
}

public class Primer extends Plat {
}

public class Segon extends Plat {
}

public class Postre extends Plat {
}

```

**IMPORTANT:** Les preguntes que segueixen a continuació es centren en 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. **Heu de proporcionar també la implementació d'aquests mètodes addicionals.**

5. Implementa el mètode +creaComanda (taula: int): void de la classe Controlador **(0.5 punts)**.

```

public class Controlador {

    public void creaComanda (int taula)
    {
        Comanda comanda = new Comanda (taula);
        this.comandes.put(taula, comanda);
    }

}

```

6. Implementa els mètodes `+getPlat(id: String):Plat` i `+getBeguda(id: String): Beguda` de la classe `Restaurant`. Aquests mètodes retornen l'objecte `Plat` (o `Beguda`) l'identificador del qual concorda amb el valor del paràmetre `id: String`, i redueixen les unitats disponibles de l'objecte `Plat` (o `Beguda`) retornat en 1 unitat. En cas que no hi hagi unitats disponibles de l'objecte `Plat` (o `Beguda`) sol·licitat, ambdós mètodes retornen `null` **(1.5 punts)**.

```
/* En la següent implementació assumirem que sempre existeix un objecte
Plat/Beguda amb el id proporcionat. Aquesta condició la podria assegurar
la interfície gràfica, només deixant elegir plats i begudes al cambrer
d'entre la llista de plats i begudes ofertes pel restaurant */
```

```
public class Restaurant {
    public Plat getPlat (String id)
    {
        Plat plat = this.plats.get(id);
        if (plat.getUnitats() > 0){
            plat.setUnitats(plat.getUnitats()-1);
            return plat;
        }
        else{
            return null;
        }
    }

    public Beguda getBeguda (String id)
    {
        Beguda beguda = this.begudes.get(id);
        if (beguda.getUnitats() > 0){
            beguda.setUnitats(beguda.getUnitats()-1);
            return beguda;
        }
        else{
            return null;
        }
    }
}

/* Getters i setters de les classes Plat i Beguda */

public abstract class Plat {

    public int getUnitats(){
        return this.unitats;
    }
    public void setUnitats(int unitats){
        this.unitats = unitats;
    }
}
```

```

public class Beguda {
    public int getUnitats() {
        return this.unitats;
    }

    public void setUnitats(int unitats){
        this.unitats = unitats;
    }
}

```

7. Implementa els mètodes `+apuntaPlat(id: String, taula: int): String` i `+apuntaBeguda(id: String, taula: int): String` de la classe `Controlador`. Aquests mètodes invoquen els mètodes adequats de la classe `Restaurant` per assegurar-se de la disponibilitat de l'objecte `Plat` o `Beguda` identificat pel valor del paràmetre `id: String`. Essent així, aquest/a és afegit a la comanda de la taula on el cambrer està prenent nota (el valor del paràmetre `taula: int`). Fixeu-vos que tots dos mètodes tenen un retorn de tipus `String`. Aquest objecte de tipus `String` retornat contindrà simplement un missatge informant de la inclusió satisfactòria del plat/beguda a la comanda o, en cas contrari, de la indisponibilitat del plat/beguda sol·licitada **(1.5 punts)**.

```

/* Com abans, assumirem que la interfície gràfica només deixa generar
comptes per aquelles taules amb una comanda activa i pendent de cobrar */

public class Controlador {

    public String apuntaPlat (String id, int taula)
    {
        Plat plat = this.restaurant.getPlat(id);
        Comanda comanda = this.comandes.get(taula);
        if (plat != null){
            comanda.addPlat(plat);
            return "Plat afegit.";
        }
        else{
            return "Plat no disponible.";
        }
    }

    public String apuntaBeguda (String id, int taula)
    {
        Beguda beguda = this.restaurant.getBeguda(id);
        Comanda comanda = this.comandes.get(taula);
        if (beguda != null){
            comanda.addBeguda(beguda);
            return "Beguda afegida.";
        }
        else{
            return "Beguda no disponible.";
        }
    }
}

```

```

    }
}

public class Comanda {
    public void addPlat (Plat plat){
        this.plats.add (plat);
    }

    public void addBeguda (Beguda beguda) {
        this.begudes.add(beguda);
    }
}

```

8. Implementa els mètodes `+generaCompte(taula:int):String` de la classe `Controlador`. Aquest mètode ha de retornar un objecte de tipus `String`, el qual ha de contenir el nom, l'adreça i el NIF del restaurant, el número de taula per a la qual s'ha generat el compte (el valor del paràmetre `taula: int`), una llista amb tots els plats/begudes consumits en aquella taula (aquells inclosos en la comanda d'aquella taula) amb el preu de cada un d'ells i, al final de tot, el preu total a pagar a compte del client **(2 punts)**.

```

public class Controlador {
    public String generaCompte (int taula)
    {
        String s = new String();
        s = this.restaurant.getNom() + "\n";
        s = s + this.restaurant.getAdreca() + "\n";
        s = s + "nif: " + this.restaurant.getNIF() + "\n";
        s = s + "Taula: " + taula + "\n";
        s = s + this.comandes.get(taula).toString();

        return s;
    }
}

public class Comanda {

    public String toString()
    {
        String s = new String();
        double total = 0.0;

        Iterator<Plat> itp = this.plats.iterator();
        while (itp.hasNext()) {
            Plat plat = itp.next();
            s = s + plat.getDescripcio() + "\t" + plat.getPreu() + "\n";
            total = total + plat.getPreu();
        }

        Iterator<Beguda> itb = this.begudes.iterator();
    }
}

```

```

        while (itb.hasNext()) {
            Beguda beguda = itb.next();
            s = s + beguda.getDescripcio()+"\t"+ beguda.getPreu() + "\n";
            total = total + beguda.getPreu();
        }

        s = s + "Total: " + total;
        return s;
    }
}

/* Getters de la classe Restaurant */

public class Restaurant {
    public String getNom(){
        return this.nom;
    }

    public String getAdreca(){
        return this.adreca;
    }

    public String getNIF(){
        return this.nif;
    }
}

/* Getters de les classes Plat i Beguda */

public abstract class Plat {
    public double getPreu(){
        return this.preu;
    }
}

public class Beguda {
    public double getPreu(){
        return this.preu;
    }
}

```