# 1. Explicar les classes implementades:

Clases Principales y Relaciones

1. **Allotjament (Abstracta)**
   Clase base que representa cualquier tipo de alojamiento en el camping. Tiene atributos comunes como nom, idAllotjament, y los métodos para gestionar la estancia mínima dependiendo de la temporada.
   - Métodos clave:
     - getEstadaMinima(): Recupera la estancia mínima dependiendo de la temporada (alta o baja).
     - correcteFuncionament(): Método abstracto que se implementa en clases hijas para verificar si el alojamiento está en funcionamiento.

2. **Casa (Abstracta)**
   Hereda de Allotjament. Representa alojamientos que tienen propiedades comunes como mida, numHabitacions, y capacitatPersones.
   - Métodos clave:
     - Métodos getNumHabitacions(), getCapacitatPersones() para acceder a los detalles de las casas.

3. **Bungalow, BungalowPremium, Glamping, MobilHome, Parcela**
   Son clases específicas que heredan de Casa. Cada una tiene atributos y comportamientos únicos.
   - Bungalow: Tiene atributos como placesParquing, terrassa, tv, y aireFred. Su método correcteFuncionament() verifica si el aire acondicionado está en funcionamiento.
   - BungalowPremium: Subclasse de Bungalow que agrega noves características como serveisExtra y codiWifi. Implementa correcteFuncionament() asegurando que el código WiFi tenga entre 7 y 17 caracteres.
   - Glamping: Representa alojamientos con un enfoque en materiales específicos y si permiten mascotas.
   - MobilHome: Alojamiento con una terraza con barbacoa.
   - Parcela: Alojamiento con una superficie en metros cuadrados y posibilidad de conexión eléctrica.

4. **Camping**
   Es la clase principal que gestiona todos los alojamientos, clientes y reservas. Contiene métodos para agregar clientes y diferentes tipos de alojamientos, además de gestionar reservas.
   - Métodos clave:
     - afegirReserva(): Permite realizar una reserva tras verificar la disponibilidad y la estancia mínima.
     - calculAllotjamentsOperatius(): Cuenta los alojamientos operativos (aquellos con correcteFuncionament() retornando true).
     - getAllotjamentEstadaMesCurta(): Devuelve el alojamiento con la estancia mínima más corta.

5. **Reserva**
   Representa una reserva específica que vincula un cliente con un alojamiento y las fechas de entrada y salida.
   - Métodos clave:
     - isEstadaMinima(): Verifica si la reserva cumple con el mínimo de estancia requerido.
     - allotjamentDisponible(): Verifica si el alojamiento está disponible para las fechas solicitadas.

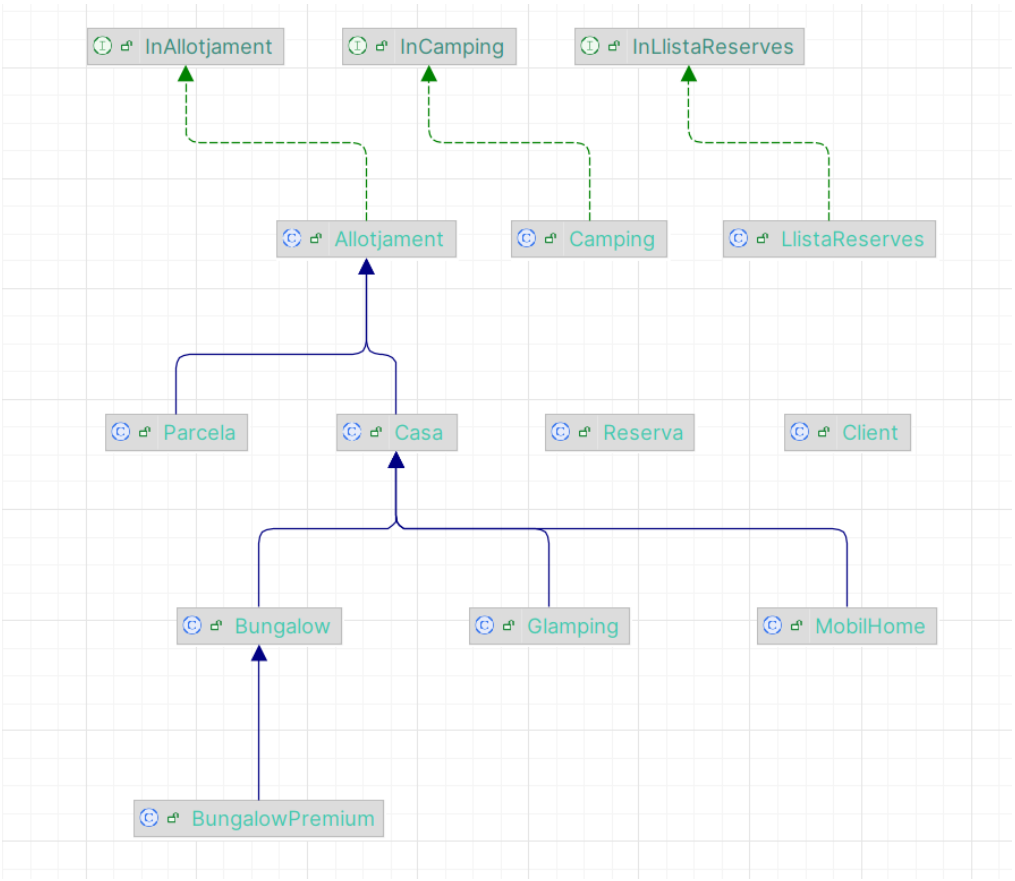
6. **Client**
   Clase que representa a un cliente con un nombre y un DNI. Tiene validación de formato para el DNI (debe tener 9 caracteres).

7. **Llista Reserves**
   Clase que gestiona todas las reservas realizadas. Permite agregar nuevas reservas, comprobar la disponibilidad de alojamientos y verificar si se cumple la estancia mínima.

Interacción entre Clases:

- Cliente y Reserva: Los clientes pueden hacer reservas para alojarse en uno de los alojamientos del camping.
- Camping y Allotjament: El camping contiene una lista de alojamientos y es responsable de añadir nuevos alojamientos y gestionar la disponibilidad.
- Reserva y Alojamiento: Cada reserva está vinculada a un alojamiento, y la disponibilidad se verifica al hacer una nueva reserva.
- Camping y Reserva: El camping puede obtener la lista de reservas y verificar la disponibilidad de los alojamientos.

## 2. Dibuixar el diagrama de relacions entre les classes.




## 3. Atributs de la classe Reserva i perquè:

**Atributs:**

1. Client client: Identifica el client que fa la reserva.
2. Allotjament allotjament: L'allotjament reservat.
3. LocalDate dataEntrada: Data d'entrada a l'allotjament.
4. LocalDate dataSortida: Data de sortida de l'allotjament.

**Raó:**

Aquests atributs permeten vincular un client amb un allotjament específic durant un període determinat, garantint el seguiment de les reserves.

## 4. Implementació de correcteFuncionament i perquè:

**On s'implementen:**

A la classe Allotjament com abstracte i en cada subclasse d'allotjament que representa un tipus diferent d'allotjament,(ex: Parcela, Bungalow Premium, etc.).

**Per què:**

Cada tipus d'allotjament té criteris diferents per funcionar correctament (ex: Parcel·la necessita connexió elèctrica, Bungalow Premium requereix codi WiFi vàlid). La implementación específica a cada subclasse permet encapsular la lògica única de cada tipus.

## 5. Classes amb toString() i perquè:

**Classes:**

1. Allotjament: Mostra nom, ID i estades mínimes.
2. Subclasses (Parcela, Bungalow, etc.): Afegeixen atributs específics (ex: mida, WiFi).

**Per què:**

Facilitar la visualització de les dades dels objectes en logs, interfícies o proves, millorant el procés de debugar.

## 6. Definicions d'atributs (public/private) i perquè:

**Private:**

Tots els atributs de totes les classes són privats.

**Raó:**

Encapsulament: Restringeix l'accés directe als atributs, obligant a usar getters/setters per validar dades (ex: DNI amb 9 caràcters). Això millora la seguretat i el control.

## 7. Crear client/allotjament durant una reserva: No és correcte.

**Problemes:**

Duplicació: Es crearien múltiples instàncies del mateix client/allotjament.

Inconsistència: Les dades no s'actualitzarien centralment (ex: canvis en un allotjament).

Solució òptima:
Utilitzar clients/allotjaments ja existents (afegits prèviament) per garantir integritat de dades.

## 8. Crear client/allotjament durant una reserva:

Vam realitzar diverses proves utilitzant JUnit per assegurar-nos que tot funcionés correctament. Es va posar a prova la creació de clients, de nous allotjaments després de fer algunes reserves i el solapament de dates de reserva. Varem verificar el correcte maneig d'excepcions, com ara les reserves en dates ocupades, la reserva d'allotjaments sense clients registrats o allotjaments no registrats. Els resultats obtinguts van ser els esperats: les reserves es van crear correctament, les excepcions es van gestionar de manera adequada i les dates solapades van ser correctament identificades, evitant errors.

## 9. Observacions generals.

En general, sobre el projecte hem après a utilitzar classes y treballar amb elles y les seves subclasses. Amb la POO vam poder dividir el projecte en parts més petites, que fa que sigui més còmode treballar i resoldre els problemes.