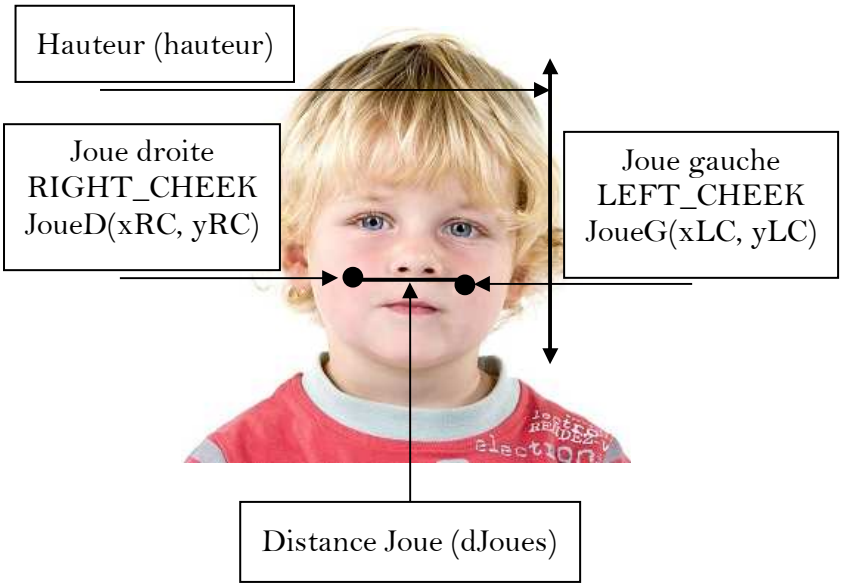
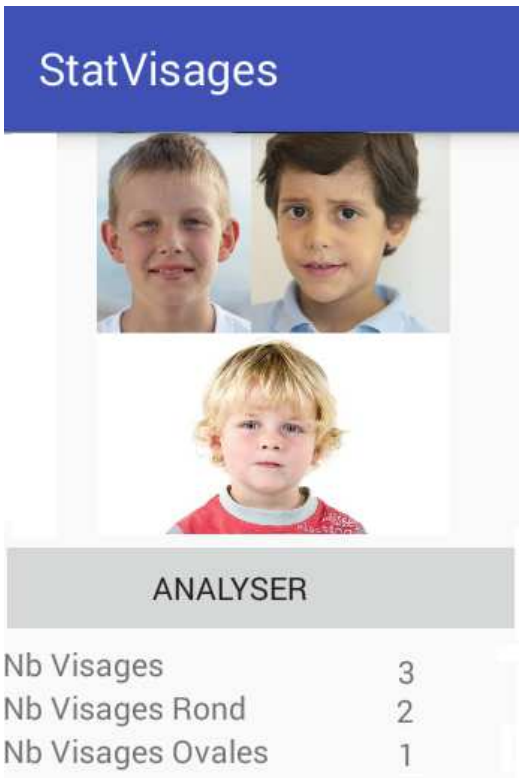
ISET SFAX AU 2021/2022 SEMESTRE I		
DEVOIR FINAL		
Classe : SEM31	*Matière : **D**éveloppement **M**obile **A**vancé*	*Nb pages : 8*
*Enseignant : **S**ouissi **H**afedh*		
Documents Non Autorisés	*Barème : 20 = 5 + 8 + 7*	*Durée : 1 heure 30 minutes*

NB. Ne pas gérer les exceptions (try catch).

Exercice1 (Utiliser Annexe I)

"**StatVisages**" est une application Android qui permet de détecter les visages d'une image et d'afficher un ensemble de statistiques sur les visages détectés. "**StatVisages**" contient une seule activité.





Pour obtenir la distance " dJoues " entre les deux points JoueG(xLC,yLC) et JoueD(xRC,yRC), utiliser la formule « **dJoues=Math.sqrt(Math.pow(xLC - xRC, 2)+Math.pow(yLC - yRC, 2))** ».

Le coefficient **cJH = dJoues/hauteur** permet de savoir si le visage est rond ou ovale.

Si **cJH >= 0,3** alors le visage est rond sinon il est ovale.

La méthode "analyser()" permet de détecter les visages de l'image, de calculer et d'afficher :

- le nombre de visages,
- le nombre de visages ronds,
- le nombre de visage ovales.

1- Donner le code de la méthode "**analyser()**" de la classe "**MainActivity**".

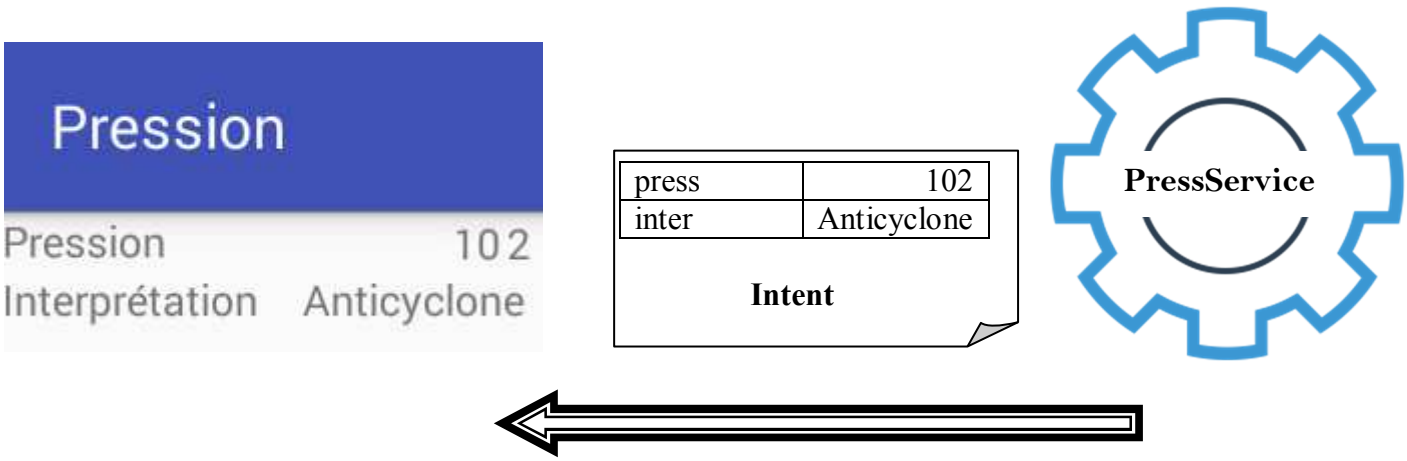
Exercice2 (Utiliser Annexe II)

"**Pression**" est une application Android qui permet d'afficher la pression atmosphérique et son interprétation. Si la pression est inférieure à 101 alors l'interprétation est "**Dépression**" sinon elle est "**Anticyclone**".

"**Pression**" est composée d'une activité "**MainActivity**" et d'un service "**PressService**". "PressService" utilise un capteur de type TYPE_PRESSURE pour intercepter les modifications de la pression atmosphérique "press", par la suite il calcule son interprétation "inter" et diffuse (broadcast) "press", et "inter" dans un Intent.

Lorsque "MainActivity" fait un "Resume" elle démarre le service et lorsqu'elle fait une "Pause" elle arrête le service. "MainActivity" utilise un BroadcastReceiver pour lire les deux valeurs envoyées par le service et les afficher dans les TextViews.

Le fonctionnement de l'application est comme suit :



1- Donner le code des méthodes "ajouterEcouteur()", "demarrerService() ", "arreterService()", "onResume()", "onPause()" et "actualiser(…)" de la classe "**MainActivity**".

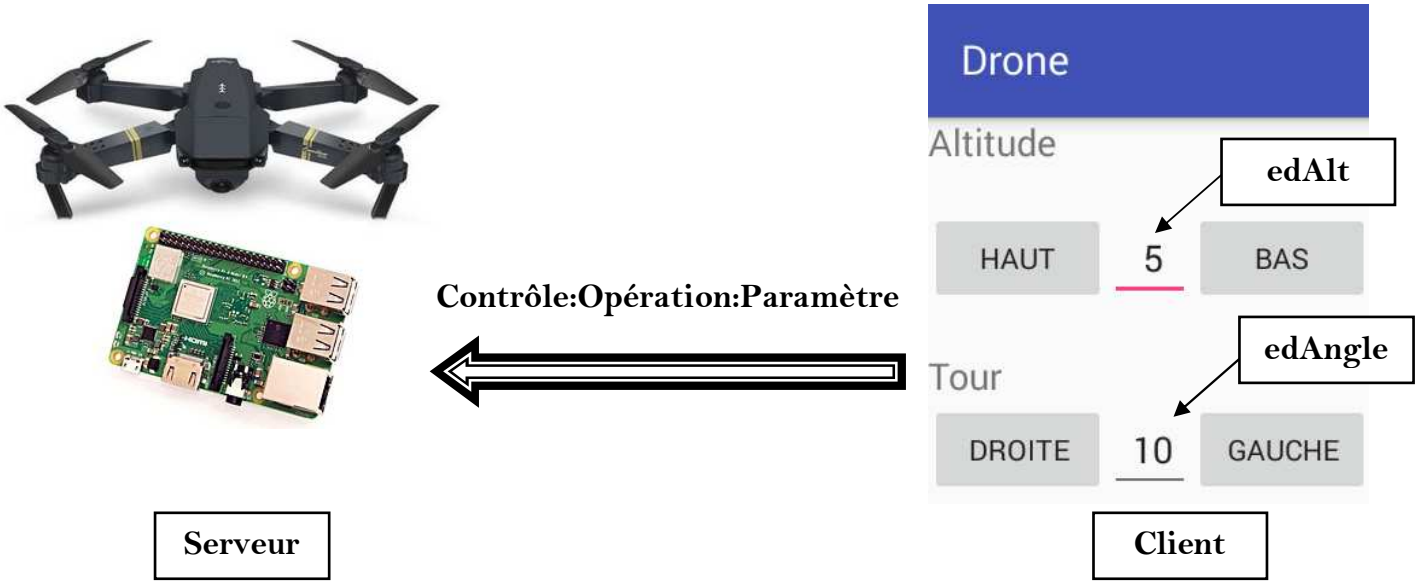
2- Donner le code des méthodes "init()", "onDestroy()" et de "onSensorChanged()" de la classe "**PressService**".

Exercice3 (Utiliser Annexe III)

On désire développer une application qui permet de contrôler un drone relié à une carte RaspBerry PI. Cette application utilise les sockets. Le programme serveur est installé sur la carte RaspBerry PI d'adresse "192.86.10.19" reliée au drone pour le commander. Le programme client est installé sur un Smartphone android pour envoyer les commandes de l'utilisateur au serveur. Les contrôles qu'on peut effectuer sur ce drone et les messages envoyés par le socket clients sont résumés dans le tableau suivant :

Contrôle	Opération	Paramètre	Message envoyé
Altitude (0)	Haut (0)	alt	0:0:alt
	Bas (1)	alt	0:1:alt
Tour (1)	Droite(0)	angle	1:0:angle
	Gauche (1)	angle	1:1:angle

Le fonctionnement de l'application est comme suit :



1- Donner le code des méthodes " lancerThreadServeur()", "demarrerServeur()" et "executerCommande(…)" de la classe " **ServDrone**" (coté serveur).

2- Donner le code des méthodes " lancerThreadClient()", "demarrerClient()", "envoyer() ", " haut() ", " bas() ", " droite() " et "gauche()" de la classe "**MainActivity**" (coté client).

Annexe I

Bitmap	//Pour transformer l'image d'un ImageView en bitmap Bitmap bitmap = ((BitmapDrawable)imgView.getDrawable()).**getBitmap**();
FaceDetector.Builder	FaceDetector.**Builder**(Context context); FaceDetector.Builder **setTrackingEnabled** (boolean trackingEnabled) FaceDetector.Builder **setLandmarkType** (int landmarkType) FaceDetector.Builder **setClassificationType** (int classificationType) FaceDetector **build**();
Frame	Frame frame = new Frame.Builder().setBitmap(bitmap).**build**();
FaceDetector	SparseArray<Barcode> **detect**(Frame frame); **ALL_LANDMARKS** **ALL_CLASSIFICATIONS**
SparseArray<Face>	int **size**() Face **valueAt**(int index);
Face	BOTTOM_MOUTH LEFT_CHEEK RIGHT_ CHEEK … float **getHeight()** List<Landmark> **getLandmarks**() (for (Landmark l : face.getLandmarks()) {})
Landmark	PointF **getPosition**() Retourne les coordonnées (x, y) du landmark avec (0, 0) est le point supérieur gauche de l'image. int **getType**() Retourne le type du landmark (BOTTOM_MOUTH, LEFT_CHEEK, RIGHT_ CHEEK, etc.).
PointF	float **x** float **y**

```
package stat.com.statvisages;
//imports
public class MainActivity extends AppCompatActivity {
    private ImageView imgVisages;
    private Button btnAnalyser;
    private TextView tvNbV;       //TextView nombre de visage
    private TextView tvNbVR;      //TextView nombre de visage ronds
    private TextView tvNbVO;      //TextView nombre de visage ovales
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        //...
        init();
    }
    private void init() {
        // Les findViewById
        ajouterEcouteur();
    }
    private void ajouterEcouteur() {
        //btnAnalyser.setOnClickListener(...) -> analyser();
    }

    private void analyser() {
    }

}
```

Annexe II

Activity
startService(Intent intent) stopService(Intent intent) registerReceiver(BroadcastReceiver br, new IntentFilter(String filter)) unregisterReceiver(BroadcastReceiver br)
Intent
Intent() Intent(Conext context, Class classe) putExtra(String nom, double valeur) double getDoubleExtra(String nom, double defaultValue) setAction(String action)
BroadcastReceiver

BroadcastReceiver br= new BroadcastReceiver() {
@Override
public void onReceive(Context context, Intent intent) { }
};

Service
sendBroadcast(Intent intent)
SensorEventListener
@Override public void onAccuracyChanged(Sensor sensor, int accuracy) { } @Override public void onSensorChanged(SensorEvent event) {}
SensorManager
SensorManager smg = (SensorManager) getSystemService(SENSOR_SERVICE); SENSOR_DELAY_UI
Sensor
Sensor sensor = smg.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
SensorEvent
sensor.getType() event.values[int indice]

```
package com.press;
//imports
public class MainActivity extends AppCompatActivity {
    private TextView tvPress;
    private TextView tvInter;
    private BroadcastReceiver br;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        init();
    }
    private void init() {
        // Les findViewById
        ajouterEcouteur();
    }
    private void ajouterEcouteur()  {    }
    private void demarrerService()  {    }
    private void arreterService()   {    }
    @Override
    protected void onResume()   {      }
    @Override
    protected void onPause()    {      }
    protected void actualiser(Intent intent)    {          }

}
package com.press;
//imports
public class PressService extends Service implements SensorEventListener {
    public static final String ACTION_PRESS="PRESS";
    private SensorManager smg;
    private Sensor press;
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
    @Override
    public void onCreate() {
        super.onCreate();
        init();
    }
    private void init() {   }
    @Override
    public void onDestroy() {    }
    @Override
    public void onSensorChanged(SensorEvent event) {    }

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

Annexe III

ServerSocket
ServerSocket(int port) Socket accept()
Socket
Socket(Inet4Address i, int port) InputStream getInputStream() OutputStream getOutputStream()
Inet4Address
Inet4Address i = (Inet4Address) Inet4Address.getByName(ADR_SERVEUR);
Runnable

Runnable r = new Runnable() {
@Override
public void run() { }
};

Thread
Thread(Runnable r) start()
BufferedReader
IntputStream inp= … ; BufferedReader br = new BufferedReader(new InputStreamReader(inp)); String cmd = br.readLine();
PrintWriter
OutputStream out= … ; PrintWriter pw = new PrintWriter(new BufferedWriter(new OutputStreamWriter (out)), true); String cmd= "Message" ; pw.println(cmd);
String
String[] split(String sep)

```
package com.drone;
public class ServDrone extends JFrame {
  private static final long serialVersionUID = 1L;
  private static final int PORT = 6095;
  private ServerSocket ss;
  private Socket s;
  private BufferedReader br;
  public static void main(String[] args) {
    ServDrone f = new ServDrone ();
    f.setVisible(true);
  }
  public ServDrone () {
    setSize(300, 200);
    setLocation(500, 200);
    setTitle("Serveur Drone");
    lancerThreadServeur();
  }
```

```
  protected void haut   (int alt)    { // monter vers le haut de alt}
  protected void bas    (int alt)    { // descendre vers le bas de alt }
  protected void droite (int angle)  { // tourner à droite de angle}
  protected void gauche (int angle)  { // tourner à gauche de angle}
```

```
  private void lancerThreadServeur() {
  }
  private void demarrerServeur() {
  }
  private void executerCommande(String commande) {
  }
}
```

```
package com.drone;
//imports
public class MainActivity extends Activity {
  private static final String ADR_SERVEUR = "192.86.10.19";
  private static final int PORT = 6095;
  private Button btnHaut;
  private Button btnBas;
  private Button btnDroite;
  private Button btnGauche;
  private EditText edAlt;
  private EditText edAngle;
  private Socket s;
  private PrintWriter pw;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    init();
  }
  private void init() {
    // Les findViewById
    ajouterEcouteur();
    lancerClient();
  }
  private void ajouterEcouteur() {
    // btnHaut.setOnClickListener(...)   -> haut();
    // btnBas.setOnClickListener(...)    -> bas();
    // btnDroite.setOnClickListener(...) -> droite();
    // btnGauche.setOnClickListener(...) -> gauche();
  }
```

```
  private void lancerThreadClient()          {     }
  private void demarrerClient()              {     }
  private void envoyer(final String cmd)     {     }
  private void haut()                        {     }
  protected void bas()                       {     }
  protected void droite()                    {     }
  protected void gauche()                    {     }
```

```
}
```