

The program will allow a user to input a number of curves and have them displayed on the screen. Our program is focused on elliptic curves (of the form  $y^2 = x^3 + ax + b$ ) and will allow the user to explore some of their interesting mathematical properties such as treating the curve as a field and doing “addition” of two points on it. The user will also be able to input other types of curves since it can be useful to compare them to elliptic curves and it is extra functionality that we can add relatively easily. The user will be able to input curves either by inputting a text file with a list of curves when they run the program or once the program is run they will have the opportunity to add curves by clicking a specific place on the screen and then typing the curve they would like to add. Ideally the user will also be able to pan and zoom, but this is not required for the program to function. The program will then respond by either plotting the curve if it is valid or giving an error message if not. The purpose this program will serve is to allow people to learn about and visually explore (mainly elliptic) curves.

Our current plans for classes (methods are non-recursive unless stated otherwise, everything is non-static unless stated otherwise):

**Main:** what the user runs and will then create instances of other classes to actually run the program. It will extend JPanel and implement `KeyListener` and `MouseListener`. Only one instance will exist.

**Member variables:** `int height`, `int width`, `int FPS`, `World world` all of which will be public. Height and width represent the width of the screen and world will have everything happening on the screen. These will be public so they can be utilized in the draw methods easily.

**Constructor:** we will only have one which takes no arguments since it will just set up the `KeyListener`, `MouseListener`, thread stuff, etc. and we only need one way to do this. We will be modifying code from previous homeworks for this.

**Methods:**

**`public static void main(String[] args)`** which will set up the JFrame and make a new instance of Main. This will also be code modified from previous homeworks.

**`public void paintComponent(Graphics g)`** which will reset the screen and then call world's draw method to draw everything we need to. This code will also be from previous hws.

**`public void keyPressed(KeyEvent e)`** which will handle user keyboard input to enter new curves and updates member variables in world accordingly.

**`public void mouseClicked(MouseEvent e)`** which will take action based on where the user clicks on the screen like clicking on an x next a curve to remove it. This also updates member variables in world accordingly. (None of us have used `MouseListener` before, so this is our best guess of the method we will use).

**`Array[int] parseEquation(String eq)`** which will take in a string and return the coefficients of the equations if it is a valid equation. If it is invalid it will return an array of length 1 (valid equations must have at least two coefficients) with the number corresponding to errors.

**Runner** - a nested class in main which implements `Runnable` and handles updating. (we will use the code from previous hws)

**Constructor:** only the default one, we don't have need for anything else

**Methods:**

**`Public void run()`** which updates the program each frame.

**World** - this will contain everything that's going to be plotted. Only one instance of world.

**Member variables:** public ArrayList<Curve> curves and public ArrayList<Point> points. These will be the list of points and curves that need to be plotted, respectively. It will also have public Pair center, and public int scale which will allow for panning and zooming if we get to implement that. These are public so that they can be easily updated in main.

**Constructors:** we will have two constructors. One will take in an ArrayList of curves (if the user inputted several in a file) and makes the curves member variable equal to that ArrayList. The other constructor will take no arguments and will just set the member variables to default values.

**Methods:**

**public void addCurve(Curve c)** adds a curve to the world's ArrayList. (Can have multiple of the same curve)

**public void addPoint(Point p)** adds a point to the world's ArrayList. If the point is already in the ArrayList it will just do nothing, but it shouldn't ever be called if that is the case.

**public void removeCurve(Curve c)** removes that curve from the ArrayList if it is in the ArrayList. It shouldn't ever be called if the curve isn't in the ArrayList, but we will use try and except just in case.

**public void removePoint(Point p)** same as above just for points.

**public void draw(Graphics g)** which will draw everything in both of the member ArrayLists, calling their draw methods.

**public void zoom(double x)** changes the scale of world by a given amount.

**public void pan(Pair newCenter)** changes the center of world to newCenter when the user pans. We are not yet sure how newCenter will be determined.

**Pair** - same class as from the hw. Just holds two doubles and lets us manipulate them. Any number can exist.

**Interface drawable** - contains a method for drawing, public void draw(Graphics G) which classes that implement it will define. It also contains a method public void centerAround() which centers the world around that instance.

**Curve implements drawable** - represents curves that will be plotting. 0 to yet to be defined upper limit can exist.

**Member variables:** private Array<int> coefficients which represent the coefficients of the equation in the curve. The first one will be the power of y, the second one will be the coefficient of  $x^0$ , the third will be for  $x^1$ , etc. It will have length at least 2, and likely a yet to be determined upper limit on length. This will be private since it doesn't need to be used by anything outside of the specific curve (if it is updated we would do that by deleting the curve and then making a new one with updated values). public Color color will represent the color that we are going to plot that curve in. It is public since we would like the user to be able to change the color so main

should be able to access it.

**Public void run()** which updates the program each frame.

**World** - this will contain everything that's going to be plotted. Only one instance of world.

**Member variables:** public ArrayList<Curve> curves and public ArrayList<Point> points. These will be the list of points and curves that need to be plotted, respectively. It will also have public Pair center, and public int scale which will allow for panning and zooming if we get to implement that. These are public so that they can be easily updated in main.

**Constructors:** we will have two constructors. One will take in an ArrayList of curves (if the user inputted several in a file) and makes the curves member variable equal to that ArrayList. The other constructor will take no arguments and will just set the member variables to default values.

**Methods:**

**public void addCurve(Curve c)** adds a curve to the world's ArrayList. (Can have multiple of the same curve)

**public void addPoint(Point p)** adds a point to the world's ArrayList. If the point is already in the ArrayList it will just do nothing, but it shouldn't ever be called if that is the case.

**public void removeCurve(Curve c)** removes that curve from the ArrayList if it is in the ArrayList. It shouldn't ever be called if the curve isn't in the ArrayList, but we will use try and except just in case.

**public void removePoint(Point p)** same as above just for points.

**public void draw(Graphics g)** which will draw everything in both of the member ArrayLists, calling their draw methods.

**public void zoom(double x)** changes the scale of world by a given amount.

**public void pan(Pair newCenter)** changes the center of world to newCenter when the user pans. We are not yet sure how newCenter will be determined.

**Pair** - same class as from the hw. Just holds two doubles and lets us manipulate them. Any number can exist.

**Interface drawable** - contains a method for drawing, public void draw(Graphics G) which classes that implement it will define. It also contains a method public void centerAround() which centers the world around that instance.

**Curve implements drawable** - represents curves that will be plotting. 0 to yet to be defined upper limit can exist.

**Member variables:** private Array[int] coefficients which represent the coefficients of the equation in the curve. The first one will be the power of y, the second one will be the coefficient of x<sup>0</sup>, the third will be for x<sup>1</sup>, etc. It will have length at least 2, and likely a yet to be determined upper limit on length. This will be private since it doesn't need to be used by anything outside of the specific curve (if it is updated we would do that by deleting the curve and then making a new one with updated values). public Color color will represent the color that we are going to plot that curve in. It is public since we would like the user to be able to change the color so main should be able to access it.

**Constructor:** we will have one constructor which will take in an Array[int] of coefficients that will be assigned to the member variable. Color will then automatically be assigned to the next color that has not yet been used. We always need the array of coefficients to define a curve and we never need to input anything else.

**Methods:**

**public void draw(Graphics G)** which plots the curve as well as puts its equation on the screen. Details of how it does this are yet to be determined. Maybe recursive.

**public boolean checkPoint(Point P, int distance)** checks if the point is within the given distance of the curve.

**public void centerAround()** which center the world around this curve and try to make the scale such that it shows the interesting parts of the curve (trying to get 0s, inflection points, etc. on screen).

**public pair compute(double x)** computes the y values for the corresponding x value (there will either be 1 or 2) if there is only 1 the second value in the pair will be the same as the first.

**EllipticCurve extends curve** - special type of curve with additional methods.

**Member variables:** no additional ones.


**Constructor:** same constructor as curve.

**Additional Methods:**

**public point addPoints(Point p1, Point p2)** which returns a point by "adding" two points on the elliptic curve. This will utilize findIntersection.


**public Array[point] findIntersection(curve line)** which will find intersections between the elliptic curve and the line. (We could maybe put this as just a method in the Curve class, but currently we don't plan on using it for other types of curves and that would make it harder to

Jamie




Lukas Luby-Prikot  
4:01 PM Nov 15

Jamie




Lukas Luby-Prikot  
4:02 PM Nov 15

Jamie




Lukas Luby-Prikot  
4:02 PM Nov 15

Jonah




Lukas Luby-Prikot  
4:02 PM Nov 15

Lukas




Lukas Luby-Prikot  
4:02 PM Nov 15

Lukas



Lukas Luby-Prikot  
4:02 PM Nov 15

Lukas



Lukas Luby-Prikot  
4:03 PM Nov 15

Lukas

**Constructor:** we will have one constructor which will take in an Array[int] of coefficients that will be assigned to the member variable. Color will then automatically be assigned to the next color that has not yet been used. We always need the array of coefficients to define a curve and we never need to input anything else.

### Methods:

**public void draw(Graphics G)** which plots the curve as well as puts its equation on the screen. Details of how it does this are yet to be determined. Maybe recursive.

**public boolean checkPoint(Point P, int distance)** checks if the point is within the given distance of the curve.

**public void centerAround()** which center the world around this curve and try to make the scale such that it shows the interesting parts of the curve (trying to get 0s, inflection points, etc. on screen).

**public pair compute(double x)** computes the y values for the corresponding x value (there will either be 1 or 2) if there is only 1 the second value in the pair will be the same as the first.

**EllipticCurve extends curve** - special type of curve with additional methods.

**Member variables:** no additional ones.

**Constructor:** same constructor as curve.

**Additional Methods:**

**public point addPoints(Point p1, Point p2)** which returns a point by “adding” two points on the elliptic curve. This will utilize findIntersection.

**public Array[point] findIntersection(curve line)** which will find intersections between the elliptic curve and the line. (We could maybe put this as just a method in the Curve class, but currently we don’t plan on using it for other types of curves and that would make it harder to implement.

If we have time we could add more additional methods for elliptic curves, but it is not clear what these would be yet or if we will get to that (we likely won’t).

**Point extends pair implements drawable**

**Member variables:** private static int radius which is the radius of the circle used when plotting points. Private since we don’t need to access it outside of the point class.

**Constructor:** same as for pair

**Additional Methods:**

**public void draw(Graphics g)** which plots the point as a circle with the radius defined in the member variable. It might also display the coordinates of the point if we decide we want to do that.

**public void centerAround()** which sets the center of the world equal to the coordinates of the point.

