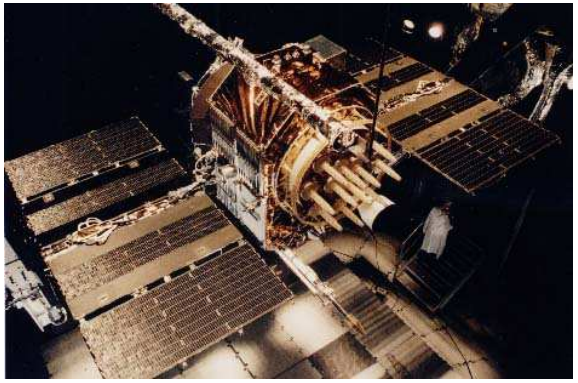
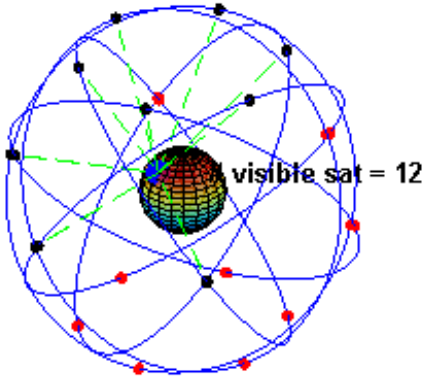
La géolocalisation

La (géo)localisation : le sytème GPS

- GPS = Global Positionning System
- 24 satellites (au moins)



tournant autour de la terre



et diffusant leur position et l'heure de diffusion

- Un récepteur, équipé d'une horloge, reçoit au moins 4 signaux de 4 satellites et calcule alors sa position
- précision de 15 à 100 mètres (version standard)
- source : `http://fr.wikipedia.org/wiki/GPS`

Les réseaux pour la géolocalisation

- Utilise les réseaux téléphoniques cellulaires et le Wi-Fi
- Est plutôt conseillé : utilise moins de puissance électrique, fonctionne à l'intérieur et à l'extérieur, est plus rapide

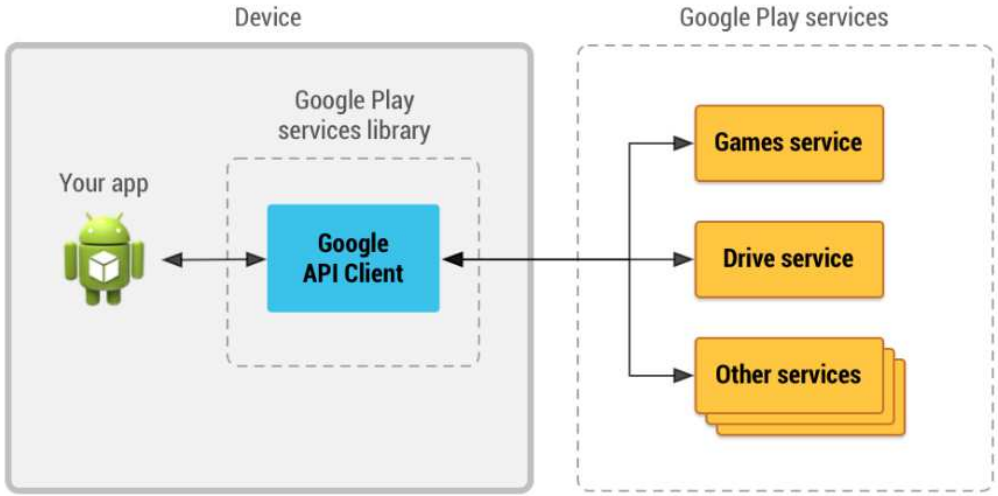
- source : `http://developer.android.com/guide/topics/location/strategies.html`

Les APIs de (géo)localisation

- Il existe 2 bibliothèques pour traiter le positionnement terrestre = la localisation = la géolocalisation. La première est l' "Android framework location APIs", la seconde est la "Google Location Services API" qui est une partie des Google Play Services
- Mais : "The Google Play services location APIs are preferred over the Android framework location APIs (android.location) as a way of adding location awareness to your app. If you are currently using the Android framework location APIs, you are strongly encouraged to switch to the Google Play services location APIs as soon as possible."
- La géolocalisation est très consommatrice de batterie
- Evidemment l'utilisateur doit autoriser la géolocalisation sur son smarphone pour l'utiliser
- Voir à `https://developer.android.com/training/location/index.html`

Google Play services location API : bibliographie

- Le tutorial officiel commence à `https://developer.android.com/training/location/index.html`
- Les exemples sont téléchargeables à `https://github.com/googlesamples/android-play-location`
- Cette bibliothèque Google Play services permet d'avoir des fonctionnalités "diverses et variées" :



Utilisation des Google Play Services

- Avec les Google Play services location APIs, on utilise le "fused location provider " pour récupérer le positionnement terrestre
- Il faut donc d'abord avoir chargé dans l'app le composant Google Play Services :
  - dans l'IDE avec le SDK Manager
  - dans l'environnement de développement et l'app
- Pour cela mettre dans le `build.gradle` du Module:app (pas de Project) :

```
apply plugin: 'com.android.application'
    ...

    dependencies {
        compile 'com.google.android.gms:play-services:10.0.1'
    }
```

- Mettre à jour l'IDE en cliquant sur le bouton "Sync Project with Gradle Files"
- bibliographie : `https://developers.google.com/android/guides/setup`

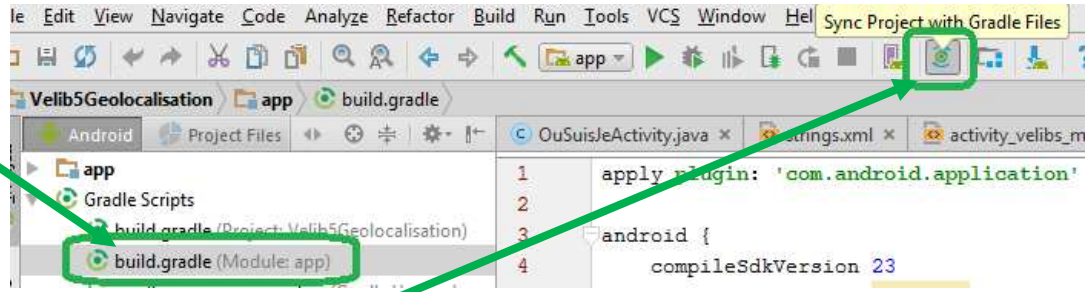
Utilisation des Google Play Services en image

- Plus précisément
- Ecrire :

```
apply plugin: 'com.android.application'
    ...

    dependencies {
        compile 'com.google.android.gms:play-services:10.0.1'
    }
```

dans le `build.gradle`



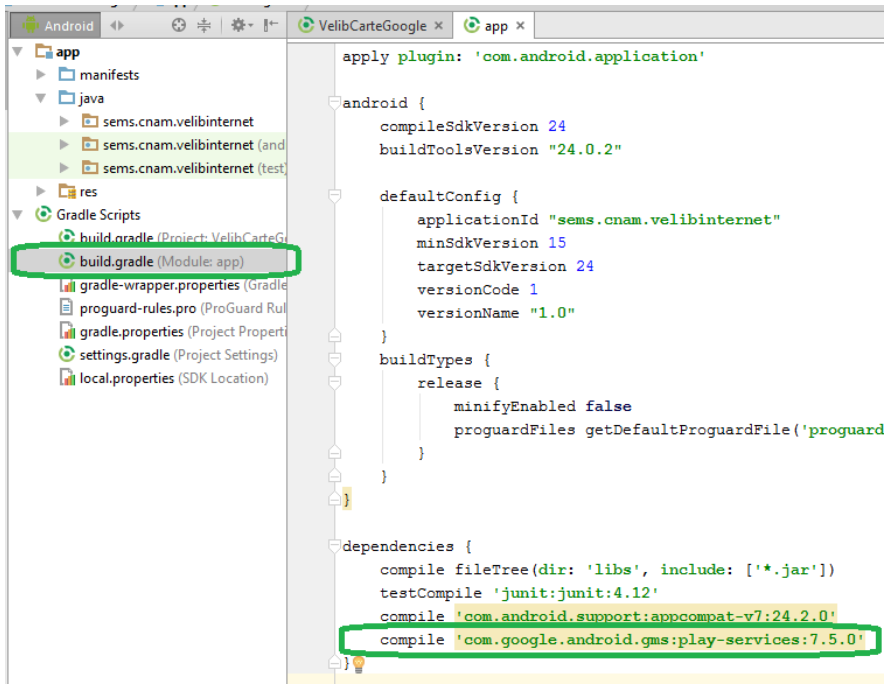
- Puis mettre à jour en cliquant
- bibliographie : `https://developers.google.com/android/guides/setup`

Rappel ? Bonne version de la bibliothèque Google Play services

- Par défaut, Android Studio propose une version 9.8.0. Cette version fonctionne ... mal ! On a des erreurs de multidex
- Il faut mieux utiliser la version des Google Play 7.5.0 (merci Tarik)
- On l'indique dans le fichier build.gradle du module par :

```
compile 'com.google.android.gms:play-services:7.5.0'
```

- C'est à dire



Les permissions

- Mettre dans le fichier manifest, les permissions :

```
<manifest xmlns:android= ... >
     <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
     <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
</manifest>
```

  fille de la balise `manifest`

- Si on n'a besoin que d'une précision "paté de maison d'une ville", la permission `ACCESS_COARSE_LOCATION` suffit

Se connecter aux Google Play Services

- Pour utiliser cette API, il suffit écrire :

```
private GoogleApiClient mGoogleApiClient;
...
if (mGoogleApiClient == null) {
    mGoogleApiClient = new GoogleApiClient.Builder(this)
        .addConnectionCallbacks(this)
        .addOnConnectionFailedListener(this)
        .addApi(LocationServices.API)
        .build();
}
```

en général dans la méthode `onCreate()`

- Cet appel construit (`build()`) un `GoogleApiClient` qui fournit le service de localisation (`LocationServices.API`) en lui indiquant ses listeners de bonnes (`addConnectionCallbacks()`) et défaillantes (`addOnConnectionFailedListener()`) connexions
- Ces listeners sont ici l'activité qui est déclarée :

```
public class OuSuisJeActivity ... implements ConnectionCallbacks,
OnConnectionFailedListener, ... { ... }
```

Se connecter aux Google Play Services

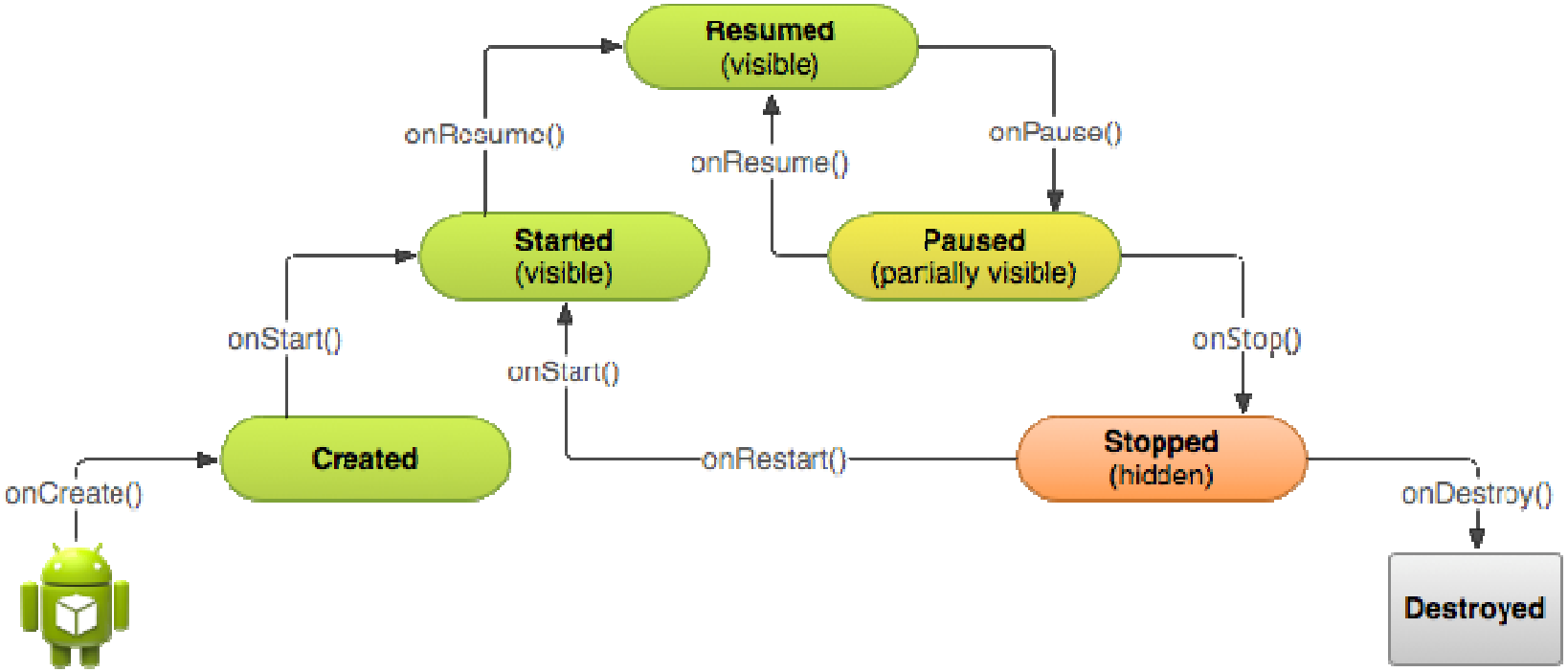
- En fait on écrit plutôt :

```
 ... onCreate(...) { ... buildGoogleApiClient(); ... }

private synchronized void buildGoogleApiClient() {
     mGoogleApiClient = new GoogleApiClient.Builder(this)
          .addConnectionCallbacks(this)
          .addOnConnectionFailedListener(this)
          .addApi(LocationServices.API)
          .build();
     ...
}
```

Cycle de vie d'une `Activity`

- L'automate (simplifié) du cycle de vie : la "step pyramid" :



- source : `http://developer.android.com/training/basics/activity-lifecycle/starting.html`

Gérer le cycle de vie de l'activité

- Il faut lancer le connexion (méthode `connect()`) dans la méthode `start()` de l'activité et l'arréter (méthode `disconnect()`) dans sa méthode `onStop()` (cf. cycle des vie des activités)
- D'où le code :

```
protected void onStart() {
    mGoogleApiClient.connect();
    super.onStart();
}

protected void onStop() {
    mGoogleApiClient.disconnect();
    super.onStop();
}
```

- On peut préciser les listeners de connexion correcte (de classe `GoogleApiClient.ConnectionCallbacks`) et incorrecte (de classe `GoogleApiClient.OnConnectionFailedListener`) comme argument de `GoogleApiClient.Builder()`. Si on met seulement this (cf. diapo précédente), il s'agit de l'activité (le contexte)

L'activité gestionnaire de connexion

- En déclarant l'activité comme `implements ConnectionCallbacks, OnConnectionFailedListener`, elle devient gestionnaire de connexion
- Etant un listener `ConnectionCallbacks`, les méthodes :
  - `onConnected()` est lancée, de manière asynchrone, lorsque la connexion au système de géolocalisation est correcte
  - `onConnectionSuspended()` est lancée lorsque la connexion au système de géolocalisation est temporairement suspendue
- Etant un listener `OnConnectionFailedListener`, la méthode `onConnectionFailed (ConnectionResult result)` est lancée lorsque la connexion au système de géolocalisation est défaillant
- D'où le code de l'activité

L'activité gestionnaire de connexion : son code

- L'activité peut être écrite :

```
public class OuSuisJeActivity ... implements ConnectionCallbacks,
     OnConnectionFailedListener, ... {

     @Override
    public void onConnected(Bundle connectionHint) {
          // recherche des coordonnées géographiques
          // et mise à jour de l'IHM
          // voir diapos suivantes
    }

     @Override
    public void onConnectionSuspended(int cause) {
     // On tente une nouvelle connexion
        mGoogleApiClient.connect();
    }

    @Override
    public void onConnectionFailed(ConnectionResult result) {
        ...
    }
}
```

Obtenir les coordonnées géographiques

- Suite à une connexion, on peut obtenir les coordonnées géographiques connues par la méthode

```
Location LocationServices.FusedLocationApi.getLastLocation(
      GoogleApiClient client)
```

- D'où le code :

```
private Location mCurrentLocation;

@Override
public void onConnected(Bundle connectionHint) {
    mCurrentLocation = LocationServices.FusedLocationApi.getLastLocation(mGoogleApiClient);
    // mettre à jour l'IHM
  }
}
```

- On veut désormais mettre à jour régulièrement les coordonnées géographiques

Le gestionnaire de requête de localisation

- Il faut créer un gestionnaire de requête de localisation qui va indiquer :
  - la fréquence en millisecondes à laquelle l'app aimerait avoir des informations de localisation (méthode `setInterval()`). Cette fréquence est indicative (si pas de réseau ?)
  - la fréquence la plus rapide en millisecondes à laquelle l'app aimerait avoir des informations de localisation (méthode `setFastestInterval()`). L'app ne recevra jamais de mise à jour plus rapidement que cette fréquence. Elle est nécessaire pour éviter des "flashs d'IHM"
  - la priorité qui est en fait la précision des coordonnées terrestres (peut être `PRIORITY_BALANCED_POWER_ACCURACY` , `PRIORITY_HIGH_ACCURACY`, `PRIORITY_LOW_POWER`, `PRIORITY_NO_POWER`)
- On écrit donc :

```
private LocationRequest mLocationRequest;

protected void createLocationRequest() {
     mLocationRequest = new LocationRequest();
     mLocationRequest.setInterval(10000);
     mLocationRequest.setFastestInterval(5000);
     mLocationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
}
```

Lancer les demandes de mise à jour de localisation

- Après :
  - avoir indiquer qu'on veut utiliser la géolocalisation fournie par les Google Play services (`mGoogleApiClient`)
  - s'être connecté sur ce service (lancement de `connect()` sur `mGoogleApiClient`)
  - avoir préparé un gestionnaire de géolocalisation (`mLocationRequest`)
  - avoir, de plus, indiqué le `LocationListener` de connexion

il faut indiquer qu'on veut recevoir régulièrement des informations de géolocalisation à l'aide de :

```
requestLocationUpdates (GoogleApiClient client,
LocationRequest request, LocationListener listener)
```

- Si l'activité est ce `LocationListener` par :

```
public class OuSuisJeActivity ... implements LocationLIstener, ...
```

l'appel peut être :

```
LocationServices.FusedLocationApi.requestLocationUpdates(
            mGoogleApiClient, mLocationRequest, this);
```

Recevoir des mises à jour de localisation

- Etre un `LocationListener` impose d'implémenter la méthode `void onLocationChanged(Location location)`
- La classe utilisée est `android.location.Location`
- La code de cette méthode est donc :

```
private Location mCurrentLocation;

@Override
public void onLocationChanged(Location location) {
     mCurrentLocation = location;
     miseAJourIHM();
}
```

- Remarque : Lorsque l'activité n'a pas le focus, il faut arréter la géolocalisation (économie de batterie). D'où la gestion du cycle de vie de l'activité (cf. diapo suivante)

Retour sur le cycle de vie de l'activité

- Il est bon de concevoir l'activité par :
  - dans l'état Created, on récupère les Google Play Services (`mGoogleApiClient`) et on construit le gestionnaire de localisation (`mLocationRequest`)
  - dans l'état Started, on lance la connexion au GooglePlayServices (`mGoogleApiClient.connect()`) et on indique qu'on demandera à être géolocalisé lorsque l'activité aura le focus (`boolean mDemandeDeGeolocalisation`)
  - dans l'état Resumed, si on est connecté et qu'on a fait une demande de géolocalisation, on lance cette demande (`requestLocationUpdates(...)`)
  - dans l'état Paused, si on est connecté, on arrête cette demande de localisation par la méthode `removeLocationUpdates (GoogleApiClient client, LocationListener listener)`
  - dans l'état Stopped, si on est connecté et qu'on a fait une demande de géolocalisation, on arrête cette demande et on se déconnecte du service de géolocalisation

Code pour le cycle de vie de l'activité (1/2)

- On a donc le code :

```
protected void onStart() {
     super.onStart();
     mGoogleApiClient.connect();
     mDemandeDeGeolocalisation = true;
}

@Override
public void onResume() {
     super.onResume();
     // Dans le code de onPause(), on a arrété la mise à jour de géolocalisation
     // mais on a gardé la communication avec GoogleApiClient
     // On ne relance donc ici que la demande de mise à jour de géolocalisation ici

     if (mGoogleApiClient.isConnected() && mDemandeDeGeolocalisation) {
          LocationServices.FusedLocationApi.requestLocationUpdates(
               mGoogleApiClient, mLocationRequest, this);
     }
}
```

Code pour le cycle de vie de l'activité (2/2)

- suivi de :

```
@Override
protected void onPause() {
     super.onPause();
     // On arrête la mise à jour de géolocalisation mais on reste connecté
     // avec GoogleApiClient
     if (mGoogleApiClient.isConnected()) {
          LocationServices.FusedLocationApi.removeLocationUpdates(mGoogleApiClient, this);
     }
}

@Override
protected void onStop() {
     super.onStop();
     if (mGoogleApiClient.isConnected() && mDemandeDeGeolocalisation) {
          mGoogleApiClient.disconnect();
     }
}
```

Retour sur `onConnected()`

- Lorsque l'activité se connecte au service de géolocalisation, il est bon qu'elle demande les mises à jour de géolocalisation. D'où le code de `onConnected()`

```
private Location mCurrentLocation;

@Override
public void onConnected(Bundle connectionHint) {
    mCurrentLocation = LocationServices.FusedLocationApi.getLastLocation(mGoogleApiClient);
    // mettre à jour l'IHM
  }
    if (mDemandeDeGeolocalisation) {
        LocationServices.FusedLocationApi.requestLocationUpdates(
            mGoogleApiClient, mLocationRequest, this);
    }
}
```

Faire afficher une carte Google dans l'activité

- C'est le cours sur les Google maps. Il faut, entre autre, avoir un "bon" fichier XML d'IHM et l'activité doit avoir :

```
public class OuSuisJeActivity extends FragmentActivity implements ..., OnMapReadyCallback {
    ...
     private GoogleMap mMap;
     private Marker mImageMarqueur;
     private boolean mPremierAffichageDeLaCarteGoogle = true;

     @Override
     public void onCreate(Bundle savedInstanceState) {
          super.onCreate(savedInstanceState);
          setContentView(R.layout.activity_velibs_maps);

          SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager()
                .findFragmentById(R.id.map);
          mapFragment.getMapAsync(this);

          mDemandeDeGeolocalisation = false;
          ...
    }

    public void onMapReady(GoogleMap googleMap) {
        mMap = googleMap;
    }
}
```

La mise à jour de la Google map

- C'est la méthode `miseAJourIHM()` :

```
private void miseAJourIHM() {
     double laLongitude = mCurrentLocation.getLongitude();
     double laLatitude = mCurrentLocation.getLatitude();
     LatLng geoPositionDeLUtilisateur = new LatLng(laLatitude, laLongitude);

     if (mImageMarqueur != null) mImageMarqueur.remove();
     mImageMarqueur = mMap.addMarker(
          new MarkerOptions().position(geoPositionDeLUtilisateur).title("User ICI"));
     if (mPremierAffichageDeLaCarteGoogle) {
          mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(geoPositionDeLUtilisateur, 17));
          mPremierAffichageDeLaCarteGoogle = false;
     } else {
          mMap.moveCamera(CameraUpdateFactory.newLatLng(geoPositionDeLUtilisateur));
     }
}
```

Exercice

- Faire afficher sur une carte la géolocalisation d'un utilisateur

Fin