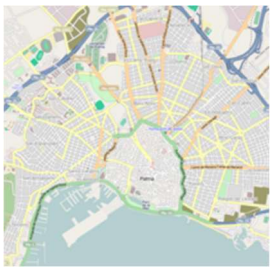
ENUNCIAT PROJECTE
Laboratori de Programació – UAB
curs 2023-24

1. INTRODUCCIÓ

OpenStreetMaps (OSM) és una organització sense ànim de lucre que distribueix dades de informació geogràfica (mapes de tot el món) obtingudes de forma col·laborativa i oberta en format XML.

[1]



En el següent enllaç podeu descarregar informació en aquest format de petites regions d'un mapa del món qualsevol:

https://www.openstreetmap.org/#map=18/41.23522/1.80830

La pràctica de Laboratori de Programació d'aquest curs es basa en la representació dels punts d'interès d'un mapa proporcionat per OSM. Partirem d'un fitxer XML a on s'inclou la informació de tots els punts d'interès i les carreteres que tenim a una secció d'un mapa real. Les carreteres es representen com a una llista de nodes connectats amb unes coordenades en forma de latitud i longitud. El primer que farem serà llegir aquest fitxer i guardar la informació que surt en ell. Després haureu de representar punts d'interès sobre un mapa, així com els camins de carretera, i representareu el camí més curt entre dos punts d'interès seleccionats a la interfície gràfica que us proporcionarem. El punt clau del projecte consistirà en dissenyar i implementar les estructures de dades més adients per desar i accedir a la informació requerida de forma òptima.

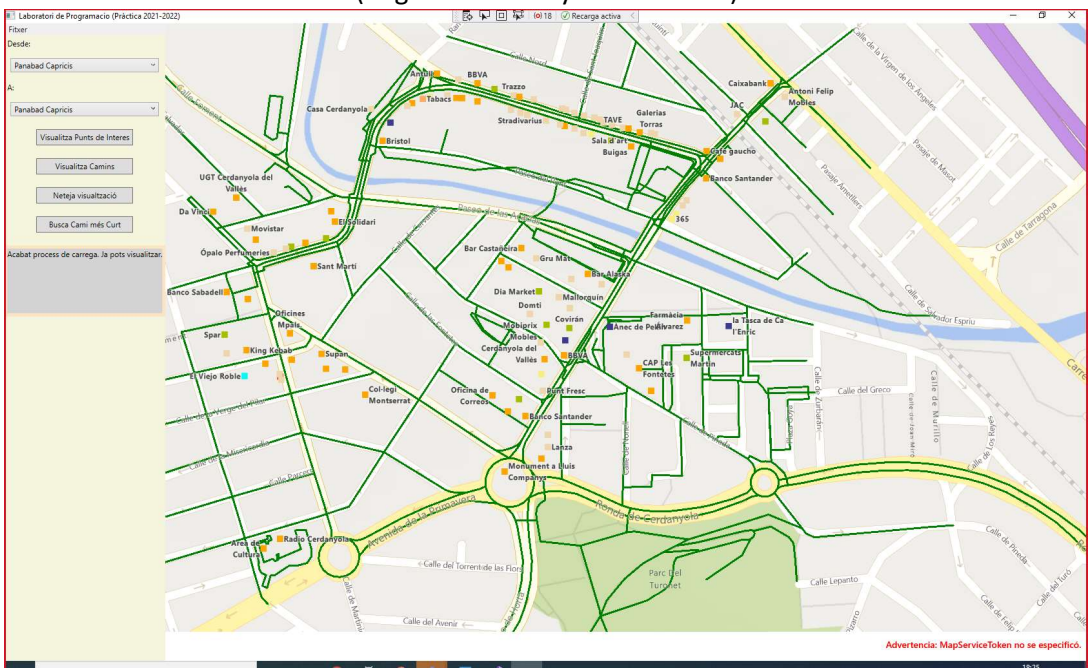
2. PROJECTE PART 1

La primera part de la pràctica consisteix en obrir un fitxer XML dels proporcionats per OpenStreetMaps (OSM), llegir-lo i desar la seva informació a les vostres estructures de dades. Aquesta informació seran els nodes que formen les carreteres i els punts d'interès.

El resultat vist dins la interfície gràfica que us donem seria quelcom semblant a la següent imatge:

[1] Imatges CC descarregades de Wikipedia: https://ca.wikipedia.org/wiki/OpenStreetMap

(Regió de Cerdanyola del Vallés)



Per assolir aquesta part heu de tenir en compte les instruccions donades dins aquest document. Al llarg de les següents consideracions us guiarem en les tasques que heu de desenvolupar. Fixeu-vos en que la tasca 0 i la tasca 1 no estan relacionades directament amb la solució de la pràctica, sinó que serveixen per entendre com està estructurada, i es faran amb un codi a part.

2.0 Fitxer XML de OpenStreetMaps

Per començar necessitem entendre el **fitxer XML** que us ve proporcionat per **OSM.**

El fitxer XML OSM consisteix en una sèrie d'elements, on us haureu de centrar amb els definits com a ***<node>*** i ***<way>.***

- ✓ Un ***<node>*** representa un punt a l'espai, i defineix la seva latitud i longitud, així com també un identificador, els conformen les dades fonamentals.
- ✓ Un ***<way>*** és un conjunt de nodes, però només tractarem un subconjunt d'aquests.

Seguidament, us detallem el què podeu trobar a dins d'un fitxer XML en detall:

- **Capçalera:** Aquesta capçalera no la necessitareu per la pràctica.

```
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="CGImap 0.8.5 (3025176 spike-
 07.openstreetmap.org)" copyright="OpenStreetMap and contributors"
 attribution="http://www.openstreetmap.org/copyright"
 license="http://opendatacommons.org/licenses/odbl/1-0/">
 <bounds minlat="41.5004500" minlon="2.1003300" maxlat="41.5043900"
 maxlon="2.1089500"/>
```

- **Nodes:** Els nodes poden ser de 2 tipus, dels quals els nodes que són camí poden ser de tres subtipus. Així doncs tenim la següent divisió:

    - Nodes que són nodes de camí
        - Nodes que no tenen tag.
        - Nodes que tenen un tag amb un atribut *highway*, *public_transport*, *access o entrance*.
        - Nodes amb tags on cap té l'atribut *name*.

    - Nodes que són Punts d'Interès

Els nodes que són Punt d'Interés són nodes que no són de camí, i que tenen un tag amb un atribut *name*. El seu valor és el que es visualitzarà al mapa com a identificació del punt.

Seguidament procedim a mostrar certs exemples de nodes de camí (fitxeu-vos en el que hi està marcat en negreta):

Node sense tags (node de camí):

```
<node id="1644458226" visible="true" version="1"
 changeset="10769439" timestamp="2012-02-23T17:43:24Z" user="EliziR"
 uid="605366" lat="41.4992770" lon="2.1048901"/>
```

Node amb tags sense cap atribut *name* (node de camí):

```
<node id="522572539" visible="true" version="3" changeset="10769439"
 timestamp="2012-02-23T17:43:34Z" user="EliziR" uid="605366"
 lat="41.4993094" lon="2.1048279">
  <tag k="barrier" v="lift_gate"/>
 </node>
```

Node amb el tag *highway*, *public_transport* , *entrance* o *access* (nodes de camí):

```
<node id="354873898" visible="true" version="8"
 changeset="103869767" timestamp="2021-04-29T21:04:44Z"
 user="caminsperfer" uid="10187213" lat="41.9662078"
 lon="2.7402486">
  <tag k="highway" v="traffic_signals"/>
 </node>

<node id="1287613323" visible="true" version="2"
 changeset="56244875" timestamp="2018-02-10T16:59:23Z"
 user="jqngarcia" uid="5126253" lat="41.5369864" lon="2.4303634">
  <tag k="bus" v="yes"/>
  <tag k="name" v="I. Català de Salut"/>
  <tag k="public_transport" v="stop_position"/>
 </node>

<node id="8647042980" visible="true" version="2"
 changeset="103643101" timestamp="2021-04-26T14:15:46Z"
 user="Interswd" uid="3414856" lat="41.4928225" lon="2.1422926">
  <tag k="access" v="yes"/>
  <tag k="emergency" v="defibrillator"/>
  <tag k="indoor" v="no"/>
  <tag k="name" v="AED"/>
  <tag k="opening_hours" v="24/7"/>
 </node>
```

```
<node id="2386533013" visible="true" version="3"
 changeset="64697386" timestamp="2018-11-20T13:22:48Z"
 user="Carto'Cité" uid="5426135" lat="48.8571334" lon="2.2908746">
  <tag k="entrance" v="main"/>
  <tag k="level" v="0"/>
  <tag k="name" v="Quai de Grenelle"/>
  <tag k="railway" v="train_station_entrance"/>
  <tag k="source" v="SNCF Transilien - Plans d'architecte et plan de
 communication - Prestation de saisie réalisée par Arx IT en hiver
 2016"/>
 </node>
```

Finalment, per finalitzar la identificació de nodes, mostrem el que seria un node Punt d'Interès:

Nodes amb el *<tag> name*, i sense tag *highway*, *public_transport*, *access* ni *entrance*. Aquest node és un punt d'interès que s'identifica visualment com la Facultat de Filosofia i Lletres.

```
<node id="1655719923" visible="true" version="7"
 changeset="57828127" timestamp="2018-04-05T09:22:58Z" user="Jesús
 Gómez" uid="6389" lat="41.5028375" lon="2.1075584">
  <tag k="name" v="Facultat de Filosofia i Lletres"/>
  <tag k="office" v="educational_institution"/>
  <tag k="website" v="https://www.uab.cat/lletres/"/>
 </node>
```

- **Way:** Indica un camí que passa per nodes que no són Punts d'Interès. Els fills de l'element ***<way>*** determinen els nodes que formen el camí, on cada node disposa de la informació relativa de latitud i longitud, i es referencia a través de l'atribut ref.

NOTA: Per a la realització d'aquesta pràctica, descartarem tots els camins que no contenen el tag ***highway***.

Seguidament, mostrem un exemple del que seria un element ***<way>*** que ens trobarem a les pràctiques, amb els nodes que el formen identificats per l'atribut ref. Fixeu-vos que els dos primers identificadors dels nodes que apareixen en el way aquí presentat (**"522572539" , "1644458226"**) són els dos primers nodes que han aparegut com a primers exemples a l'apartat de nodes.

```
<way id="151708041" visible="true" version="10" changeset="54933536"
 timestamp="2017-12-26T16:13:54Z" user="FvGordon" uid="161619">
  <nd ref="522572539"/>
  <nd ref="1644458226"/>
  # ...
  <tag k="foot" v="yes"/>
  <tag k="highway" v="service"/>
  <tag k="service" v="parking_aisle"/>
  <tag k="wheelchair" v="limited"/>
 </way>
```

Tot el conjunt de característiques que OSM defineix, el podeu trobar a Map features - OpenStreetMap Wiki.

Tasca 0: Anar a https://www.openstreetmap.org/export, exportar una regió d'un poble de manera manual, i observar-ne el contingut de *<node>* i *<way>* que el formen.

2.1 Estructura XmlElement

La interfície que us donem feta llegeix el fitxer XML utilitzant una llibreria dinàmica externa en C++ (xerces-c_3_2D.dll), i desa tota la informació dins un vector STL. El vector estarà format per `XmlElement` (definit a Common.h). Un `XmlElement` guardarà la informació dels nodes i els camins del fitxer i està definit de la següent manera:

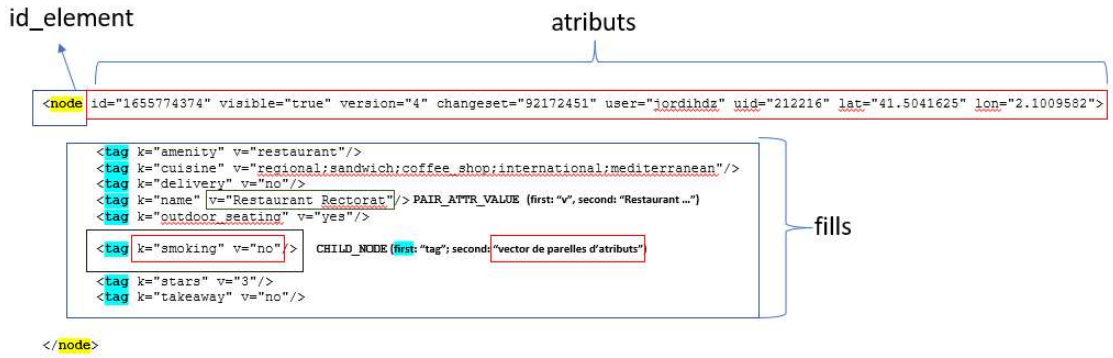
```
typedef std::pair<std::string, std::string> PAIR_ATTR_VALUE;
typedef std::pair<std::string, std::vector<PAIR_ATTR_VALUE>> CHILD_NODE;


typedef struct {
    std::string id_element;
    std::vector<PAIR_ATTR_VALUE> atributs;
    std::vector<CHILD_NODE> fills;
} XmlElement;
```

Aquesta estructura consisteix en el següents tres camps:

- **`id_element`:** defineix el tipus d'element
- **`atributs`:** és un vector de parelles d'atributs d'aquest element
- **`fills`**: És el seu conjunt de **sub elements**.
    - Cada sub-element, consisteix en una parella on el seu primer valor identifica el tipus d'aquest element, i el segon valor un vector d'atributs, d'aquest sub-element.

Representem seguidament en un diagrama, fent una comparativa amb el que us trobareu en un fitxer d'OSM, identificant cadascuna de les variables.



Per donar un exemple de com seria un punt d'interès anomenat "El Roble", que es tracta d'un restaurant que ofereix cuina regional i facilita l'accés a persones amb mobilitat reduïda, l'element `XmlElement` es construiria de la següent manera:

```
<node id="357101444" visible="true" version="3" changeset="77532251"
 timestamp="2019-11-25T15:26:03Z" user="MultiDavid2001b"
 uid="4946691" lat="41.4902204" lon="2.1406477">
  <tag k="amenity" v="restaurant"/>
  <tag k="cuisine" v="regional"/>
  <tag k="name" v="El Viejo Roble"/>
  <tag k="wheelchair" v="yes"/>
 </node>
```

```
using namespace std;
XmlElement viejoRoble = {
	//id_element =
	"node",

	// atributs =
	{
		make_pair("id", "357101444"),
		make_pair("visible", "true"),
		make_pair("version", "3"),
		make_pair("changeset", "77532251"),
		make_pair("timestamp", "2019-11-25T15:26:03Z"),
		make_pair("user", "MultiDavid2001b"),
		make_pair("uid", "4946691"),
		make_pair("lat", "41.4902204"),
		make_pair("lon", "2.1406477")
	},

	// fills =
	{
		make_pair<string,vector<pair<string, string>>>
			("tag", {
				make_pair<string, string>("k", "amenity"),
				make_pair<string, string>("v", "restaurant")
		}),
		make_pair<string, vector<pair<string, string>>>
				("tag", {
					make_pair<string, string>("k","cuisine"),
					make_pair<string, string>("v", "regional")
		}),
		make_pair <string, vector<pair<string, string>>>
				("tag", {
					make_pair<string, string>("k", "name"),
					make_pair<string, string>("v", "El Roble")
				}),
		make_pair <string, vector<pair<string, string>>>
				("tag", {
					make_pair("k","wheelchair"),
					make_pair("v", "yes")
				})
	}
};
```

Vosaltres no us haureu de preocupar de construir dades de XmlElement per la presentació de la pràctica, sinó de llegir-les, però és important entendre com es construeix/forma per poder llegir la informació que hi guarda de manera correcte.

Per comprovar que efectivament es tracta d'un node, llegirem el camp `id_element`, i en aquest cas, emmagatzemarem en una variable el valor del atribut `timestamp`, així com setejarem en un booleà si el node facilita l'accés a persones amb mobilitat reduïda o no.

Per llegir-ne el valor del tag, utilitzarem la funció kvDeTag de la classe Util que se us explicarà més endavant.

Seguidament us mostrem el codi exemple basant-se amb la construcció de l'anterior:

```
string tempsValor = "";
bool permetMobilitatReduida = false;
// Comprovem per id_element, que es tracta d'un node
if (viejoRoble.id_element == "node") {
	// En cas que sigui un node, recorrem els seus atributs
	for (int i = 0; i < viejoRoble.atributs.size(); i++) {
		// Iterem fins trobar el atribut que ens interessa
		if (viejoRoble.atributs[i].first == "timestamp") {
			tempsValor = viejoRoble.atributs[i].second;
		}
	}
	// Recorrem els fills d'aquest node
	for (int i = 0; i < viejoRoble.fills.size(); i++) {
		// Iterem i avaluem tots els fills que son tags
		if (viejoRoble.fills[i].first == "tag") {
			// Emmagatzemem el valor d'aquest tag
			pair<string, string> valorTag =
				Util::kvDeTag(viejoRoble.fills[i].second);
			// Comprovem que es el tag que busquem
			if (valorTag.first == "wheelchair") {
				permetMobilitatReduida =
					(valorTag.second == "yes")? true : false;
			}
		}
	}
}
```

Tasca 1: Imprimir per pantalla l'identificador, latitud (***lat***), l'longitud (***lon***), així com el nom (***name***), del element XML `viejoRoble` construit en aquest apartat 2.2.

Teniu en compte que, almenys, haureu de transformar latitud i longitud a tipus

`double`.

Recordeu que en la solució de la pràctica haureu de recollir tots els punts d'interès, que per definició, són nodes que **no** són nodes de camí que tenen un atribut ***name***.

Implementa aquesta Tasca 1 al projecte Visual Studio LP-OSM-Tasca1-i-2.sln

En el cas dels camins, se'ns presentaran de la següent forma:

```
<way id="31889094" visible="true" version="6" changeset="102251634"
 timestamp="2021-04-03T21:17:17Z" user="Interswd" uid="3414856">
  <nd ref="357125056"/>
  <nd ref="8591278795"/>
  <nd ref="357124693"/>
  <nd ref="357125054"/>
  <tag k="highway" v="living_street"/>
  <tag k="name" v="Carrer de Sant Joaquim"/>
  <tag k="oneway" v="yes"/>
  <tag k="surface" v="paving_stones"/>
 </way>
```

On, els nodes que formen part del way s'haurien llegit prèviament i el seu XmlElement, estaria construït de la següent manera:

```
XmlElement camiJoaquim = {

	// id_element =
	"way",

	// atributs =
	{
		make_pair("id", "31889094"),
		make_pair("visible", "true"),
		make_pair("version", "6"),
		make_pair("changeset", "1002251634"),
		make_pair("timestamp", "2021 - 04 - 03T21:17 : 17Z"),
		make_pair("user", "Interswd"),
		make_pair("uid", "3414856")
	},

	// fills =
	{
		make_pair<string, vector<pair<string, string>>>
			("nd", {make_pair("ref", "357125056")}),
		make_pair< string, vector<pair<string, string>>>
			("nd", {make_pair("ref", "8591278795")}),
		make_pair< string, vector<pair<string, string>>>
			("nd", {make_pair("ref", "357124693")}),
		make_pair< string, vector<pair<string, string>>>
			("nd", {make_pair("ref", "357125054")}),
		make_pair<string, vector<pair<string, string>>>
			("tag", {
				make_pair("k", "highway"),
				make_pair("v", "living_street")
			}),
		make_pair < string, vector<pair<string, string>>>
			("tag", {
				make_pair("k", "name"),
				make_pair("v", "Carrer de Sant Joaquim")
			}),
		make_pair <string, vector<pair<string, string>>>
			("tag", {
```

```
                        make_pair("k", "oneway"),
                        make_pair("v", "yes")
                }),
        make_pair <string, vector<pair<string, string>>>
                ("tag", {
                        make_pair("k", "surface"),
                        make_pair("v", "paving_stones")
                })
    }
};
```

Tasca 2: Imprimir per pantalla els identificadors dels nodes que formen el `XmlElement camiJoaquim`, així com el valor del atribut ***highway***. Implementa aquesta tasca a la solució

Recordeu que en la solució de la pràctica només haureu de tenir en compte tots els camins que tenen el tag ***highway*** (independentment del valor que pugui tenir), així com saber quins nodes de tipus camí, formen cada camí.

Implementa aquesta Tasca 2 al projecte Visual Studio LP-OSM-Tasca1-i-2.sln

2.2 Classe PuntDeInteres

Haureu de definir diferents classes que defineixin diferents punts d'interès, mitjançant l'herència i definició d'atributs i mètodes addicionals de la classe **PuntDeInteresBase**. Haureu de definir un nom i color, que serà el que es veurà representat en la interfície gràfica. Sou lliures de definir el nom que vulgueu, tanmateix, us recomanem que captureu el que us proporciona les dades OSM de l'atribut "name" dels nodes d'interès.

Per altra banda, per la definició del color, haureu d'escollir certs atributs extres de **PuntDeInteresBase**, que defineixin el punt, i definir-li un color en format hexadecimal (com 0xFFFFFF o 0xFFA500).

Per donar dos exemples, podeu definir una classe Restaurant representant en color blau els que serveixen ***pizza***:

```
<node id="8606512476" visible="true" version="1"
 changeset="102562605" timestamp="2021-04-08T12:04:11Z"
 user="Interswd" uid="3414856" lat="41.4928550" lon="2.1426380">
  <tag k="amenity" v="restaurant"/>
  <tag k="cuisine" v="pizza"/>
  <tag k="name" v="El Colosseo"/>
 </node>
```

O definir una classe PuntDeInteresBotiga, que tenen el tag ***shop*** que faciliten l'accés a gent que va en cadira de rodes:

```
<node id="2561518617" visible="true" version="2"
 changeset="48212270" timestamp="2017-04-28T00:25:16Z" user="Sioul"
 uid="832276" lat="41.4905682" lon="2.1447148">
  <tag k="addr:city" v="Cerdanyola del Vallès"/>
  <tag k="addr:housenumber" v="16"/>
  <tag k="addr:postcode" v="08290"/>
  <tag k="addr:street" v="Carrer González,"/>
  <tag k="name" v="Mobiprix Mobles Cerdanyola del Vallès"/>
  <tag k="operator" v="Mobiprix"/>
  <tag k="phone" v="+34 902 104 168"/>
  <tag k="shop" v="furniture"/>
  <tag k="website" v="http://www.mobiprix.com"/>
  <tag k="wheelchair" v="yes"/>
 </node>

<node id="4448289429" visible="true" version="5"
 changeset="102250867" timestamp="2021-04-03T20:47:04Z"
 user="Interswd" uid="3414856" lat="41.4929978" lon="2.1423720">
  <tag k="name" v="Casa Cerdanyola"/>
  <tag k="opening_hours" v="Mo-Fr 09:00-14:00,16:00-21:00, Sa 09:00-
 14:30,16:00-21:00"/>
  <tag k="shop" v="hardware"/>
  <tag k="wheelchair" v="yes"/>
 </node>
```

Tant el nom com el color escollits, hauran de ser retornats mitjançant la implementació dels mètodes en la classe derivada:

```
virtual std::string getName();
virtual unsigned int getColor();
```

Tasca 3: Creeu dos classes, una anomenada **PuntDeInteresBotigaSolucio** i una altra anomenada **PuntDeInteresRestaurantSolucio** on implementen els mètodes getName() i getColor() de la classe **PuntDeInteresBase**.

El **PuntDeInteresBotigaSolucio** disposarà d'un atribut addicional que contindrà el valor del tag "shop", on:

1. Retornarà el color **0xA5BE00** els que són "supermarket"

2. Retornarà el color **0xFFAD69** els que són "tobacco"

3. Retornarà el color **0xE85D75** els que són "bakery"

4. Retornarà el color **0x4CB944** d'aquells "bakery" si tenen el "opening_hours" obert de "06:00-22:00" i tenen accés facilitat per persones amb mobilitat reduïda.

Nota: El valor pot ser "Div 06:00-22:00" o "Dill 06:00-22:00". Les dos entrades son valides. Utilitza el find de std::string.

5. Finalment, retornarà 0xEFD6AC en els altres casos (que tenen el tag "shop" però que el seu valor no és dels esmentats).

El **PuntDeInteresRestaurantSolucio** disposarà de dos atributs addicionals, el tipus de cuina ("cuisine"), i si en facilita l'accés a les persones amb mobilitat reduïda, on:

1. Retornarà el color **0x03FCBA** tots els restaurants que ofereixen "pizza" i faciliten l'accés a persones amb mobilitat reduïda.

2. Retornarà el color 0xA6D9F7 tots els restaurants que ofereixen "chinese" com a tipus de cuina.

3. Retorna el color **0x251351** tots els restaurants que faciliten l'accés a persones amb mobilitat reduïda

4. Retornarà el color definit per defecte en els altres casos.

Nota: El color per defecte ve definit per la classe PuntDeInteresBase, i pot ser canviat. En altres paraules, eviteu utilitzar el "valor de manera directe", i accediu al valor que té a través de la classe.

2.3 Classe virtual pura CamiBase

Finalment, per concloure la primera part de la pràctica, haureu d'implementar la solució per la definició dels camins.

Per assolir aquesta part, us oferim la classe **CamiBase**, que és una classe base pura virtual, on haureu d'implementar una classe derivada que implementarà el mètode getCamiCoords().

```
class CamiBase {
	public:
		virtual std::vector<Coordinate> getCamiCoords() = 0;
};
```

Aquest mètode retorna un vector de coordenades, on `Coordinate` és una estructura de dades amb dos camps, `lat` i `lon`, que defineix la posició en el mapa d'un node del camí.

La vostra classe CamiSolucio, haurà de derivar d'aquesta classe **CamiBase**, i haurà de formar part del vostre MapaSolucio.

Fent això, us serà més fàcil finalment retornar el vector de **CamiBase** (conjunt de tots els camins que formen el mapa), a través del mètode getCamins().

Tasca 4: Com a prova per visualitzar algun camí, creeu una classe anomenada **CamiSolucio** que implementi el mètode getCamiCoords() que retorni el següent conjunt de coordenades, definit per (lat, lon)

- (41.4928803, 2.1452381)
- (41.4929072, 2.1452474)
- (41.4933070, 2.1453852)
- (41.4939882, 2.1456419)

NOTA: Això s'haurà de canviar al final perquè no siguin uns punts fixes sinó que es puguin obtenir a partir de les dades.

2.4 Classe virtual pura MapaBase

Per poder visualitzar per pantalla els vostres resultats i consultes tenim creada la interfície d'una classe que es comunicarà amb la DLL que genereu. Aquesta classe és la **classe virtual pura MapaBase**, i conté les signatures dels diferents mètodes que haureu d'implementar.

Vosaltres haureu de crear una classe derivada de **MapaBase**, que es dirà **MapaSolucio** i que guardarà la seva informació com a estructures internes. Aquestes estructures les haureu de pensar, dissenyar i implementar perquè siguin òptimes, i guardaran la informació del fitxer xml (punts d'interès i camins formats per nodes).

Sobre la vostra classe derivada **MapaSolucio** haureu d'implementar les següents funcions:

- **`void getPdis(std::vector<PuntDeInteresBase*>& pdis)`**
  - Retorna tot el conjunt de punts d'interès del mapa (supermercats, pastisseries, escoles, comissàries...).

Tasca 5: Implementeu el mètode getPdis(std::vector<PuntDeInteresBase*>&) mitjançant la creació d'una classe derivada de MapaBase anomenada MapaSolucio, on:

1. Retorna dos punts d'interés, un **PuntDeInteresBotigaSolucio** i un **PuntDeInteresRestaurantSolucio**, que heu implementat a la **Tasca 3**.

2. **PuntDeInteresBotigaSolucio** es un ***shop*** amb valor "***bakery***" anomenat "La Millor Pastisseria" situada a (41.4918606, 2.1465411)

3. **PuntDeInteresRestaurantSolucio** es un ***amenity*** amb valor ***restaurant*** que ofereixen cuina ***regional*** i que en permet l'accés a persones amb mobilitat reduïda. Aquest restaurant està situat a (41.4902204, 2.1406477), i s'anomena

"El Millor Restaurant".

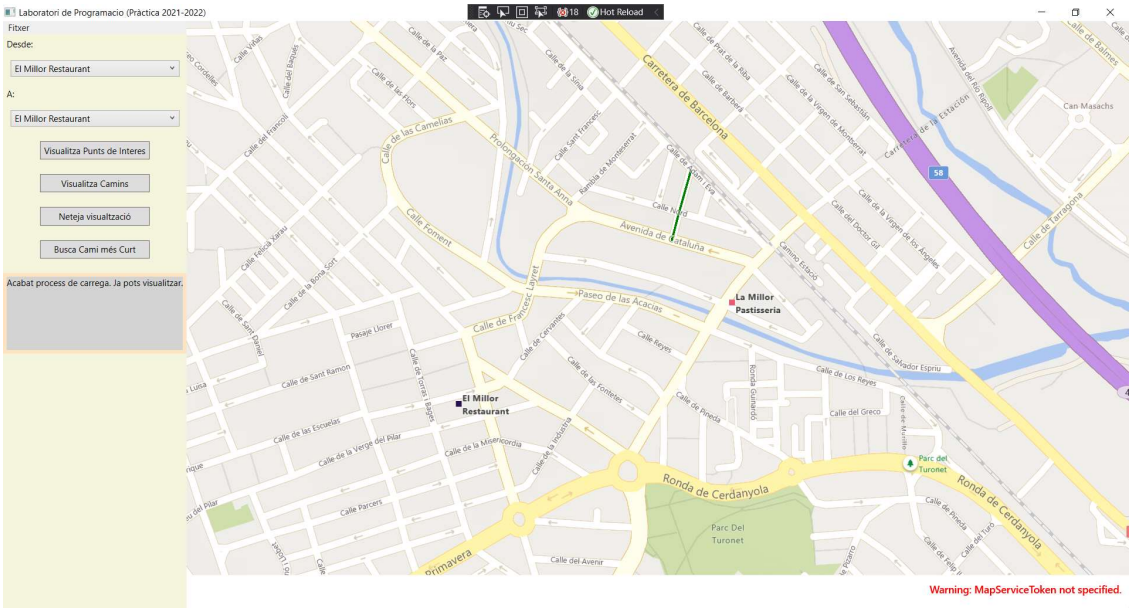
4. Instancieu correctament la propietat m_mapaBase de la classe **MapaRender** construint la vostra solució de **MapaSolucio**.

NOTA: Aquesta tasca més endavant haurà de retornar els punts d'interès del vector i no uns fixes.

- **`void getCamins(std::vector<CamiBase*>& camins)`**
  - Retorna tots els camins que haureu parsejat i que són **highway**.
  - Nota: La **classe CamiBase** és una classe virtual pura i necessitareu definir una classe derivada **`CamiSolució`** on afegireu nous atributs i funcionalitats.

Tasca 6: Retorneu el camí que heu implementat a la **Tasca 4** mitjançant l'implementació de getCamins(std::vector<CamiBase*>&)

Un cop realitzades les Tasques descrites fins aquí, ja podeu començar a visualitzar punts d'interès i camins per la interfície gràfica. Però encara no heu llegit cap fitxer OSM. Si heu implementat correctament els mètodes i atributs que us hem detallat, el vostre resultat hauria de representar dos punts d'interès amb un camí, en la regió de Cerdanyola:



- **`void parsejaXmlElements(std::vector<XmlElement>& xmlElements)`**
  - Aquesta funció rep tots els nodes del fitxer xml que ha llegit la classe **MapaRender**.
  - Cal que a partir d'analitzar cada un dels elements rebuts com a paràmetre, modifiqueu l'estructura interna que haureu dissenyat per desar aquesta informació.

Ara ja podem anar a utilitzar les dades reals d'un fitxer OSM concret:

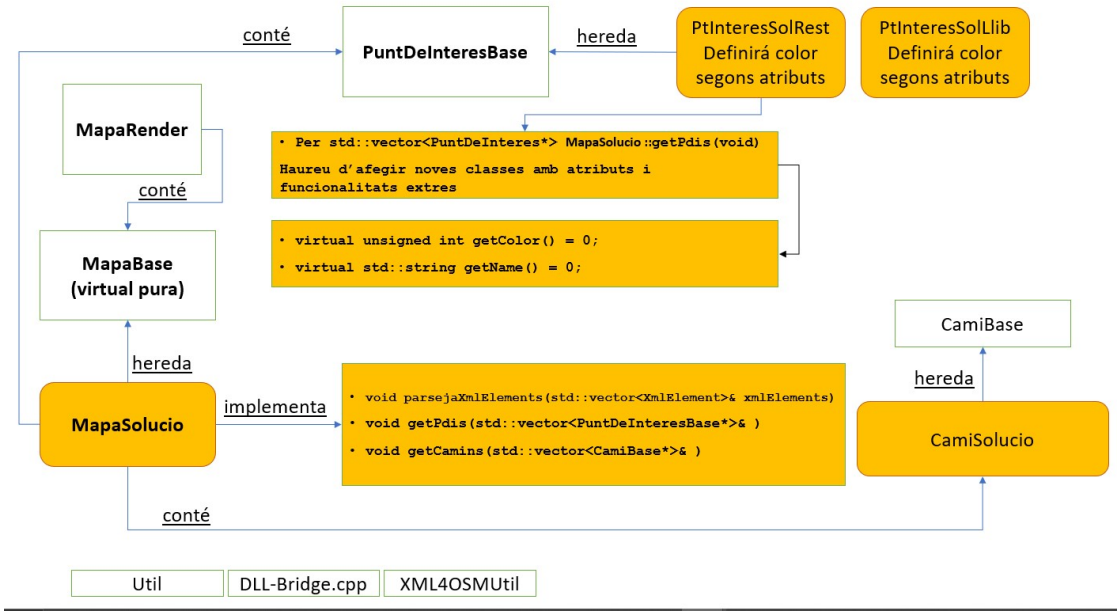
Tasca 7: Pensar les estructures de dades, atributs i mètodes interns de **MapaSolucio** i **CamiSolucio** que haurien de tenir per parsejar i emmagatzemar la informació de xmlElements a parsejaXmlElements(...) tenint en compte els exercicis realitzats a la **Tasca 1** i **Tasca 2**.

Tasca 8: Construir i retornar tots els Punts d'interès parsejats per xmlElements a través de getPdis(std::vector<PuntDeInteresBase*>&), utilitzant els colors descrits en la **Tasca 3**.

Tasca 9: Construir i retornar tots els camins parsejats per xmlElements a través de getCamins(std::vector<CamiBase*>&)

2.5 Diagrama de Classes

Aquí us facilitem seguidament un diagrama de classes per entendre com s'interrelacionen entre si. En taronja, hi ha el que haureu d'implementar per assolir els objectius de la primera part de la pràctica, en blanc el que ja us donem implementat.



2.6 Classe Util i fitxer Common.h

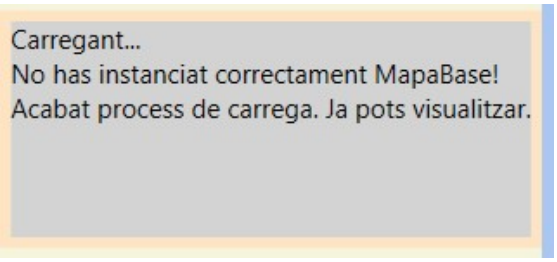
En el fitxer de capçalera Common.h, trobareu la definició de les estructures **Coordinate** i **XmlElement**, així com els tipus definits **PAIR_ATTR_VALUE i CHILD_NODE**, que us faran falta per

entendre com definim la longitud i la latitud dels que formen el mapa, així com podem iterar la informació del XML del OSM, tal com s'ha explicat en el punt **2.1 Estructura XmlElement**.

Els mètodes a destacar de la classe Util són els següents:

1. `Util::escriuEnMonitor(std::string text)`: Una funció util que us permetrà visualitzar pel monitor de la interfície el text que escriviu pel paràmetre ***text***.

Heu de tenir en compte que el text que us apareix el monitor es veurà entrellaçat amb informació que us ofereix la interfície gràfica.



En aquest cas, si no heu instanciat correctament MapaBase (cosa que succeeix amb el codi inicial), us apareixerà aquest missatge, a on es fa la següent crida:

```
Util::escriuEnMonitor("No has instanciat correctament MapaBase!");
```

2. `Util::kvDeTag(std::vector<PAIR_ATTR_VALUE>& atributsTag)` : A través de **kvDeTag** us permet llegir els sub-elements de tipus **<tag>** de manera fàcil.

Aquesta funció se'n encarrega d'iterar tots els atributs del tag (és a dir, `k="clau" v="valor"`), i en retorna un parell on el primer valor es "clau" i el segon "valor". Com a paràmetre, heu de facilitar-li tots els atributs del tag que esteu computant. Torneu a fer una ullada a l'exemple donat just abans de la descripció de la Tasca 1.

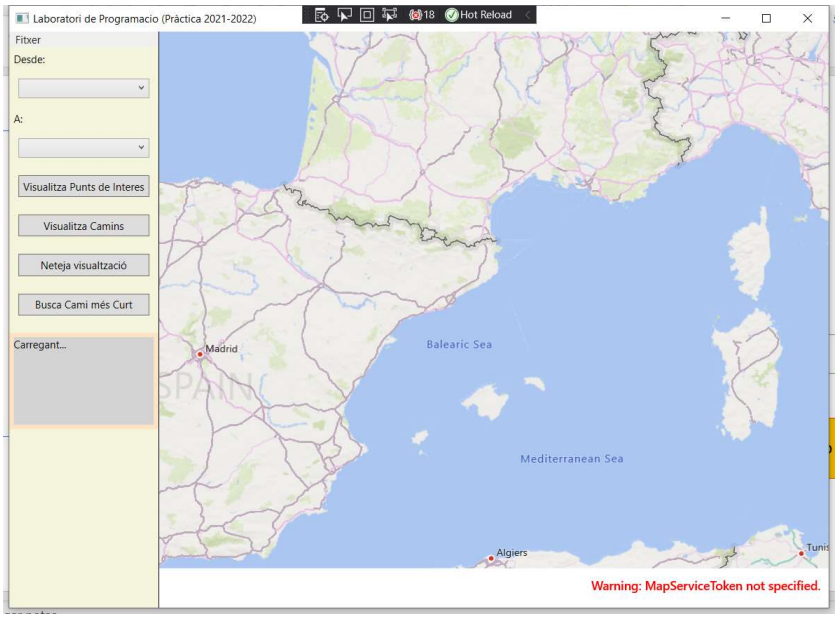
2.7 Consideracions finals: Visió tècnica interna i consells varis

La solució en Visual Studio 2019 del projecte presentada aquest curs forma part de dos projectes, un escrit en C# .NET (LP-OSM-2122-WPF), i un altre que és el que treballareu vosaltres, en C++ (LP-OSM-2122-DLL).

La comunicació entre els dos projectes es realitza mitjançant una llibreria dinàmica (DLL). A diferència de com hi esteu acostumats, aquest any el vostre projecte genera una llibreria que s'integra automàticament dins del projecte gràfic, oferint les implementacions a través de les crides dels mètodes definits a DLL-Bridge.

Degut a això, és important saber que no podreu utilitzar tal com esteu habituats el std::cout de la STL per imprimir valors en pantalla, de manera que us recomanem que us familiaritzeu amb el Debugger per qualsevol prova que esteu realitzant, o utilitzar la funció Util **escriuEnMonitor**.

No heu de tocar res del codi DLL-Bridge.cpp, ni de cap altre classe que no s'ha especificat en l'anterior guió. Tant mateix, seguidament us expliquem en detall com es troba implementada:



La interfície gràfica es pot dividir bàsicament en dues parts:

1. Càrrega d'informació: Es fa a través de Fitxer > Obrir
2. Representa informació carregada: A través dels botons de visualització.

Quan carreguem la informació, LP-OSM-2122-WPF comença un thread (a través del BackgroundWorker_DoWork), on crida el mètode loadOSMData(), que es troba definida al DLL-Bridge.cpp.

Aquesta funció crida del **MapaRender** el mètode construeixOSM(), on parseja el XML mitjançant parseXML de la classe **XML4OSMUtil**, que simplement utilitza una llibreria externa anomenada Xercesc que parseja el XML, i formateja el resultat com a un vector de XmlElements.

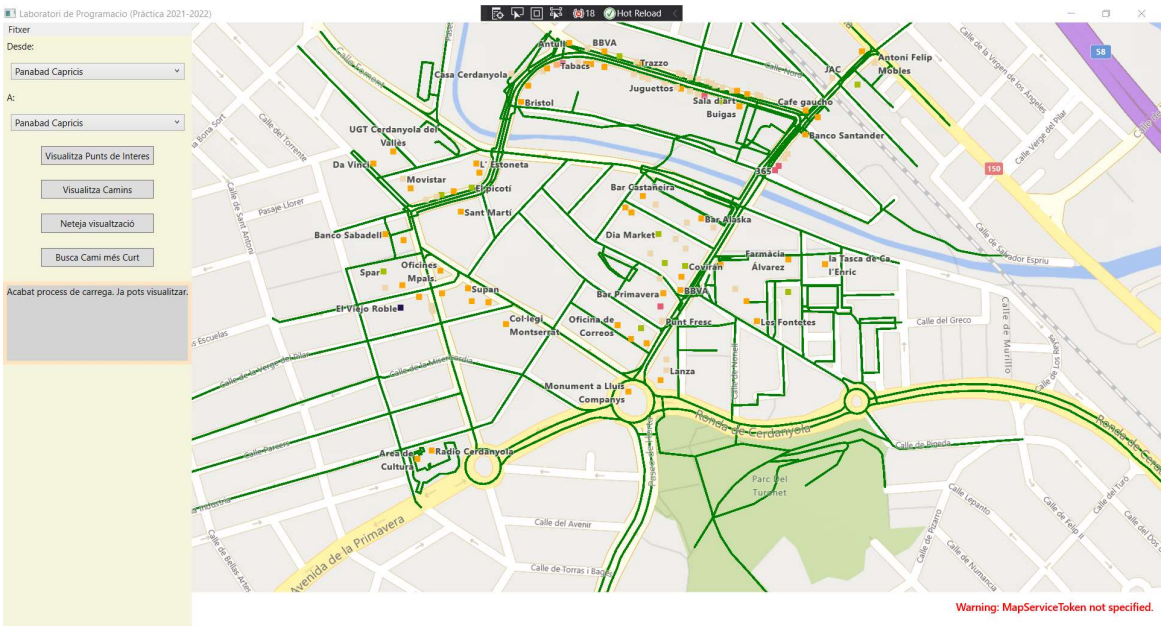
El resultat d'aquesta funció és el que vosaltres rebeu com entrada al parsejaXmlElements del **MapaBase**, que us demanem que implementeu a través de la derivada seva **MapaSolucio**.

Un cop loadOSMData de LP-OSM-2122-WPF ha finalitzat, en el mateix thread carrega els valors de punts d'interès i camins a través dels mètodes `renderPois` i `renderCamins`.

Aquests dos mètodes són els responsables de cridar les vostres implementacions getPdis(), getCamins() i getCamiCoords(), preparant les dades per tal que el projecte de C# les pugui entendre, mitjançant les estructures de dades **PoiBridge** i **WayBridge**.

El fet que **MapRender** segueix un patró de disseny Singleton, les dades que heu estructurat a través del parsejaXmlElements de la classe **MapaBase** no s'han perdut entre aquestes crides de les funcions. D'altra manera, perdríem la informació un cop acabada la localitat de la funció del **construeixOSM**, degut al alliberament de la memòria.

Ara que les dades estan en el projecte de LP-OSM-2122-WPF, aquest utilitza l'API de MapControl per poder representar la informació que li heu donat des de la DLL per pantalla (colors i noms i coordenades pels punts d'interès, conjunt de coordenades pels camins).



La classe XML4OSMUtil, i el conjunt de funcions que conté dins DLL-Bridge, treballen en background interactuant amb la vostra implementació, per parsejar bé el XML (amb una llibreria externa estàndard anomenada Xercesc) i poder representar finalment en el mapa.

ATENCIÓ: Degut a com està estructurat el codi de la pràctica, és important remarcar que la classe XML4OSMUtil i el codi DLL-Bridge, són interns. Per tant els Bridges definits al fitxer capçalera Common.h **no els heu de modificar**. Tampoc hi ha cap avantatge en consultar res del codi intern per assolir els objectius d'aquesta pràctica (encara que ho podeu fer si voleu).