TOUCH
MULTITOUCH
GESTURE

Touch

- « CLICK » (1 touch, sans plus d'information:
  - Attribut « onclick » dans le layout (avec un nom de méthode de l'activité courrante)
  - Interface OnClickListener
    - Méthode void onClick (View v)
    - setOnClickListener sur la View
- TOUCH
  - Surcharge (pour une View) de public boolean onTouchEvent(MotionEvent e)
  - Interface View.OnTouchListener
    - Méthode boolean onTouch(View v, MotionEvent event)
      - » Retourne true si l'événement a été prise en compte
    - setOnTouchListener sur la View

MultiTouch (1/2)

- Information dans le MotionEvent (e)
- e.getActionMasked() / e.getActionIndex ()
  - Les actions :
    - ACTION_CANCEL — The current gesture has been aborted.
    - ACTION_DOWN — A pressed gesture has started, the motion contains the initial starting location.
    - ACTION_HOVER_ENTER — The pointer is not down but has entered the boundaries of a window or view.
    - ACTION_HOVER_EXIT — The pointer is not down but has exited the boundaries of a window or view.
    - ACTION_HOVER_MOVE — A change happened but the pointer is not down (unlike ACTION_MOVE).
    - ACTION_MOVE — A change has happened during a press gesture (between ACTION_DOWN and ACTION_UP).
    - ACTION_OUTSIDE — A movement has happened outside of the normal bounds of the UI element.
    - ACTION_POINTER_DOWN — A non-primary pointer has gone down.
    - ACTION_POINTER_UP — A non-primary pointer has gone up.
    - ACTION_SCROLL — The motion event contains relative vertical and/or horizontal scroll offsets.
    - ACTION_UP intermediate — A pressed gesture has finished, the motion contains the final release location as well as any
  - e.getActionIndex( ) permet de savoir quel “pointeur” en est la source

MultiTouch (2/2)

- e.getPointerCount() : nombre de pointeurs = nombre de points de contact reconnus
- e.getPointerId(int index) : un nombre (0,1,2,…) associé à un pointeur
  - Dans l'ordre d'apparition
  - Ré-indexation en cas de disparition…
  - e.getPointerId(index) permet d'avoir l'id du pointer qui lui ne changera pas jusqu'à sa disparition
- findPointerIndex (int pointerId) : permet de retrouver l'index à partir d'un id
- e.getPointerCoords(int index)
- Et les autres méthodes avec un paramètre pointerIndex ou un pointerId : http://developer.android.com/reference/android/view/MotionEvent.html

Geste

- 2 niveaux de reconnaissance de geste
  - http://developer.android.com/reference/android/view/GestureDetector.html
  - http://developer.android.com/reference/android/view/ScaleGestureDetector.html
- Il faut appeler le (ou les) détector dans onTouch : detecor.onTouchEvent(motionEvent);
  - Attention à la fusion des setOnTouchListener
- Le détector appelle en callback une méthode en correspondance avec le geste détecté

GestureDetector

- Pour les « long touch », « scroll » et « swipe »
- Objet utilisé (délégation)
  - Appel depuis onTouch

```
public boolean onTouch(View v, MotionEvent e) {
      detector.onTouchEvent(e);
      return true;  // l'événement a été consommé
   }
```

- Appel en callback un GestureDetector.OnGestureListener

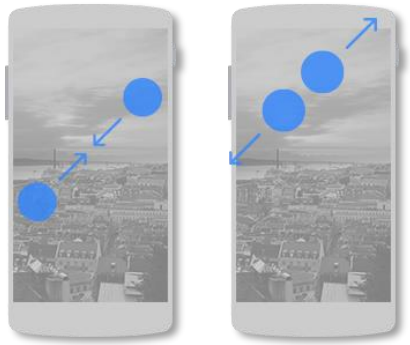
```
new GestureDetector ( Context context,
            GestureDetector.OnGestureListener listener)
```

GestureDetector.OnGestureListener

- boolean onDown (MotionEvent e)
  - Précéde tous les autres events
  - Doit être consommé (return true) pour que onFling ou onScroll soit appelé
- boolean onFling (MotionEvent initial, MotionEvent current, float velocityX, float velocityY)
  - Détection d'un mouvement ≈ touch « dragged » après avoir relevé le doigt
  - Tests sur velocityX (seuil) et
    Test sur current.getX() – initial.getX( ) (seuil et signe)
    Pour savoir s'il y a un swipe horizontal
  - Idem pour Y
- onLongPress (MotionEvent e)
- void onShowPress (MotionEvent e)
  - Après un onDown mais pas bougé depuis. Pour retour d'information
- boolean onSingleTapUp (MotionEvent e)
  - Le doigt se relève… et ne touche plus l'écran
- boolean onScroll (MotionEvent initial, MotionEvent current, float distanceX, float distanceY)
  - Pour les scrolls, comme fling, mais le doigt touche encore l'écran

- Dans la suite « normale » :
  - On appuie sur l'écran avec le doigt : onTouch / onDown
  - On déplace le doigt : suite de onTouch / onScroll
  - On relève le doigt : onTouch / onFling

ScaleDetector



- Pour les zooms avec un pitch
  - C.f. image de http://developer.android.com/design/patterns/gestures.html
- Objet utilisé en délégation
- Appel en callback un ScaleGestureDetector.OnScaleGestureListener

```
new ScaleGestureDetector ( Context context,
ScaleGestureDetector.OnScaleGestureListener
 listener)
```

ScaleGestureDetector.OnScaleGestureListener

- 3 méthodes
  - boolean onScale(ScaleGestureDetector detector)
  - Boolean onScaleBegin(ScaleGestureDetector detector)
  - onScaleEnd(ScaleGestureDetector detector)
- On obtient les informations par rappel au détecteur
  - Par exemple : detector. getScaleFactor() pour avoir le facteur d'échelle

Combiner plusieurs détecteurs

- 1 seul onTouch qui appel tous les détecteurs
  - Directement
  - Ou indirectement en appelant des onTouch pour un fonctionnement isolé

Commande vocale,

Text to Speech,

Lecture de son

AUDIO / VOCALS

Reconnaissance Vocale (service Google)

- Google propose une « activité » pour la reconnaissance vocale.
  - RecognizerIntent.ACTION_RECOGNIZE_SPEECH
  - http://developer.android.com/reference/android/speech/RecognizerIntent.html
  - Pas d'autorisation particulière
- On lance donc cette activité depuis l'activité de l'application, dans l'attente d'un résultat.
  - Pour le paramétrage, on utilise un Intent
- On reçoit le résultat dans un autre Intent

Vérifier la disponibilité de l'activité « recognition »

```
PackageManager pm = getPackageManager();
List<ResolveInfo> activities =
pm.queryIntentActivities( new Intent(
RecognizerIntent.ACTION_RECOGNIZE_SPEECH ),
0);

if (activities.size() != 0) {
     // disponible…
} else {
     // non disponible…
}
```

Lancer la reconnaissance vocale

```
// on est dans une activité
// lancement de l'intent avec le nom de la classe... (constante)
Intent intent = new Intent(
      RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,
            RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);

// titre de la boite de dialogue
intent.putExtra(RecognizerIntent.EXTRA_PROMPT, "Fisheye - Speech recognition");

// VOICE_CODE pour retrouver l'appel pour le résultat
// nombre choisi par le développeur
startActivityForResult(intent, VOICE_CODE);
```

Réception du résultat

- **Pour test unitaire : isoler dans une méthode le traitement du retours dans une méthode**

```
protected void onActivityResult(int requestcode, int resultcode,
Intent data) {
        // test si le retour vient de la reconnaissance vocale
        // et si l'activité à bien fonctionnée
  if (requestcode == VOICE_CODE && resultcode == RESULT_OK) {
        ArrayList<String> matches =
  data.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS);
        processVoiceInput(matches);
        }
  else {
        // autres retours d'activité…
  }
}
```

Traitement du résultat

```
protected void processVoiceInput(ArrayList<String> matches) {
if (nb_rep > 0)
        {
        // matches contient les réponses potentielles
        // il faut regarder dans ces réponses s'il y a
        // une valeur attendue…
        // ici sur les 3 1ère réponses
        int nb_test = Math.min(3, nb_rep);

        for(int i = 0; i < nb_test; i++) {
                if (matches.get(i). equalsIgnoreCase("commande")) {
                        // faire l'action associée à la commande
                        break;
                        }
                // etc.
                }
        }
}
```

Text To Speech

- Lecture de texte
- Mode asynchrone
  - lecture sur un autre thread que le thread principal
- Téléchargement de voix lors de la 1ère utilisation

Initialisation de TTS

```
// this est l'activité en cours, c'est surtout un Context
tts = new TextToSpeech(this, new TextToSpeech.OnInitListener() {
    @Override
    public void onInit(int status) {
        if (status == TextToSpeech.SUCCESS) {
            // traitement en cas de succes
            // par exemple activation d'un bouton lecture
            play.setEnabled(true) ;
            play.setOnClickListener(NomDeLaClasseActivity.this);

            // NomDeLaClasseActivity.this : référence à l'activité,
            //car ici this = le onInitListener

            // déterminer la langue
            tts.setLanguage(Locale.FRANCE);

            // pour savoir quand un texte est fini d'être lu
            tts.setOnUtteranceProgressListener(ttsListener);
            }
        }
});
```

Lire un texte (vers sdk < 21)

```
// version sdk < 21
HashMap<String, String> params = new HashMap<String,
String>();
// pour recevoir les événements de fin de lecture
params.put(TextToSpeech.Engine.KEY_PARAM_UTTERANCE_ID,
"id"+System.currentTimeMillis());
// écrase une éventuelle lecture en cours
// possibilité d'ajouter : TextToSpeech.QUEUE_ADD
tts.speak(txt,  TextToSpeech.QUEUE_FLUSH, params);
```

- Version sdk < 21 : speak(String text, int queueMode, HashMap<String, String> params)
- Version sdk 21 : speak(CharSequence text, int queueMode, Bundle params, String utteranceId)

Listener sur le TTS

```
// class abstraite…
UtteranceProgressListener ttsListener = new UtteranceProgressListener() {
        @Override
        public void onStart(String utteranceId) {
          // ...
        }

        @Override
        public void onDone(String utteranceId) {
          // ...
         }

        @Override
        public void onError(String utteranceId) {
          // ...
        }
    };

// exécuté potentiellement sur un thread différent du thread principal
// donc => lactivité.runOnUiThread( ... ) pour des mises à jours graphiques
```

MediaPlayer

- http://developer.android.com/guide/topics/media/mediaplayer.html
- http://developer.android.com/reference/android/media/MediaPlayer.html

- Version évoluée : https://developer.android.com/guide/topics/media/exoplayer.html

Les capteurs disponibles

Le gps

SENSORS

Accelerometers

- Interface SensorEventListener
- Connecter
  - À lancer dans onResume ou onStart

```
SensorManager sm = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
Sensor mAccelero = sm.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
sm.registerListener(the_listener, mAccelero, SensorManager.SENSOR_DELAY_UI);
```

- Plusieurs capteurs
  - http://developer.android.com/reference/android/hardware/Sensor.html
  - Lumière, pression, … et `Sensor.TYPE_ACCELEROMETER`
  - Présence non garantie (dépend du dispositif)
- Déconnecter

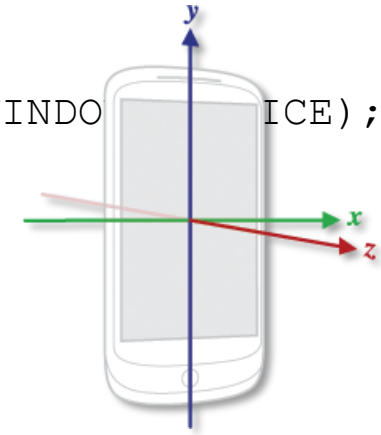
```
mSensorManager.unregisterListener(the_listener);
```

  - si on sort de l'application / onStop - onPause
  - À reconnecter si on revient / onStart - onResume

onSensorChanged (1/2)

- Listener à séparer de l'application (test)
- public void onSensorChanged(SensorEvent se)
- Tenir compte de l'orientation

```
WindowManager wm = (WindowManager) getSystemService(Context.WINDO      ICE);
Display display = wm.getDefaultDisplay();
display.getRotation();
```



- On obtient des accélérations…
  - selon les 3 axes avec la gravité incluse
  - Il faut un gyroscope (pour `Sensor.`*`TYPE_MAGNETIC_FIELD`*`)` pour « extraire » la gravité
    - Absent de la tablette…

onSensorChanged (2/2)

```
@Override
public void onSensorChanged(SensorEvent sensorEvt) {
    if (sensorEvt.sensor.getType() == Sensor.TYPE_ACCELEROMETER){
        // traitement des valeurs
        // dans le tableau sensorEvent.values
        // qui contient dans l'ordre
        // les accélérations selon x, y, z
    }
}
```

Accéléromètre et Gyroscope (1/3)

Etape 1 : déclaration

```
// Il faudra implémenter l'interface :
// SensorEventListener

// variable nécessaire pour s'abonner
    private SensorManager mSensorManager;
    private Sensor mAccelerometer;
```

Etape 2 : retrouver le capteur fourni par le système

```
// ce code est extrait d'une Activity
// Là c'est pour savoir les changements d'orientation du téléphone
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
// pour obtenir la « gravité »
mAccelerometer=mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
// pour obtenir le champ magnétique
mMagnetic = mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);
```

Accéléromètre et Gyroscope (2/3)

Etape 3 : s'abonner par exemple quand l'application est réactivée

```
protected void onResume() {
        super.onResume();
        // on s'abonne en précisant le listener, le capteur
        // et le type de fréquence de réception des événements
        mSensorManager.registerListener(this, mAccelerometer, SensorManager.SENSOR_DELAY_UI);
        mSensorManager.registerListener(this, mMagnetic, SensorManager.SENSOR_DELAY_FASTEST);
}
```

Etape 3 bis : on doit se désabonner quand on quitte l'application

```
// ce code est extrait d'une Activity
protected void onPause() {
        super.onPause();
        mSensorManager.unregisterListener(this);
}

@Override
protected void onStop() {
        super.onStop();
        mSensorManager.unregisterListener(this);

}
```

Accéléromètre et Gyroscope (3/3)

Etape 4 : on implémente `SensorEventListener`

```
public void onAccuracyChanged(Sensor sensor, int accuracy) { }

float [] gravity = new float[3];
float [] geomagetic = new float[3];

public void onSensorChanged(SensorEvent event) {
                // pour obtenir l'orientation, il faut avoir des matrices de rotation et d'inclinaison
                // pour les avoir, il faut la gravité et le champ magnétique
                if (event.sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD) {
                                for(int i=0; i<3; i++){  geomagetic[i] =  event.values[i];   }
                                }

                if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
                                for(int i=0; i<3; i++){  gravity[i] =  event.values[i];   }
                                }


                if ((gravity[2] == 0) || (geomagetic[0] == 0)) { // on n'a pas les infos pour calculer les matrices
                                return;
                                }

                // calcul des matrices
                float[] RotationMatrix = new float[9];
                float[] I = new float[9];
                SensorManager.getRotationMatrix(RotationMatrix, I, gravity, geomagetic);

                // calcul de l'orientatoin
                float [] values = new float[3];
                SensorManager.getOrientation(RotationMatrix, values);

                // il reste à faire le traitement de l'orientation
                }
```

GPS

- Description complète à http://developer.android.com/guide/topics/location/strategies.html
- Principe d'abonnement / callback
  - Pour « essayer » ou tester : « Mock » / via adb / avec une application qui simule le gps (fake gps) / émulateur (?)

Pour utiliser le gps

- Permission :

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

- Méthode getSystemService de contexte

```
LocationManager locationManager = (LocationManager)
uneActivite.getSystemService(Context.LOCATION_SERVICE);
```

- Abonnement aux événements

```
locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, 0,
0, aLocationListener);
```

- Interface logicielle LocationListener

```
public void onLocationChanged(Location location)
public void onStatusChanged(String provider, int status, Bundle extras)
public void onProviderEnabled(String provider)
public void onProviderDisabled(String provider)
```

Location

- http://developer.android.com/reference/android/location/Location.html
- Contient latitude, longitude, etc.
- Notez la méthode distanceTo…