CSTA Processing Workshop 2013

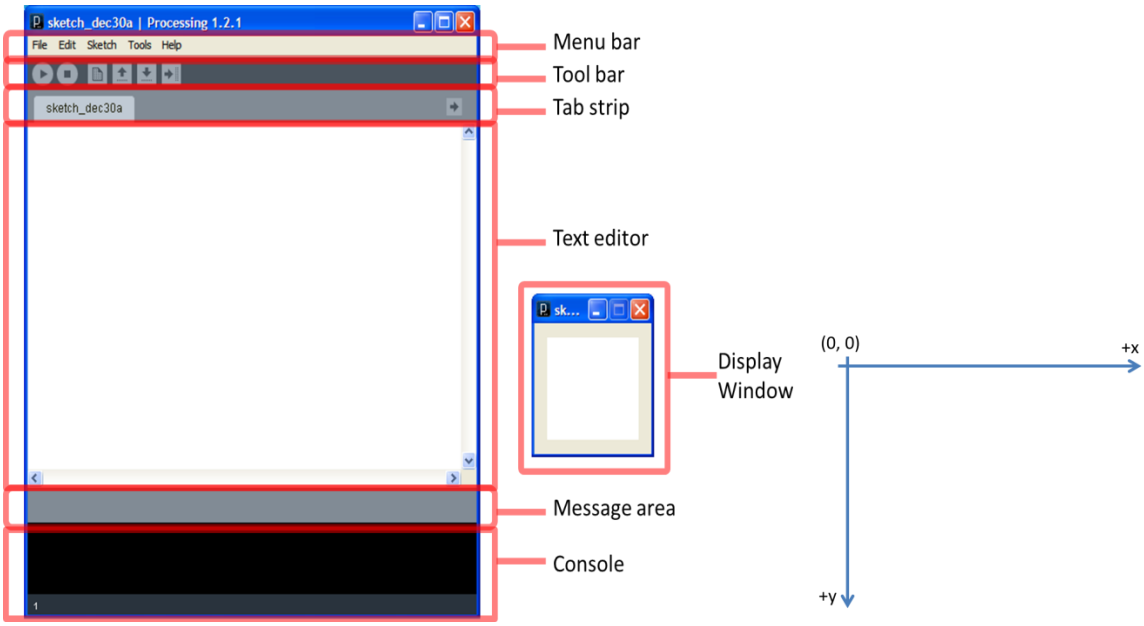
Presenter: Dr. Bina Ramamurthy (bina@buffalo.edu)
CSE Department, University at Buffalo, NY 14260

Introduction: Processing is a programming environment for problem solving and implementing solutions using highly intuitive commands. See http://processing.org/

Goal of the workshop: Introduce the workshop participants to (i) Processing development environment (PDE), (ii) problem solving using processing and (iii) software development using processing.

Instructions:

1. Locate the Processing icon on your desktop or Start menu and click on it.
2. Lets now go through the Processing environment and understand the various regions and buttons. (This picture is from Bryn Mawr's Dr. Deppak Kumar's slides, with permission)



3. **Sketch 1: {setup and draw functions}**

   Now lets write our first Processing program (called a **sketch**). A typical processing program has a ***setup*** function that is executed once at the start for initialization purposes and a ***draw*** function that is a continuous loop and that is also an event handler for drawing commands.

```
void setup()
{
   size(600, 600);  // set the size of the canvas
  background(255);   }
```

```
void draw()
{
   fill(110,10,200); // color of the pen;  fill color of shapes RGB: 0-255, 0-255, 0-255
   rect( 300, 300, 50,50); // draw rectangle at location x = 300, y =300, size 50X50
   ellipse(400,450, 50,50);// draw a circle
}
```

4. **Sketch 2**: Lets do another sketch with some images and dynamics (motion). You need the image Serengeti.jpg for this. You can download any Africa image for this and store in a directory called **data** in the current directory of the sketch. Save as Africa.pde

```
PImage africa;

void setup()
{ size(500,500);
  africa = loadImage("serengeti1.jpg");
  image(africa,0,0, 500,500);
  frameRate(10);
}
int x=0; int y =0;
void draw()
{ image(africa,0,0,500,500);
  fill(0);
  translate(x,y);
  rect(0,300, 40,15);
  rect(20,290, 10,10);
  x = x+1;
  if (x>width) x = 0;
}
```

5. **Sketch 3: Final exercise : Memory Game: See the discussion in the slides: here is the code:**

```
//constants
final int n = 4;  // size of the board
final int numTiles = n * n;
// arrays: board layout
int []tiles = new int[numTiles]; //all the tiles
int tileW, tileH;  // individual tile width and height for computation
PImage photo; // the image holder
int numCorrect =0;

void setup()
{
  size (600,600);
  tileW = width / n;
  tileH = width/n;
  initializeBoard(n);
}
```

```
void draw()
{

}

boolean noMatch = false;
boolean selection1 = true;

void mousePressed()
{ if (noMatch) { closeTiles(); selection1 = true;}
  if (selection1)
    {openTile1(); }
    else {openTile2(); matchPair();}
 }

void initializeBoard(int n)
{
  // create a blank board
  fill(0,255,0);
  for (int row=0;row<= n; row++) // illustrating two approaches to the board abstraction
     for (int col=0; col <= n; col++)
       rect(row*tileW, col*tileH, width/n, height/n);

  for (int i=0;i<numTiles; i++)
    tiles[i] = -1;  // unassigned state for the array representing

  int rNum;
  for (int pairNum=0; pairNum <numTiles/2; pairNum++) //n pairs of pictures
  {
    placeRandomPair(pairNum); // fill the first tile of the pair
    placeRandomPair(pairNum); // fill the second tile of the pair
  }
  println(tiles);
}

void placeRandomPair(int j) // place the random pairs in the array representing board
{
    int rNum = (int)random(numTiles);
    while( tiles[rNum]!= -1)
    {
       rNum = (int)random(numTiles);
    }
    tiles[rNum] = j;
}


int row1, col1, tileNum1; // these information are needed for closing the tile on no match
void openTile1()
```

```
{
  row1 = mouseX/tileW;
  col1 = mouseY/tileH;
  tileNum1 = col1*n + row1;
  photo = loadImage("pic"+tiles[tileNum1]+".jpg");
  image(photo, row1*tileW, col1*tileH, tileW, tileH);
  selection1 = false;
}

int row2, col2, tileNum2;
void openTile2()
{
  row2 = mouseX/tileW;
  col2 = mouseY/tileH;
  tileNum2 = col2*n + row2;
  photo = loadImage("pic"+tiles[tileNum2]+".jpg");
  image(photo, row2*tileW, col2*tileH, tileW, tileH);
}

int numTry =0;
void matchPair()
{ numTry++;
  println("Tiles are" + tiles[tileNum1] + tiles[tileNum2]);
  println("tiles"+ tileNum1 + tileNum2);
  if (tiles[tileNum1] == tiles[tileNum2])
  { numCorrect++;
    println("match..Match" + tileNum1 + tileNum2);
    println( " correct " + numCorrect);
    if (numCorrect == (numTiles/2))
     { textSize(40);
       text("Game over", 100, 100);
       text("Number of Tries = " + numTry, 30,300);
     }

  }
  else
  { // no match
    println(" No match");
    noMatch = true;
  }
  selection1 =true;
}
void closeTiles()
{
   rect(row1*tileW, col1*tileH, width/n, height/n);
   rect(row2*tileW, col2*tileH, width/n, height/n);
   noMatch = false;
}
```