COSC 1P02
Method Summary

`Math`

Method/Value	meaning
`E`	*constant*: the mathematical constant `e`
`PI`	*constant*: the mathematical constant `π`
v `= abs(`*x*`)`	returns the absolute value of *x*
v `= acos(`*x*`)`	returns the arc cosine of *x*
v `= asin(`*x*`)`	returns the arc sine of *x*
v `= atan(`*x*`)`	returns the arc tangent of *x*
v `= cos(`*x*`)`	returns the cosine of *x*
v `= log(`*x*`)`	returns the natural logarithm of *x*
v `= pow(`*a*`,`*b*`)`	returns a^b
v `= random()`	returns a random value between `0.0` and `1.0`
v `= sin(`*x*`)`	returns the sine of *x*
v `= sqrt(`*x*`)`	returns the square root of *x*
v `= tan(`*x*`)`	returns the tangent of *x*

`Turtle`

method	meaning
`SLOW`, `MEDIUM`, `FAST`	*constants*: turtle speeds
t `= new Turtle()`	*constructor*: creates a turtle object
t `= new Turtle(`*speed*`)`	*constructor*: creates a turtle object drawing at *speed*.
t`.backward(`*units*`)`	move backwards *units* drawing units
t`.forward(`*units*`)`	move forward *units* drawing units
t`.left(`*radians*`)`	turn left radians *radians*
t`.moveTo(`*x*`,`*y*`)`	move turtle to coordinates `(`*x*`,`*y*`)`
t`.penDown()`	place the pen on the canvas
t`.penUp()`	raise the pen from the canvas
t`.right(`*radians*`)`	turn right radians *radians*
t`.setPenColor(`*color*`)`	change the turtle's pen color to *color*
t`.setPenWidth(`*width*`)`	change the turtle's pen width to *width*

`PictureDisplayer`

method	meaning
d `= new PictureDisplayer()`	*constructor*: creates a new displayer with canvas 200x200
d `= new PictureDisplayer(`*pic*`)`	*constructor*: creates a new displayer with canvas to fit *pic* and with *pic* placed on displayer
d `= new PictureDisplayer(`*width*`,` *height*`)`	*constructor*: creates a new displayer with canvas of specified *height* and *width*
d`.close()`	wait until user presses `Close` button and close displayer
d`.placePicture(`*pic*`)`	place *pic* on the displayer
d`.waitForUser()`	wait until user presses `OK` before continuing

`Picture`

method	meaning
`p = new Picture()`	*constructor*: creates a picture object loading pixels from a file selected via a file open dialog
`p = new Picture(width,height)`	*constructor*: creates a picture object with specified `height` and `width` with all pixels `white`
`i = p.getHeight()`	returns height (in pixels) of picture
`q = p.getPixel(x,y)`	returns pixel in column `x` of row `y`
`i = p.getWidth()`	returns width (in pixels) of picture
`b = p.hasNext()`	returns `true` if another pixel is available
`q = p.next()`	returns the next available pixel
`p.save()`	present file save dialog to allow user to save picture as modified

`Pixel`

method	meaning
`i = q.getBlue()`	obtain blue color channel of pixel
`c = q.getColor()`	obtain color of pixel
`r = q.getDistance(color)`	returns the color distance between this pixel's color and `color`
`i = q.getGreen()`	obtain green color channel of pixel
`i = q.getRed()`	obtain red color channel of pixel
`q.setBlue(v);`	change blue channel value to `v`
`q.setColor(color)`	change color of pixel to `color`
`q.setGreen(v)`	change green channel value to `v`
`q.setRed(v)`	change red channel value to `v`

`Color`

method	meaning
`red, green, …, RED, GREEN, …`	*constant*: standard colors
`c = new Color(r,g,b)`	*constructor*: creates a new color object with specified `r`, `g` and `b` components
`c = new Color(value)`	*constructor*: creates a new color object with color `value (0-16,777,215)`

`ASCIIPrompter`

method	meaning
`d = new ASCIIPrompter()`	*constructor*: creates a prompter with default label
`d.close()`	closes prompter
`r = d.readDouble()`	waits for user to enter data and press `OK`, then reads data as a `double` and returns value
`i = d.readInt()`	waits for user to enter data and press `OK`, then reads data as an `int` and returns value
`s = d.readString()`	waits for user to enter data and press `OK`, then reads data as a `String` and returns value
`d.setLabel(label)`	sets the prompt label to `label`

`ASCIIDisplayer`

method	meaning
`d = new ASCIIDisplayer()`	*constructor*: displays window with a text area to display text
`d.close()`	waits for user to press `Close` and then closes displayer
`d.newLine()`	writes a line marker to the display so next output begins on next line
`d.writeDouble(r)`	writes the `double` value `r` to the display
`d.writeInt(i)`	writes the `int` value `i` to the display
`d.writeLine(s)`	writes the `String` value `s` to the display and positions to next line
`d.writeString(s)`	writes the `String` value `s` to the display

`ASCIIDataFile`

method	meaning
`f = new ASCIIDataFile()`	*constructor*: presents a file open dialog and opens text (data) file for input
`f.close()`	closes data file
`b = f.isEOF()`	returns `true` if last read failed because of EOF
`r = f.readDouble()`	reads next field as a `double` and returns value
`i = f.readInt()`	reads next field as an `int` and returns value
`s = f.readString()`	reads next field as a `String` and returns value

`ASCIIOutputFile`

method	meaning
`f = new ASCIIOutputFile()`	*constructor*: opens a new text output file presenting a File Save dialog.
`f.close()`	closes file
`f.newLine()`	writes a line marker to the file so next field begins on next line
`f.writeDouble(r)`	writes the `double` value `r` as a field
`f.writeInt(i)`	writes the `int` value `i` as a field
`f.writeString(s)`	writes the `String` value `s` as a field

`BinaryDataFile`

method	meaning
`f = new BinaryDataFile()`	*constructor*: presents a file open dialog and opens binary data file for input
`f.close()`	closes data file
`b = f.isEOF()`	returns `true` if last read failed because of EOF
`r = f.readDouble()`	reads next 8 bytes as a `double` and returns value
`i = f.readInt()`	reads next 4 bytes as an `int` and returns value
`o = f.readObject()`	reads some number of bytes as an `Object` and returns object reference
`s = f.readString()`	reads some number of bytes as a `String` and returns value

BinaryOutputFile

method	meaning
f = new BinaryOutputFile()	*constructor*: opens a new binary output file presenting a File Save dialog.
f.close()	closes file
f.writeDouble(*r*)	writes the double value *r* as 8 bytes
f.writeInt(*i*)	writes the int value *i* as 4 bytes
f.writeObject(*o*)	writes the Object *o* as a sequence of bytes
f.writeString(*s*)	writes the String value *s* as a sequence of bytes

BasicForm

method	meaning
f = new BasicForm()	*constructor*: creates a new form with one default button (OK) that will size to the layout of the fields added.
f = new BasicForm(*button*$_1$, *button*$_2$, …)	*constructor*: creates a new form with default buttons *button*$_1$, *button*$_2$… that will size to the layout of the fields added.
f.accept()	presents the default button(s) and awaits user pressing a button
i = *f*.accept()	presents the default button(s), awaits user pressing a button and returns button number (from 0) of button pressed
i = *f*.accept(*button*$_1$,*button*$_2$, …)	presents the specified *button*(s), awaits user pressing a button and returns button number (from 0) of button pressed
f.clear(*name*)	clears the field *name*
f.clearAll()	clears all fields in the form
f.close()	closes the form so that it cannot be used
f.placePicture(*name*,*picture*)	places *picture* onto the canvas *name*
f.placeTurtle(*name*,*turtle*)	places *turtle* onto the canvas *name*
r = *f*.readDouble(*name*)	reads and returns the field *name* as a double
i = *f*.readInt(*name*)	reads and returns the field *name* as an int
s = *f*.readString(*name*)	reads and returns the field *name* as a String
f.setEditable(*name*,*editable*)	makes field name *editable* (true) by user
f.setTitle(*title*)	sets title in title bar of window to *title*
f.writeDouble(*name*,*r*)	writes the double value *r* to the field *name*
f.writeInt(*name*,*i*)	writes the int value *i* to the field *name*
f.writeString(*name*,*s*)	writes the String value *s* to the field *name*

`BasicForm` Widgets

method	meaning
f.addCanvas(*name*,*label*,*width*, *height*,*x*,*y*)	adds a labeled canvas with *name*, *width*, and *height* at (*x*,*y*)
f.addCheckBox(*name*,*label*,*x*,*y*)	adds a labeled checkbox with *name* at (*x*,*y*)
f.addLabel(*name*,*label*,*x*,*y*)	adds a label with *name* at (*x*,*y*)
f.addRadioButtons(*name*,*label*, *vertical*,*x*,*y*,$button_1$,…)	adds labeled radio buttons with *name*, vertical or horizontal at (*x*,*y*) with button names $button_i$
f.addSlider(*name*,*label*,*min*,*max*, size,*x*,*y*)	adds a labeled slider with *name* and *size* over the range *min* to *max* at (*x*,*y*)
f.addSound(*name*,*label*,*x*,*y*)	adds a labeled sound play button with *name*, at (*x*,*y*)
f.addTextArea(*name*,*label*,*rows*, *columns*,*x*,*y*)	adds a labeled text area with *name*, *rows* and *columns* at (*x*,*y*)
f.addTextField(*name*,*label*, *format*,*width*,*x*,*y*)	adds a labeled text field with *name*, *format* and *width* at (*x*,*y*)

`ReportPrinter`

method	meaning
p = new ReportPrinter()	*constructor*: creates a new report in portrait orientation
p = new ReportPrinter(*portrait*)	*constructor*: creates a new report in portrait (`true`) or landscape (`false`) orientation
p.addField(*name*,*label*, *format*,*width*)	adds (L-R) a field to the report with *name*, *label* *format* and *width*
p.close()	closes report and prints it (print dialog)
p.newLine()	writes a line marker to the display so next output begins on next line
p.newPage()	forces next write to new page, printing header
p.setTitle($line_1$, $line_2$, …)	Sets title for page to specified *line*s
p.writeDouble(*name*,*r*)	writes the `double` value *d* to field *name*
p.writeInt(*name*,*i*)	writes the `int` value *i* to field *name*
p.writeLine(*s*)	writes the `String` value *s* starting at the next field and moves to next line
p.writeString(*name*,*s*)	writes the `String` value *s* to field *name*

`Formats`

method	meaning
f = getCurrencyInstance()	returns a format for currency (dollars)
f = getDateInstance()	returns a format for dates
f = getDateTimeInstance()	returns a format for date and time
f = getDecimalInstance(*n*)	returns a format for decimal number with *n* decimal places
f = getIntegerInstance()	returns a format for integer number
f = getPercentInstance()	returns a format for percentages
f = getTimeInstance()	returns a format for time