Práctica 1

Vamos a crear una pequeña aplicación (`comarquescli`) de consola con el fin de obtener información sobre las provincias y comarcas de la Comunidad Valenciana.

Nuestra aplicación ofrecerá desde la línea de órdenes tres funcionalidades:

- Obtener la lista de provincias de la comunidad, con una imagen representativa de cada una,
- Obtener un listado de comarcas para una provincia dada, con una imagen representativa, y
- Obtener la información detallada sobre una comarca en concreto.

Para ello, nuestra aplicación recibirá varios argumentos por la línea de órdenes, actuando el primero como suborden que nos indica el tipo de consulta a realizar (provincias, comarcas e infocomarca), y el resto los argumentos que ésta necesita: el nombre de la provincia para obtener el listado de comarcas, o de la comarca para obtener la información detallada de la comarca. La obtención de la lista de provincias no necesitará ningún argumento adicional.

Veamos algunos ejemplos:

- Para obtener la lista de provincias, haremos:

```
dart run comarquescli provincies
```

- Para obtener la lista de comarcas de Castellón, haremos:

```
dart run comarquescli comarques Castelló
```

- Y con el fin de obtener información sobre la comarca de "La Ribera Baixa", haremos:

```
dart run comarquescli infocomarca La Ribera Baixa
```

Debemos tener en cuenta que si utilizamos nombres de comarca con apóstrofos, habrá que escapar estos. Por ejemplo:

```
dart run comarquescli infocomarca L\'alcoià
```

Consideraciones

Api de comarques

Para obtener la información sobre las provincias y comarcas haremos uso de la API REST que hemos presentado a la unidad. Concretamente, las rutas:

- https://node-comarques-rest-server-production.up.railway.app/api/comarques/provincies-
- https://node-comarques-rest-server-production.up.railway.app/api/comarques/comarquesAmbImatge/$provincia
- https://node-comarques-rest-server-production.up.railway.app/api/comarques/infoComarca/$comarca

Siendo `$provincia` y `$comarca` la provincia y la comarca sobre la que queremos pedir información.

Proyecto de base

La estructura básica del proyecto que ya se os proporciona como base es la siguiente:

```
.
├── bin
│   └── comarquescli.dart
├── lib
│   ├── comarca.dart
│   ├── comarques_service.dart
│   └── provincia.dart
...
```

- El fichero principal `bin/comarquescli.dart` contiene la funcionalidad principal de la aplicación: recoge lo que le proporcionamos por la línea de órdenes y hace uso del resto de clases y funciones para mostrar los resultados.
- Los ficheros `lib/comarca.dart` y `lib/provincia.dart` contienen las clases Comarca y Provincia respectivamente, que detallaremos a continuación.
- El fichero `lib/comarques_service.dart` contiene la clase ComarcasService, que contiene métodos estáticos (con el fin de no tener que instanciar la clase) con las peticiones al servicio web.

La funcionalidad correspondiente a la parte de las provincias ya se os da implementada a modo de ejemplo, de manera que habrá que implementar la funcionalidad correspondiente a las comarcas.

La clase Provincia

La clase Provincia que ya se os proporciona implementada, contiene dos atributos de tipo String: el nombre y la imagen (opcional):

```
class Provincia {
  late String nom; // Declarem el nom, i indiquem que l'inicialitzarem després
  String? imatge; // La url a la imatge és nul·lable
...}
```

Ésta contiene un constructor por defecto, con argumentos con nombre, y un constructor con nombre, para crear la provincia a partir del JSON:

```
 /*
  Constructor amb arguments amb nom:
   - nom és obligatori, i
   - imatge opcional.
  */
  Provincia({
    required this.nom,
    this.imatge,
  });

  /*
  Constructor amb nom a partir d'un diccionari.
  */

  Provincia.fromJSON(Map<String, dynamic> objecteJSON) {
    nom = objecteJSON["provincia"] ?? "";
    imatge = objecteJSON["img"] ?? "";
  }
```

La Clase Comarca

Habrá que implementar una clase Comarca, que guardará la información de la comarca cuando se hace una consulta.

Esta clase tendrá las siguientes propiedades:

- comarca, de tipo String, que será el nombre de la comarca,
- capital, de tipo String,
- poblacio, de tipo String, que contendrá la cantidad de habitantes. A pesar de tratarse de una cantidad, ya que la API nos lo proporciona en formato String,
- img, de tipo String,
- desc, de tipo String,
- latitud, de tipo double,
- longitud, de tipo double.

Todas las propiedades de la clase, salvo `comarca` podrán ser nulas.

Esta clase soportará dos constructores:

- Un **constructor `Comarca` con argumentos con nombre**, donde todos serán opcionales, salvo la comarca (requerido).
- Un **constructor con nombre `Comarca.fromJSON`**, que recibirá un JSON a partir del cual se inicializará.

Además, se sobreescribirá el método toString, para devolver un String con la información de la comarca formateada, de la siguiente forma:

```
Comarca:        La Ribera Baixa
Capital:        Sueca
Poblacio:       78.070

Imatge:         https://upload.wikimedia.org/...
Desc:           La Ribera Baixa és una comarca valencianoparlant...
Coordenades:    (39.2025604, -0.3111645)
```

Cuando se reciba respuesta a la petición HTTP pidiendo información sobre una comarca, deberemos crear un objeto de la clase `Comarca`. Posteriormente, cuando vamos a mostrar el resultado, haremos uso del método `toString` que hemos sobreescrito en esta clase.

Acceso al servicio

La clase **`ComarcasService`** será la encargada de proporcionar, mediante métodos estáticos el acceso al servicio web, y devolver las listas o los objetos requeridos por la aplicación principal.

Esta clase implementará los métodos:

- `Future<List<Provincia>> obtenerProvincias()`: Que devuelve un *Future* que se resolverá en una lista de objetos de tipo *Provincia*, generado a partir de lo que nos devuelve la petición web a la ruta `/api/comarcas`. **Este método ya se os proporciona implementado**.
- `Future<List<dynamic>> obtenerComarcas(String provincia)`: Que devolverá un *Future* que se resolverá en una lista de objetos dinámicos a partir de lo que nos devuelve la petición web a la ruta `api/comarcas/comarcasAmbImagen/$provincia`. Ten en cuenta que, en este método, a diferencia del anterior, no es necesario hacer ninguna conversión del resultado de la petición web; directamente, devolveremos el JSON que nos devuelve.
- `Future<Comarca?> infoComarca(String comarca)`: Que devolverá un *Future* con un objeto de tipo Comarca, generado a partir de la petición web a la ruta `/api/comarcas/infoComarca/$comarca`. En este caso, sí habrá que hacer una conversión del objeto JSON que recibimos a un objeto de tipo *Comarca*.

El fichero principal comarcascli.dart. Tratamiento de argumentos

En el fichero principal del proyecto de base ya se realiza la captura de argumentos y se decide cuál es el orden o la acción a realizar a través del primer argumento (provincias, comarcas o infocomarca). El resto de argumentos, representarán bien el nombre de la comarca o el de la provincia, según el caso.

Para obtener el orden y el resto de argumentos, lo que hacemos es obtener una copia de la lista de argumentos, guardarnos el primero como la orden, lo eliminamos, y combinamos (con join) el resto de argumentos para obtener el nombre de comarcas compuestas (por ejemplo "La Ribera Baixa").

```
String? ordre;
String? args;

// Creem una còpia de la llista d'arguments del programa
List<String> llistaArgs = List.from(arguments);

// L'ordre és el primer argument
ordre = llistaArgs[0];

// Eliminem l'ordre
llistaArgs.removeAt(0);

// I reconstruim la resta d'arguments com un String,
// separant-los amb un espai.
args = llistaArgs.join(" ");
```

El hecho de hacer una copia de la lista de argumentos y no trabajar directamente con estos es para poder utilizar el método `removeAt()` de la clase `List`, lo cual no se puede hacer directamente con la lista de argumentos en el programa.

Después de esto, el programa discrimina qué queremos hacer con un switch e invoca la función apropiada para ello, comprobando y proporcionándole los argumentos que necesita:

```
switch (ordre) {
    case "provincies":
      mostraProvincies();        // Implementació amb Future
      // mostraProvinciesSync(); // Implementació amb async/await
      break;

    case "comarques":
      ...
```

```
      mostraComarques(args);
      break;

    case "infocomarca":
      ...
      mostraInfoComarca(args);
      break;
    default:
      print("\x1B[31mOrdre desconeguda\x1B[0m");
  }
}
```

Las funciones `mostraProvincies` y `mostraProvinciesSync` ya se os dan implementadas, y son las que realizan la función de obtener y mostrar las provincias, cada una de una forma diferente. `mostraProvincies()` hace uso del tratamiento de Future, y `mostraProvinciesSync()` lo hace haciendo uso de `async/await`.

Vuestra tarea aqui será implementar las funciones `mostraComarcas()` y `mostraInfoComarca()`, haciendo uso de uno u otro mecanismo.

Pintando la salida

Si deseamos pintar el ensañamiento, podemos hacer uso de los códigos de escape ANSI para los siguientes colores:

```
Negro:   \x1B[30m
Rojo:    \x1B[31m
Verde:    \x1B[32m
Amarillo:    \x1B[33m
Azul:    \x1B[34m
Magenta: \x1B[35m
Cyan:    \x1B[36m
Blanco:   \x1B[37m
Reset:   \x1B[0m
```

Por ejemplo, para imprimir un texto en rojo haríamos:

```
print("\x1B[31mTexto en rojo\x1B[0m");
```