# PSP - Unidad 4 - Examen

Índice de contenido:



---

# 1. Consulta de códigos postales

Crea un programa usando las clases estudiadas en esta unidad para realizar consultas de los datos de **códigos postales en España** atacando a esta API:

https://api.zippopotam.us/

El funcionamiento de esta API es sencillo. La URL base es:

https://api.zippopotam.us/es/

Y para consultar los datos de cualquier código postal, sólo hay que añadirlo al final. Por ejemplo:

https://api.zippopotam.us/es/**04006**

Dará la siguiente información:

```
{
  "post code": "04006",
  "country": "Spain",
  "country abbreviation": "ES",
  "places": [
    {
      "place name": "Almería",
      "longitude": "-2.4508",
      "state": "Andalucia",
      "state abbreviation": "AN",
      "latitude": "36.8353"
    }
  ]
}
```

El **rango de códigos postales** para España está entre **01001** y **52080** (ambos incluidos).

# 2. Diseño del programa

Tu programa funcionará en **modo consola**. Solicitará al usuario que introduzca un código postal y mostrará los detalles de dicho código.

Ejemplo:

```
(pide un código)
Introduzca un código postal: 04006

(muestra los resultados)
Datos del código postal 04006:
- País: Spain (ES)
- Ubicaciones:

    Lugar: Almería
    Latitud: 36.8353
    Longitud: -2.4508
    Está en Andalucia (AN).
    ——————————————————————————————————
```

## Estructura de clases

Tu programa tendrá varias clases:

- Una clase **`Main`** que lanza el programa.
  Esta clase solicitará códigos postales **de forma ininterrumpida** hasta que el usuario introduzca un guión (-), que significa que el programa finaliza.
- Una clase **`PostalCode`** donde implementamos la funcionalidad.
- Las clases necesarias para mapear el json de respuesta.

## Consideraciones

La clase **`PostalCode` puede incluir los métodos y atributos que creas oportunos**, pero debe incluir obligatoriamente el método público **`printInfoForCode(postalCode)`**. Dicho método se encargará de:

- Atacar a la API con el código postal introducido como parámetro.
- Recibir el json con la información correspondiente y convertirla en un objeto para su tratamiento.

  Para ello se puede usar la librería Gson de Google. La dependencia, en caso de usar Maven, sería la siguiente:

  ```
  <!-- https://mvnrepository.com/artifact/com.google.code.gson/gson
  -->
  <dependency>
      <groupId>com.google.code.gson</groupId>
      <artifactId>gson</artifactId>
  ```

```
    <version>2.12.1</version>
</dependency>
```

- Recibir la información e imprimirla por consola de forma organizada con el siguiente formato:

```
Datos del código postal <código postal>:
- País: <country>(<country abbreviation>)
- Ubicaciones:

    Lugar: <places/place name>
    Latitud: <places/latitude>
    Longitud: <places/longitude>
    Está en <places/state>(<places/state abbreviation>).
    ——————————————————————————————————
```

Para el ejemplo anterior (código postal 04006, el programa debería mostrar lo siguiente en la consola:

```
Datos del código postal 04006:
- País: Spain (ES)
- Ubicaciones:

    Lugar: Almería
    Latitud: 36.8353
    Longitud: -2.4508
    Está en Andalucia (AN).
    ——————————————————————————————————
```

- Debes validar la petición de códigos postales fuera de rango. En caso de códigos postales fuera de rango, que el programa muestre por pantalla un mensaje indicándolo.
- Debes validar el estado de la petición (si la petición http ha sido correcta o no). En caso de no serla, hay que mostrar un mensaje en pantalla indicando el código de error.
- Debes contemplar las distintas excepciones que puedan ocurrir al atacar a la API.

# 3. Parsear json con Gson.

Recuerda que para usar esta librería sólo tienes que importar la clase Gson del paquete com.google.gson.

Para parsear el json recibido y convertirlo en un objeto java usando esta librería, recuerda que tienes que:

1. Crear una clase con la misma estructura que el json. La clase se llamará **`CPInfo`**.
2. Para parsear el json en un objeto, debes crear un objeto Gson y usar el método fromJson pasando el json en si y la clase que hayas implementado en el paso anterior.

   Ejemplo:

```
…
Gson gson = new Gson();
CPInfo info = gson.fromJson(cuerpoJson, CPInfo.class);
…
```

# 4. Criterios de evaluación (CE)

Este ejercicio permite evaluar el **RA 4. Desarrollar aplicaciones que ofrecen servicios en red, utilizando librerías de clases y aplicando criterios de eficiencia y disponibilidad** con los siguientes criterios:

| |
|---|
| 4.a) Se han analizado librerías que permitan implementar protocolos estándar de comunicación en red. |
| 4.b) Se han programado clientes de protocolos estándar de comunicaciones y verificado su funcionamiento. |
| 4.c) Se han desarrollado y probado servicios de comunicación en red. |
| 4.d) Se han analizado los requerimientos necesarios para crear servicios capaces de gestionar varios clientes concurrentes. |
| 4.e) Se han incorporado mecanismos para posibilitar la comunicación simultánea de varios clientes con el servicio. |
| 4.f) Se ha verificado la disponibilidad del servicio. |
| 4.g) Se han depurado y documentado las aplicaciones desarrolladas. |