ECS100 Java Documentation

Brief, partial documentation of UI and other Java classes, with examples of usage and common patterns.

UI **class**:

```
// Text output
  public void clearText()                    // Clears the text pane
  public void print(anytype val)             // Prints val with no newline, val can be any type including array or list
  public void println(anytype val)           // Prints val and newline, val can be any type including array or list
  public void println()                      // Prints a newline
  public void printf(String format, ...)     // Prints the format string, inserting remaining arguments at the %'s:
                                             //   %s for Strings.
                                             //   %d for ints, (%3d: %pad to at least 3 characters),
// Text input                                //   %6.2f for 2dp %doubles, pad to at least 6 characters
  public String askString(String question)       // ask the user for a line of text
  public int askInt(String question)             // ask the user for an integer
  public double askDouble(String question)       // ask the user for a number
  public String askToken(String question)        // ask the user for a single token
  public boolean askBoolean(String question)     // ask the user for a yes/no or true/false value
  public ArrayList<Double> askNumbers(String question) // ask the user for a list of numbers
  public ArrayList<String> askStrings(String question) // ask the user for a list of Strings
// Graphics output
  public void clearGraphics()                    // Clears the graphics pane
  public void setColor(Color c)                  // Change the colour for later drawing commands
  public void setFontSize(double s)              // Change the size of the Font for later drawString commands
  public void setLineWidth(double width)         // Change the width of lines for later drawing commands
// draw outlines of shapes
  public void drawLine(double x1, double y1, double x2, double y2)
  public void drawRect(double left, double top, double width, double height)
  public void drawOval(double left, double top, double width, double height)
  public void drawArc(double left, double top, double width, double height, double startAngle, double arcAngle)
  public void drawPolygon(double[] xPoints, double[] yPoints, int numPoints)
// draw solid shapes (not outlines)
  public void fillRect(double left, double top, double width, double height)
  public void fillOval(double left, double top, double width, double height)   // Draws solid oval
  public void fillArc(double left, double top, double width, double height, double startAngle, double arcAngle)
  public void fillPolygon(double[] xPoints, double[] yPoints, int numPoints)
// draw strings and images
  public void drawString(String s, double left, double baseline)
  public void drawImage(String filename, double left, double top)
  public void drawImage(String filename, double left, double top, double width, double height)
  // Note: all the drawXXX commands except drawImage also have an eraseXXX and invertXXX form
// other methods
  public void sleep(double millis) // pause program for specified time (milliseconds).
  public void initialise()         // ensure UI window has been initialised
  public void quit()               // delete UI window; usually halts the program.
```

```
// Event-based input
public void addButton(String name, mth) // Add a button to input panel, mth is a method with no parameters
public void addTextField(String s, mth) // Add a textField to input panel, mth is a method with a String parameter
public void addSlider(String s, double min, double max, mth) // Add a slider to input panel, mth is a method
public void addSlider(String s, double min, double max, double initial , mth)  //  with a double parameter
public void setMouseListener(mth) // Set mouseListener, mth is method with a String and two double parameters
public void setMouseMotionListener(mth) // Set mouseMotionListener,
public void setKeyListener(mth) // Set keyListener for Graphics pane, mth is a method with a String parameter
```

String **class**:

```
public int length()                          // Returns the length (number of characters) of the string
public boolean equals(String s)              // Returns true if string has same characters as s
public boolean equalsIgnoreCase(String s)    // String has same characters as s, ignoring their case
public String toUpperCase()                  // Returns upper case copy of string
public String toLowerCase()                  // Returns lower case copy of string
public boolean startsWith(String patrn)      // Returns true if first part of string matches patrn
public boolean endsWith(String patrn)        // Returns true if last part of string matches patrn
public boolean contains(String patrn)        // Returns true if patrn matches some part of the string
public int compareTo(String other)           // Returns -ve if this string is alphabetically before other,
                                             //         +ve if after other, 0 if equal to other
public String substring(int j, int k)        // Returns substring from index j to index k-1
                                             //  requires 0 <= j <= k <= length of string
public String trim ():                       // Returns copy of string with spaces at ends removed
public int indexOf(String patrn)             // Returns the index of where patrn first matches
                                             // Returns -1 if string does not contain patrn anywhere
```

Color **class**:

```
Color.gray, Color.blue, Color.red, Color.green, Color.black, Color.white // Some of the predefined colours
public Color(int red, int green, int blue)        // Make a colour; arguments must be 0..255
public Color(float red, float green, float blue)  // Make a colour; arguments must be 0.0 to 1.0
```

Integer **class**:

```
public static final int MAX_VALUE    // The largest possible int: 2^(31)-1
public static final int MIN_VALUE    // The smallest possible int: -2^31
```

Double **class**:

```
public static final double MAX_VALUE          // The largest possible double: just under 2^(1024)
public static final double MIN_VALUE          // The smallest possible positive nonzero double
public static final double POSITIVE_INFINITY // positive infinity (greater than any number)
public static final double NEGATIVE_INFINITY // negative infinity (less than any number)
public static final double NaN                // The double that is 'Not a Number'
```

Math **class**:

```
public static double sqrt(double x)              // Returns the square root of x
public static double min(double x, double y)     // Returns the smaller of x and y
public static double max(double x, double y)     // Returns the larger of x and y
public static double abs(double x)               // Returns the absolute value of x
public static int min(int x, int y)              // Returns the smaller of x and y
public static int max(int x, int y)              // Returns the larger of x and y
public static int abs(int x)                     // Returns the absolute value of x
public static double random()                    // Returns a random number between 0 and 1.0
public static double hypot(double dx, double dy) // Returns sqrt(dx*dx + dy*dy)
```

UIFileChooser **class**:

```
public static String open()              // Ask user for an existing file
public static String open(String prompt)
public static String save()              // Ask user for a new or existing file
public static String save(String prompt)
```

Path **class** and Files **class**:

```
Path: public Path of (String fname)      // Returns the path of the file named fname
Files: public List<String> readAllLines (Path p)   // Read all lines of a file from the path to the file
```

Scanner **class**:

```
public Scanner (String s)          // Constructor, for reading from a string
public boolean hasNext()           // Returns true if there is more to read
public boolean hasNextInt()        // Returns true if the next token is an integer
public boolean hasNextDouble()     // Returns true if the next token is a number
public String next()               // Returns the next token (chars up to a space/line)
public String nextLine()           // Returns string of chars up to next newline
                                   //  (next and nextLine throw exception if no more tokens)
public int nextInt()               // Returns the integer value of the next token
                                   //  (throws exception if next token is not an integer or no more tokens)
public double nextDouble()         // Returns the double value of the next token
                                   //  (throws exception if next token is not a number or no more tokens)
public void close()                // Closes the file (if it is wrapping a File object)
```

PrintStream **class**:

```
                                   // Note, System.out is also a PrintStream object
public PrintStream (String filename)   // Constructor, for printing to a file
public void close()                // Close the file
public void print (anytype value)  // Prints the value with no newline, value can be any type
public void println ()             // Prints a newline
public void println (anytype value)// Prints the value followed by newline, value can be any type
public void printf (String format, ...)  // Prints the format string, inserting arguments at the %'s.
```

Note: UI is not a PrintStream, but it has the same print... methods, printing to the UI text pane.

ArrayList **class**:

```
// Assumes that itemType is the type of the items in the ArrayList
public int size()                        // Returns the current size (number of items) of the list
public boolean isEmpty()                 // Returns true if list is currently empty
public boolean add(itemType val)         // Adds val to the end of the ArrayList, returns true if successful
public void add(int index, itemType val) // Inserts val at position index, moving later items up
public itemType get(int index)           // Returns the item at position index
public itemType set(int index, itemType val) // Puts val at position index, returns the old value
public itemType remove(int index)        // Removes and returns item at position index, moving later items down
public boolean contains(itemType val)    // Returns true if val is currently in the list
public int indexOf(itemType val)         // Returns position of first occurrence of val; -1 if not present
public boolean remove(itemType val)      // Removes 1st instance of val from list. Returns false if val not present
public void clear()                      // Removes all items from the list
```

Comparison Operators:

```
==    // ''is the same value''. For objects, compares their ID's. You should usually use .equals(...) on objects
!=    // ''is not the same value''. Use to check if a value is not null
<  >  <=  >=
```

Logical Operators:

```
&&          // and
||          // or
!           // not (prefix operator)
```

Increment *Operators*:

```
var++  var--        // add (subtract) 1 from var, [returning previous value of variable]
++var  --var        // add (subtract) 1 from var, [returning new value of variable]
var += value; [ or  -= *= /= ]   // add [or subtract,] value to var
```

break and **return**:

```
break               // exit the immediately enclosing loop (for or while)
return              // exit the method (no value returned)
return <expression> // exit the method, returning the value of the expression
```

Some Common Patterns:

```
//Using Math
    if ( Math.random()<0.1) { ...                  // do something with probablity 0.1
    int size = (int)(Math.random()*50);            // a random integer between 0 and 49.
    double size = Math.min(400, UI.askDouble("size")); // ensure size is no more than 400.
    double diagonal = Math.hypot(wd, ht);   same as   double diagonal = Math.sqrt((wd*wd) + (ht*ht));
```

```
// Using Strings (assume name, answer, and courseCode are all variables of type String)
    if ( answer.equals(name) ) { ...      OR      if ( answer.equalsIgnoreCase(name) ) { ...
    UI.println("Ans:"+ answer.toLowerCase());
    if ( name.startsWith("Pe") ) { ...
    UI.println("Course number =" + courseCode.substring(4, 7));
```

```
//  setting up a GUI with UI:
    UI.addButton("Load", this::loadData);
    UI.addTextField("Name", this::setName);
    UI.addSlider("Size", 1, 30, 5, this::setSize);
    UI.setMouseListener(this::doMouse);
    UI.addButton("Quit", UI::quit);

    public void loadData(){...
    public void setName(String v){...
    public void setSize(double v){...
    public void doMouse(String action, double x, double y){
        if (action.equals("pressed")) {....
        else if (action.equals("released")){.....
```

```
//Using Color
    UI.setColor(Color.blue);
    UI.setColor(Color.blue.darker());
    Color col = new Color(red, green, blue);
    Color brightCol = Color.getHSBColor((float)Math.random(),1.0f, 1.0f);
```

```
// if and if .. else
   if (name.startsWith("p") ){
      UI. println ("report to "+name);
   }

   if (Math.hypot(x–this.xCen, y–yCen) < this.rad){
       this.moveLeft(50);
   }
   else if (x > this.xCen–this.rad && x < this.xCen+this.rad){
       this.jump(5);
   }
   else if (Math.abs(x–this.xCen)<=2 && y >= this.yCen){
       this. fall ();
   }
   else {
       this.step ();
   }
```

```
// SwitchExpressions: (works for integers and strings (and Enums))
   switch (day){
      case "Mon", "Tue"–> {UI.println("12–5pm");}
      case "Sat" –> {UI.println("9am–1pm");}
      case "Sun" –> {UI.println("Closed");}
      default –> {UI.println("9am–6pm");}
   }
```

```
// Stepping through a list of numbers with a for–each loop
    double min = Double.POSITIVE_INFINITY; // find minimum of an ArrayList of doubles
    for (double num : numbers){
        if (num < min){
           min = num;
        }
    }
```

```
// Stepping through a list of objects with a for–each loop
    for (Ball b : poolBalls){
        b.step ();
        if (b.getX()>=HALF_WAY){
           b.slow(0.5);
        }
        else if (b.getX() >= TARGET){
           b.reset ();
        }
    }
```

```
//Using a ''counted'' for loop
    for (int i = 0; i < count; i++){     // draw a sequence of count circles
        UI.drawOval(LEFT+i*width, TOP, width, height);
    }
    // draw a sequence of circles from LEFT to RIGHT
    for (double left = LEFT; left < RIGHT; left = left +width){
        UI.drawOval(left, TOP, width, height);
    }
    // print powers of 3 up to max
    for (int n=3; n < max; n=n*3){
        UI.println(n);
    }
```

```
//Using nested for loops for drawing a grid of colours
    for (int row = 0; row < rows; row++){
       for (int col = 0; col < cols; col++){
            Color colr = new Color(row, col, 0); // create a red/green color
            UI.setColor(colr);
            UI.fillRect(TOP+row*SIZE, LEFT+col*SIZE, SIZE, SIZE);
       }
    }
```

```
//using a while loop to get valid input
    String passKey = UI.askString("Enter new passKey");
    while ( ! this.isValid(passKey)){
        UI.println(passKey + " is not valid, try again.");
        passKey = UI.askString("Enter new passKey");
    }
//OR (using a break)
    String passKey = "";
    while (true){
        passKey = UI.askString("Enter new passKey");
        if (this.isValid(passKey)){ break; }
        UI.println(passKey + " is not valid, try again.");
    }
```

```
//Processing a file, line by line, pulling values out of the lines with a Scanner
   try{
        String filename = UIFileChooser.open("Choose data file to process");
        List<String> allLines = Files.readAllLines(Path.of(filename));
        for (String line : allLines) {
            Scanner sc = new Scanner(line);
            String town = sc.next();           // extract value from the line
            double dist = sc.nextDouble();     // extract value from the line
            if (dist < 50) {                   //use the values
                UI.printf("%s (%.1fkm away) is a candidate\n", town, distance);
            }
        }
   }catch(IOException e){UI.println("File reading failed: "+e);}
```

```
// Processing file using a scanner (not line-by-line)
   try{    // add up all the numbers in the file, ignoring non-numbers.
        Scanner scan = new Scanner(Path.of(UIFileChooser.open()));
        double total = 0;
        while (scan.hasNext()) {
            if (scan.hasNextDouble()) { total += scan.nextDouble(); }
            else { scan.next(); }
        }
        scan.close();
   } catch(IOException e){UI.println("File reading failed: "+e);}
```

```
// Processing file with header lines using a scanner (not line-by-line) and creating an arraylist.
   try{
       ArrayList<Item> allItems = new ArrayList<Item>();
       Scanner scan = new Scanner(Path.of("backupData.txt"));
       String title = scan.next();
       int numItems = scan.nextInt();
       for (int i=0; i<numItems; i++){
           String name = scan.next();
           int size = scan.nextInt();
           allItems.add(new Item(name, size));
       }
    } catch(IOException e){UI.println("File reading failed: "+e);}
```

```
// Using ArrayLists
     for (Item item : allItems){  item.display();  }

     for (int i=0; i<allItems.size()-1; i++){
         Item item1 = allItems.get(i);
         if (item1.matches(allItems.get(i+1))){
             duplicateItems.add(item1);
             i++;
         }
     }
```

```
// Using a 1D array of objects
   for(int i = 0; i<rack.length; i++){
       if (rack[i]==null){          rack[i] = new Tile(); }
       else if (rack[i].isFull()){  rack[i] = null;       }
       else {                       rack[i].expand(2);    }
   }
```

```
// using a 2D array of numbers.
   for (int row=0; row<matrix.length; row++){
      for(int col=0; col<matrix[0].length; col++){
           double temp = matrix[row][col];
           matrix[row][col] = matrix[col][row];
           matrix[col][row] = temp;
      }
   }
```