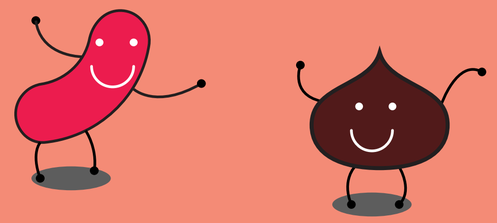
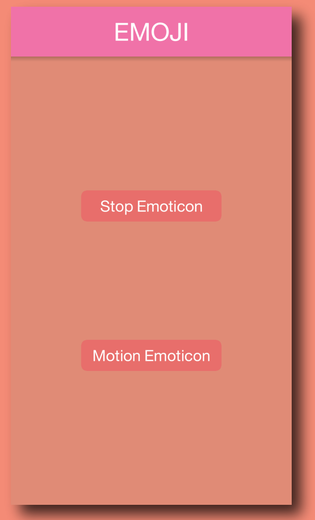
EMOJI

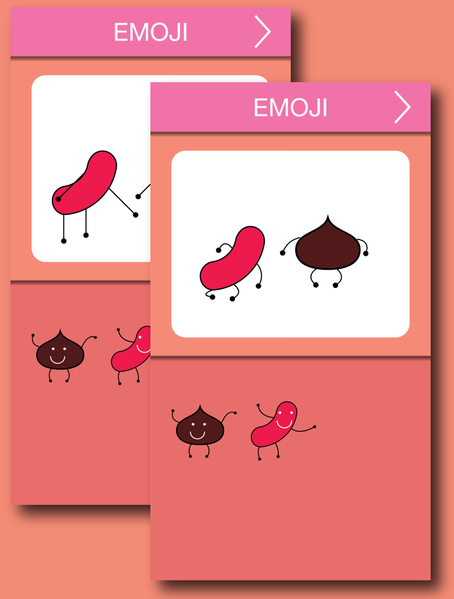
ID430 Software Prototyping

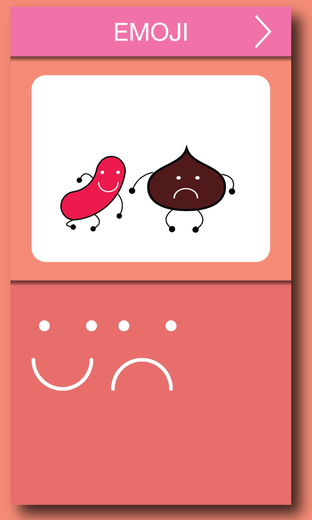
20120371 Hyeongjong Kim

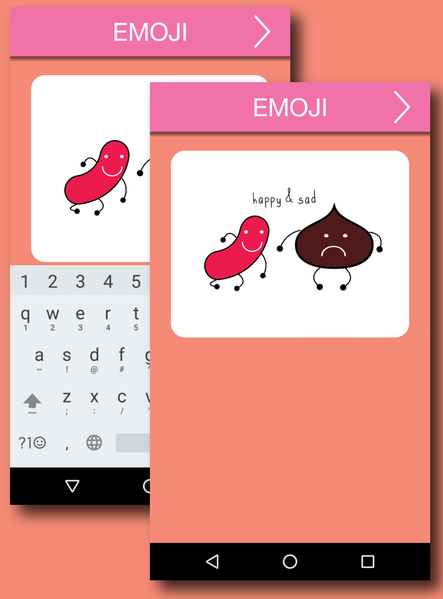


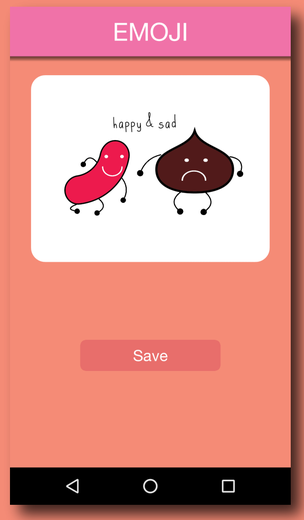
What is EMOJI? Making emoticon











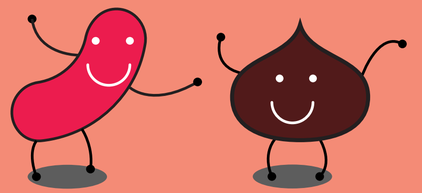
First page
User can select emoticon making mode

Posture Page
User can select character and add it to the display. And can modify arms and legs of emoticons and scale or move emoticon.

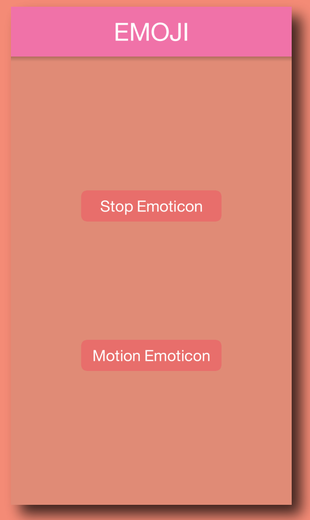
Expression Page
User can select expression of emoticon. (User still can scale or move emoticon.)

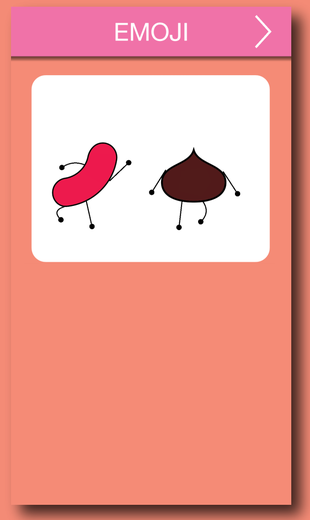
Text Page
User can type text of emoticon. Also user can move text. (User still can scale or move emoticon.)

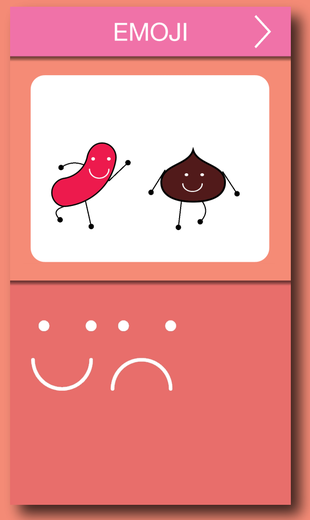
Save Page
User can save image of emoticon.

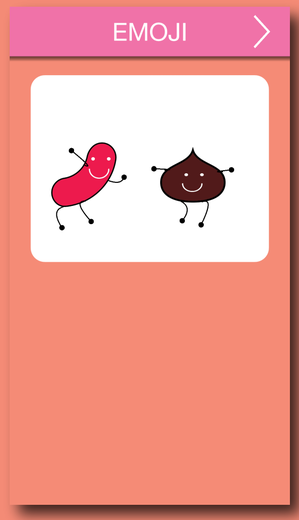


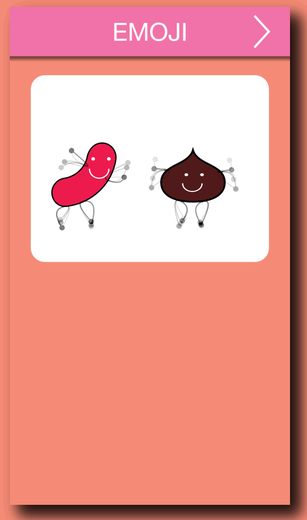
What is EMOJI? Making moving emoticon

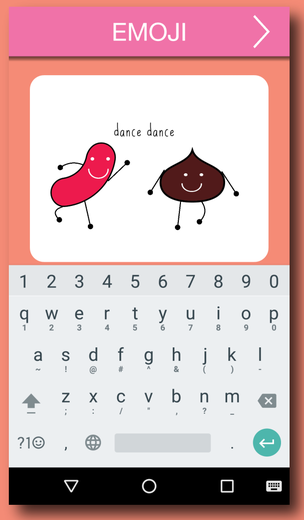


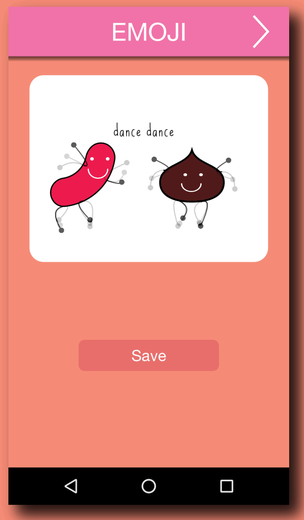












First page
User can select emoticon making mode

Posture Page
User can select character and add it to the display. And can modify arms and legs of emoticons and scale or move emoticon.

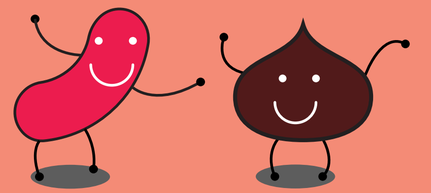
Expression Page
User can select expression of emoticon. (User still can scale or move emoticon.)

2nd Posture Page
User can select character and add it to the display. And can modify arms and legs of emoticons and scale or move emoticon.

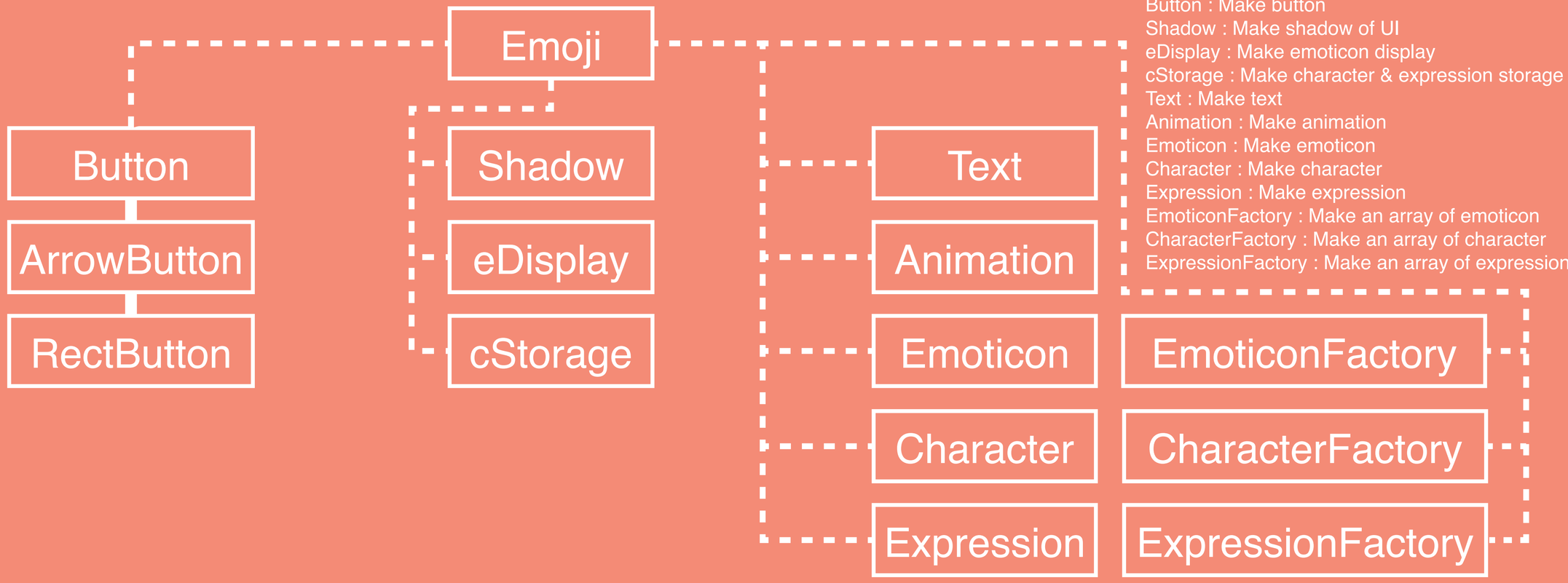
Cheking Page
User can check an animation. (Animation is made from first posture to second posture)

Text Page
User can type text of emoticon. Also user can move text. (User still can scale or move emoticon.)

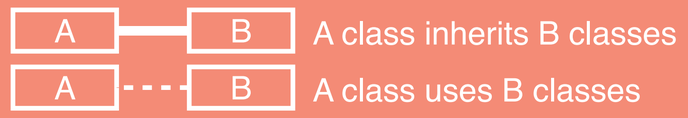
Save Page
User can save image of emoticon.

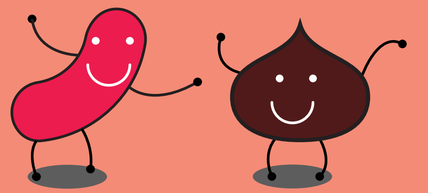


Structure of EMOJI?



Emoji : Manage whole emoji system
Button : Make button
Shadow : Make shadow of UI
eDisplay : Make emoticon display
cStorage : Make character & expression storage
Text : Make text
Animation : Make animation
Emoticon : Make emoticon
Character : Make character
Expression : Make expression
EmoticonFactory : Make an array of emoticon
CharacterFactory : Make an array of character
ExpressionFactory : Make an array of expression



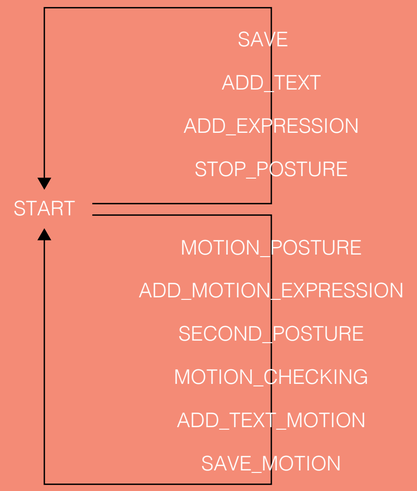


Emoji Class

```
class Emoji {
  protected Mode mode;
  RectButton stopButton;
  RectButton motionButton;
  RectButton saveButton;
  ArrowButton arrowButton;
  eDisplay eDisplay;
  cStorage cStorage;
  Shadow shadowB;
  Shadow shadowCS;
  CharacterFactory characterFactory;
  ArrayList<Character> allCharacters;
  ExpressionFactory expressionFactory;
  ArrayList<Expression> allExpressions;
  EmoticonFactory emoticonFactory;
  ArrayList<Emoticon> allEmoticons;
  ArrayList<Emoticon> emoticonsDraw;
  Text emoticonText;
  PImage mask;
  PGraphics pg;
  ArrayList<Emoticon> startEmoticonsDraw;
  ArrayList<Emoticon> endEmoticonsDraw;
  Animation ani;
```

- for screen mode
- for buttons
- for emoticon display
- for character & expression display
- for shadow of UI
- for getting characters from xml file
- for getting expressions from xml file
- for getting emoticons from xml file
- storage for emoticons to draw
- storage for text to draw
- mask for eDisplay
- storage for emoticon to save
- storage for emoticon to animate
- for animation

```
enum Mode
{
  START,
    STOP_POSTURE,
    MOTION_POSTURE,
    ADD_MOTION_EXPRESSION,
    SECOND_MOTION,
    MOTION_CHECKING,
    ADD_EXPRESSION,
    ADD_TEXT,
    ADD_TEXT_MOTION,
    SAVE,
    SAVE_MOTION
};
```




```
void start() {
  void mousePressed() {
    void mouseReleased() {
      void mouseDragged() {
        switch (mode) {
        case START:
          break;
        case STOP_POSTURE:
          if (allCharacters == null) return;
          for (Character c : allCharacters) {
            if (c.isOver(mouseX, mouseY)) {
              c.x = mouseX;
              c.y = mouseY;
              if (eDisplay.isOver(mouseX, mouseY))
              {
                Emoticon emoticon = null;
                for (Emoticon e : allEmoticons) {
                  if (e.name.equals(c.name)) {
                    emoticon = e;
                  }
                }
```

Emoji has 10 modes. And emoji behave differently if mode is different. Mainly Emoji has 2 stream which are making normal emoticon and making motion emoticon

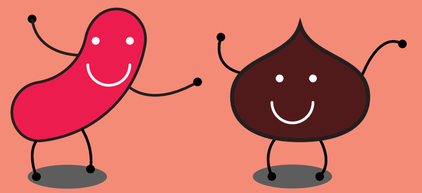
These class are the class that decides the behavior of Emoji. So they contain switches with mode.

```
void keyPressed() {
  if (mode == Mode.ADD_TEXT || mode == Mode.ADD_TEXT_MOTION) {
    if (key == ENTER || key == RETURN) {
      hideVirtualKeyboard();
    } else if ((int) key == 65535 && keyCode == 67) {
      if (emoticonText.text.length()>1) {
        emoticonText.setText(emoticonText.text.substring(0, emo
      } else {
        emoticonText.setText("");
      }
    } else {
      emoticonText.setText(emoticonText.text+key);
    }
  }
}
```

keypressed function has also switch with mode. To differ the shift and delete, I also use keyCode. Thay have same int number value.

```
void reset () {
  mode = Mode.START;
  stopButton = new RectButton(width/2
  motionButton = new RectButton(width
  saveButton = new RectButton(width/2
  arrowButton = new ArrowButton(9*wid
  eDisplay = new eDisplay(width/2, 13
  cStorage = new cStorage();
  shadowB = new Shadow(0, height/10,
  shadowCS = new Shadow(0, 11*height/
  characterFactory = new CharacterFac
  characterFactory.loadXMLfile("chara
  allCharacters = characterFactory.ge
  expressionFactory = new ExpressionF
  expressionFactory.loadXMLfile("expr
  allExpressions = expressionFactory.
```

To reset the whole emoji when making emoticon process is done.



Emoticon Class

```
class Emoticon {
  //lLegPivotPos, rLegPivotPos, lArmPivotPos, rArmPivotPos
  private ArrayList<PVector> startPivotPos;
  private ArrayList<PVector> middlePivotPos;
  private ArrayList<PVector> endPivotPos;
  private PVector bodyPivotPos;
  private PVector facePivotPos;
  private PVector maniPivotPos;
  private PVector transPivotPos;
  private float w, h;
  private int x, y;
  private String name;
  private float ratio;
  private String filename;
  private PImage img;
  private boolean[] isMani;
  private boolean isTrans;
  private String expFilename;
  private PImage expImg;
```

for Pivot points of Arm & Legs

for Pivot points of Body & FACE

Temporary Pivopoints for manipulation & translation

storage for width, hiehgt x ,y

emoticon name

emoticon scale in app

filename of emoticon

image of emoticon

boolean for managing

filename of expression

image of expression

```
void draw() {
  imageMode(CENTER);
  image(img, x, y, ratio, h/w*ratio);
  for (int i=0; i<4; i++) {
    noFill();
    stroke(0);
    strokeWeight(4);
    beginShape();
    vertex(startPivotPos.get(i).x, startPivotPos.get(
    quadraticVertex(middlePivotPos.get(i).x, middlePi
    endShape();
    fill(0);
    ellipse(endPivotPos.get(i).x, endPivotPos.get(i).
  }
}
```

```
void drawExpression() {
  imageMode(CENTER);
  image(expImg, facePivotPos.x, facePiv
}
```

```
void setExpression(String filename) {
  this.expFilename = filename;
  this.expImg = loadImage(filename);
}
```

These functions are for draw emoticons & expression. It is devided into two with emoticon and

```
void manipulate(float mx, float my, int i) {
  float dx = mx-maniPivotPos.x;
  f void translate(int mx, int my) {
      i  void scale(float s) {
  f   i    this.ratio = this.ratio*s;

  f   tl
  f   tl     facePivotPos = new PVector((facePivotPos.x-x)*s+x, (facePivo
  f
      fa     for (int i=0; i<4; i++) {
  f            this.startPivotPos.set(i, new PVector((startPivotPos.get(i
  f   fo     }

      }      this.middlePivotPos.set(0, new PVector((this.middlePivotPos.
             this.middlePivotPos.set(1, new PVector((this.middlePivotPos.
             this.middlePivotPos.set(2, new PVector((this.middlePivotPos.
      tl     this.middlePivotPos.set(3, new PVector((this.middlePivotPos.
      tl
```

To manipulate each pivots, manipulate functions get mx, my, and int i. mx, my are for calcuating manipulated pivot points. And int i is for varify which pivot is manipulated. Also translate need mx, my for calculate translation. And scale need float s for calcuate ratio.

```
void pgDraw(PGraphics pg) {
  pg.imageMode(CENTER);
  pg.image(img, x, y, ratio, h/w*ratio);
  for (int i=0; i<4; i++) {
    pg.noFill();
    pg.stroke(0);
    pg.strokeWeight(4);
    pg.beginShape();
    pg.vertex(startPivotPos.get(i).x, startPivotPo
```

```
void pgDrawExpression(PGraphics pg) {
  pg.imageMode(CENTER);
  pg.image(expImg, facePivotPos.x, fac
}
```

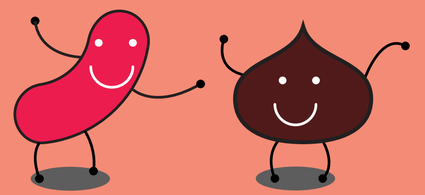
To save emotion, all the emoticon drawing goes into PGraphics. For that, I implemented extra draw function that is for drawing emoticons in PGraphics.

```
void manipulateInverse(float hx, float hy, int i) {
  float dx = hx-maniPivotPos.x;
  float dy = hy-maniPivotPos.y;

  float distance = sqrt(dx*dx+dy*dy);

  float a= ratio/5;
  float b = ratio/5;
  float c = min(distance, a+b);

  float b1 = (b*b-a*a-c*c)/(-2*a*c);
  float c1 = (c*c-a*a-b*b)/(-2*a*b);
```

manipulation for animation.

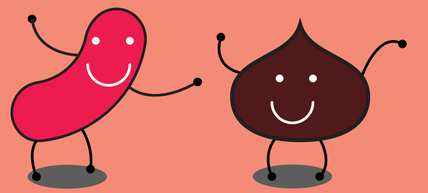
```
boolean isOverPivot(int mx, int my, int i) {
  boolean isOver(int mx, int my) {
    if (mx<x-ratio/2 || mx>x+ratio/2) return false;
    if (my<y-h/w*ratio/2 || my>y+h/w*ratio/2) return false;
}   return true;
  }
```

IsOver functions that varify that mouseX, mouseY is over the emoticons or Pivots



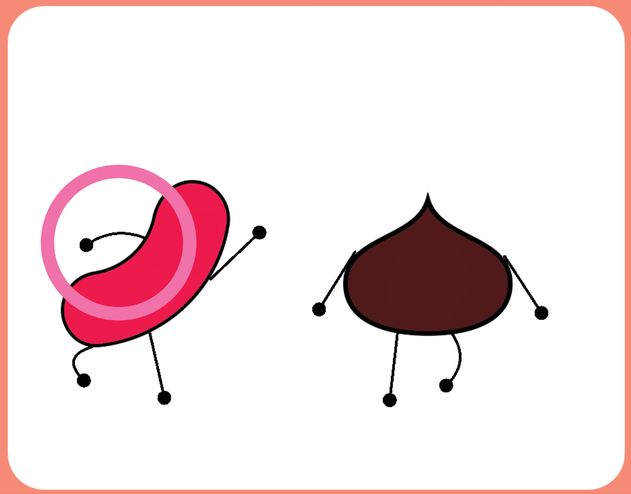
Library?

```
import ketai.ui.*;
import android.view.MotionEvent;
import android.view.inputmethod.InputMethodManager;
import android.content.Context;
```

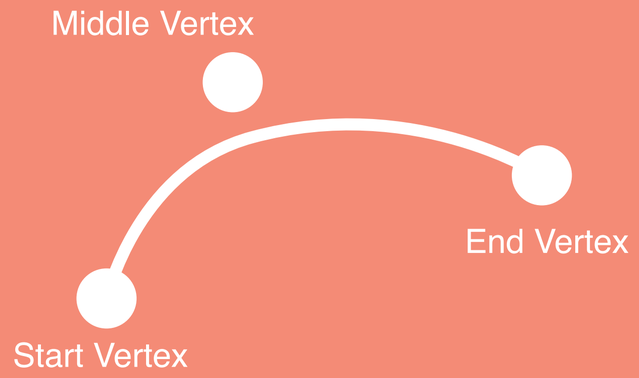


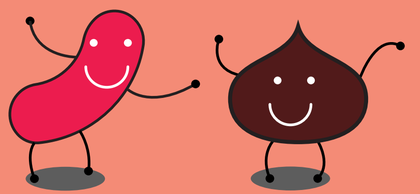
Flexible Arms & Legs

```
void draw() {
  imageMode(CENTER);
  image(img, x, y, ratio, h/w*ratio);
  for (int i=0; i<4; i++) {
    noFill();
    stroke(0);
    strokeWeight(4);
    beginShape();
    vertex(startPivotPos.get(i).x, startPivotPos.get(i).y);
    quadraticVertex(middlePivotPos.get(i).x, middlePivotPos.get(i).y, endPivotPos.get(i).x, endPivotPos.get(i).y);
    endShape();
    fill(0);
    ellipse(endPivotPos.get(i).x, endPivotPos.get(i).y, ratio/15, ratio/15);
  }
}
```



For making flexible Arms, I use quadrat-icVertex function





Inverse Kinematics

```
void manipulate(float mx, float my, int i) {
  float dx = mx-maniPivotPos.x;
  float dy = my-maniPivotPos.y;

  float distance = sqrt(dx*dx+dy*dy);

  float a= ratio/5;
  float b = ratio/5;
  float c = min(distance, a+b);

  float b1 = (b*b-a*a-c*c)/(-2*a*c);
  float c1 = (c*c-a*a-b*b)/(-2*a*b);

  if (b1>1.0) {
    b1=1.0;
  } else if (b1<-1.0) {
    b1=-1.0;
  }

  if (c1>1.0) {
    c1=1.0;
  } else if (c1<-1.0) {
    c1=-1.0;
  }

  float B = (1-2*(i%2))*acos(b1);
  float C = (1-2*(i%2))*acos(c1);
  float D = atan2(dy, dx);
  float E = D+B+C+PI;

  float ex = (cos(E) * a) + startPivotPos.get(i).x;
  float ey = (sin(E) * a) + startPivotPos.get(i).y;
  //print("UpperArm Angle=  "+degrees(E)+"    ");

  float hx = (cos(D+B) * b) + ex;
  float hy = (sin(D+B) * b) + ey;
  //println("LowerArm Angle=  "+degrees((D+B)));

  middlePivotPos.set(i, new PVector(ex, ey));
  endPivotPos.set(i, new PVector(hx, hy));
}
```

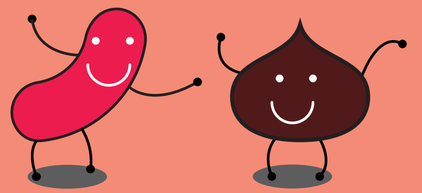
dx and dy is for calcuating end and middle points.
So it requires to get how this end points and middle poitns point gonna move. So I put mouseX, mouseY in mx, my when user drag the mouse. And maniPivotPos.x is temporary points that saves mouseX, mouseY when user presse mouse.

Often b1 or c1 is become 1.0000000001 or -1.00000001. And these makes a lot of error. So I put these to make 1.000000001 to 1 and -1.000000001 to -1.

Also middle points need to be filpped when pivot's side is different. For exapm-ple, left arms are swollen to left side and right arms are swollen to right side. So I put (1-2*1%2) to flip the middle points.

```
void manipulateInverse(float hx, float hy, int i) {
  float dx = hx-maniPivotPos.x;
  float dy = hy-maniPivotPos.y;

  float distance = sqrt(dx*dx+dy*dy);

  float a= ratio/5;
  float b = ratio/5;
  float c = min(distance, a+b);

  float b1 = (b*b-a*a-c*c)/(-2*a*c);
  float c1 = (c*c-a*a-b*b)/(-2*a*b);
```

When we draw animation we know how end point gonna move. So we need to guess middle point from end point. These function is for that.



Animation + Thread

```
void animate() {
  tempEmoticons = new ArrayList<Emoticon>();

  for (int i =0; i<startEmoticonsDraw.size(); i++) {
    Emoticon e1 = startEmoticonsDraw.get(i);
    Emoticon e2 = endEmoticonsDraw.get(i);
    Emoticon e3 = new Emoticon (e1);
    if (frame>0) {
      e3.setTransPivotPos(new PVector(0, 0));
      e3.translate((e2.getX()-e3.getX())/totalFrame*frame, (e2.getY()-e3.getY())/totalFrame*frame);
      e3.scale(1+(e2.getRatio()/e3.getRatio()-1)*frame/totalFrame);

      for (int j= 0; j<4; j++) {
        PVector e2j = e2.getEndPivotPos().get(j);
        PVector e3j = e3.getEndPivotPos().get(j);
        e3.setManiPivotPos(new PVector(e3j.x+(e2j.x-e3j.x)/totalFrame*(frame-flow), e3j.y+(e2j.y-e3j.y)/totalFrame*(frame-flow)));
        if (e2j.x-e3j.x != 0 && e2j.y-e3j.y != 0) {
          e3.manipulateInverse(e3j.x+(e2j.x-e3j.x)/totalFrame*frame, e3j.y+(e2j.y-e3j.y)/totalFrame*frame, j);
        }
      }
    }

    if (frame > totalFrame || frame < 1) {
      flow *= -1;
      frame += flow;
    } else {
      frame += flow;
    }

    tempEmoticons.add(e3);
  }

  resultEmoticons = new ArrayList<Emoticon> (tempEmoticons);
}

public void run () {
  while (true) {
    if (startEmoticonsDraw.size()>0 && endEmoticonsDraw.size()>0 &&
    endEmoticonsDraw.size() == startEmoticonsDraw.size()) {
      animate();
      try {
      Thread.sleep(20);
    } catch (Exception e) {}
    }
  }
}
```

Get first emoticon e1 & second emoticon e2. Make a copy e3 of first emoticon. save animated emoticon in e3.

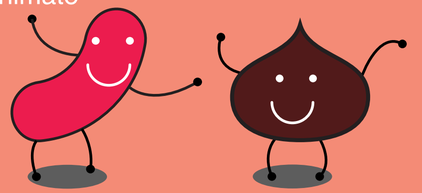
caclulate animated emoticon

count frame

save all the emoticons in tempEmoticons.

update resultEmoticons which is gonna be drawn in animate.draw().

Using thread, control how often animate function performs.



Save Processing Image

```
pg = createGraphics(width, height, P2D);
```

put the renderer in size() or fullScreen()

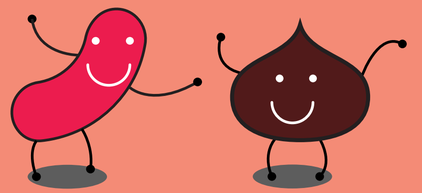
create PGraphics for save image with only emoticons. It is really important to put renderer for android. If not, image is not saved properly.

```
pg.beginDraw();
eDisplay.pgDraw(pg);
if (emoticonsDraw != null) {
  for (Emoticon e : emoticonsDraw) {
    e.pgDraw(pg);
    if (e.expImg != null) {
      e.pgDrawExpression(pg);
    }
  }
}

if (emoticonText != null && emoticonText.text.length()>0) {
  emoticonText.pgDrawText(pg);
}
pg.endDraw();

imageMode(CENTER);
image(pg, width/2, height/2);
```

put all the drawing in PGraphics

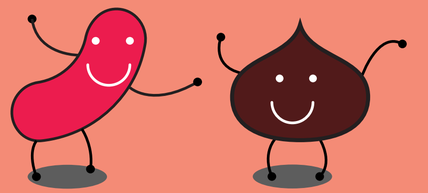
```
case SAVE:
  if (saveButton.isOver(mouseX, mouseY)) {
    pg.save("/sdcard/DCIM/Camera/abc.png");
    mode = Mode.START;
    reset();
  }
```

save PGraphics as image.



Further more?

save motion emoticon in gif



Thanks.