Team A: GARP

Abby Weinreb, Eli Granade, Bryson Lee, Jack Silkaitis

Milestone 2: Iteration and Refactoring

04/06/2025

- [ ] List of completed features (M2)
    - [ ] Customizable fitness function → Eli
    - [ ] Tournament option for different populations → Abby
    - [ ] Possibly extra selection options - Jack
    - [ ] UI cleanup → Bryson
- [ ] Potential future features (M3/4)
    - [ ] Chromosome editor
    - [ ] Graph doesn't disappear on end
    - [ ] Restart button
    - [ ] Elitism works with all parameters

Jack -
Divergent change:

```
public void truncationSelection() {
    Collections.sort(curPop, new sortPop());

    ArrayList<Chromosome> toRemove = new ArrayList<Chromosome>();
    int count = 0;
    if (curPop.size() == numPop) {
        for (int i = 0; i < curPop.size(); i++) {
            if (count < (curPop.size() / 2)) {
                toRemove.add(curPop.get(i));
                count++;
            }
        }
    }

    for (int j = 0; j < toRemove.size(); j++) {
        curPop.remove(toRemove.get(j));

    }
}
```

```
public void rankSelection() {
    Collections.sort(curPop, new sortPop());
    double[] valuesArray = new double[curPop.size()];
    Chromosome[] secondArray = new Chromosome[curPop.size()];
    int total = 0;
    double rankPercentSum = 0;

    for (int i = 0; i < curPop.size() - 1; i++) {
        total += i;
    }

    for (int i = 0; i < curPop.size(); i++) {
        total += curPop.get(i).fitness;
        double percentage = (double) i / (double) total;
        rankPercentSum += percentage;
        valuesArray[i] = rankPercentSum;
        secondArray[i] = curPop.get(i);

    }

    curPop = new ArrayList<>();
    for (int i = 0; i <= 50; i++) {
        double chance = Math.random();
        int beforeSize = curPop.size();
        for (int j = 0; j < valuesArray.length; j++) {

            if (valuesArray[j] > chance) {

                curPop.add(secondArray[j]);
                break;
            }

        }
        if (curPop.size() == beforeSize) {
            curPop.add(secondArray[secondArray.length - 1]);
        }

    }
}
```

```
public void roulette() {
    Collections.sort(curPop, new sortPop());
    double[] valuesArray = new double[curPop.size()];
    Chromosome[] secondArray = new Chromosome[curPop.size()];
    int total = 0;
    double rankPercentSum = 0;

    for (int i = 0; i < curPop.size() - 1; i++) {
        total += curPop.get(i).fitness;
    }

    for (int i = 0; i < curPop.size(); i++) {
        total += curPop.get(i).fitness;
        double percentage = (double) i / (double) total;
        rankPercentSum += percentage;
        valuesArray[i] = rankPercentSum;
        secondArray[i] = curPop.get(i);

    }

    curPop = new ArrayList<>();
    for (int i = 0; i <= 50; i++) {
        double chance = Math.random();
        int beforeSize = curPop.size();
        for (int j = 0; j < valuesArray.length; j++) {

            if (valuesArray[j] > chance) {

                curPop.add(secondArray[j]);
                break;
            }

        }
        if (curPop.size() == beforeSize) {
            curPop.add(secondArray[secondArray.length - 1]);
        }

    }
}
```

After refactoring

```java
public void changeSelectionStrategy(SelectionType type)
{
    if(type == SelectionType.TRUNCATION)
    {
        this.selectionStrategy = new TruncationSelection(numPop);
    }
    else if(type == SelectionType.RANK)
    {
        this.selectionStrategy = new RankSelection();
    }
}

public void selection()
{
    curPop = selectionStrategy.select(curPop);
}
```

```java
public class TruncationSelection extends SelectionStrategy {

    private int numPop;

    public TruncationSelection(int numPop)
    {
        this.numPop = numPop;
    }

    @Override
    public ArrayList<Chromosome> select(ArrayList<Chromosome> curPop)
    {
        ArrayList<Chromosome> newPop = super.select(curPop);
        ArrayList<Chromosome> toRemove = new ArrayList<Chromosome>();
        int count = 0;
        if (newPop.size() == numPop) {
            for (int i = 0; i < newPop.size(); i++) {
                if (count < (newPop.size() / 2)) {
                    toRemove.add(newPop.get(i));
                    count++;
                }
            }
        }

        for (int j = 0; j < toRemove.size(); j++) {
            newPop.remove(toRemove.get(j));

        }

        return newPop;
    }
}
```

```java
public class RankSelection extends SelectionStrategy {


    @Override
    public ArrayList<Chromosome> select(ArrayList<Chromosome> curPop)
    {
        ArrayList<Chromosome> newPop = super.select(curPop);
        double[] valuesArray = new double[newPop.size()];
        Chromosome[] secondArray = new Chromosome[newPop.size()];
        int total = 0;
        double rankPercentSum = 0;

        for (int i = 0; i < newPop.size() - 1; i++) {
            total += i;
        }

        for (int i = 0; i < newPop.size(); i++) {
            total += newPop.get(i).fitness;
            double percentage = (double) i / (double) total;
            rankPercentSum += percentage;
            valuesArray[i] = rankPercentSum;
            secondArray[i] = newPop.get(i);

        }

        newPop = new ArrayList<>();
        for (int i = 0; i <= 50; i++) {
            double chance = Math.random();
            int beforeSize = newPop.size();
            for (int j = 0; j < valuesArray.length; j++) {

                if (valuesArray[j] > chance) {

                    curPop.add(secondArray[j]);
                    break;
                }
```

The selection types, previously done in separate methods within the EvolutionLoop class, now exist within separate classes with a strategy pattern separating them. This makes changing any one selection method, or the process in general, more focused.

```
public void runTruncation(boolean crossoverOption, int mutate) {
    if (crossoverOption) {
        System.out.println("truncate");
        evoLoop.changeSelectionStrategy(SelectionType.TRUNCATION);
        evoLoop.selection();
        evoLoop.flipMutation(mutate);
        this.highFit.add(evoLoop.returnHighestAverage());
        this.averageFit.add(evoLoop.returnAverage());
        this.lowFit.add(evoLoop.returnLowestAverage());
        this.fittest.add(evoLoop.returnFittest());
    }

    else {
        System.out.println("truncate");
        evoLoop.changeSelectionStrategy(SelectionType.TRUNCATION);
        evoLoop.selection();
        evoLoop.crossoverMutation(50);
        this.highFit.add(evoLoop.returnHighestAverage());
        this.averageFit.add(evoLoop.returnAverage());
        this.lowFit.add(evoLoop.returnLowestAverage());
        this.fittest.add(evoLoop.returnFittest());
    }
}

public void runRank(boolean crossoverOption, int mutate) {
    if (crossoverOption) {
        System.out.println("rank");
        evoLoop.changeSelectionStrategy(SelectionType.RANK);
        evoLoop.selection();
        evoLoop.crossoverMutation(50);
        this.highFit.add(evoLoop.returnHighestAverage());
        this.averageFit.add(evoLoop.returnAverage());
        this.lowFit.add(evoLoop.returnLowestAverage());
        this.fittest.add(evoLoop.returnFittest());
    }

    else {
        System.out.println("rank");
        evoLoop.changeSelectionStrategy(SelectionType.RANK);
        evoLoop.selection();
        evoLoop.flipMutation(mutate);
        this.highFit.add(evoLoop.returnHighestAverage());
```

Code duplication:

```
public void runRankElite(boolean crossoverOption, int mutate, int n) {
    if (crossoverOption) {
        System.out.println("rank");
        evoLoop.changeSelectionStrategy(SelectionType.RANK);
        evoLoop.selection();
        evoLoop.crossoverMutation(50);
        evoLoop.elitism(mutate, n);
        this.highFit.add(evoLoop.returnHighestAverage());
        this.averageFit.add(evoLoop.returnAverage());
        this.lowFit.add(evoLoop.returnLowestAverage());
        this.fittest.add(evoLoop.returnFittest());
    }

    else {
        System.out.println("rank");
        evoLoop.changeSelectionStrategy(SelectionType.RANK);
        evoLoop.selection();
        evoLoop.elitism(mutate, n);
        this.highFit.add(evoLoop.returnHighestAverage());
        this.averageFit.add(evoLoop.returnAverage());
        this.lowFit.add(evoLoop.returnLowestAverage());
        this.fittest.add(evoLoop.returnFittest());
    }

}

public void runRouletteElite(boolean crossoverOption, int mutate, int n) {
    if (crossoverOption) {
        System.out.println("roulette");
        evoLoop.changeSelectionStrategy(SelectionType.RANK);
        evoLoop.selection();
        evoLoop.crossoverMutation(50);
        evoLoop.elitism(mutate, n);
        this.highFit.add(evoLoop.returnHighestAverage());
        this.averageFit.add(evoLoop.returnAverage());
        this.lowFit.add(evoLoop.returnLowestAverage());
        this.fittest.add(evoLoop.returnFittest());
    }
```

```
if (selectionType.equals("Truncation")) {
    if (evolveType.equals("Regular")) {
        for (int i = 0; i < numGenerationsVal; i++) {
            newComponent.runTruncation(crossoverOption.isSelected(), mutationRateVal);
        }

    }
    if (evolveType.equals("Elitism")) {
        for (int i = 0; i < numGenerationsVal; i++) {
            newComponent.runTruncationElite(crossoverOption.isSelected(), mutationRateVal,
                    elitismNum);
        }
    }
}

if (selectionType.equals("Roulette")) {
    if (evolveType.equals("Regular")) {
        for (int i = 0; i < numGenerationsVal; i++) {
            newComponent.runRoulette(crossoverOption.isSelected(), mutationRateVal);
        }

    }
    if (evolveType.equals("Elitism")) {
        for (int i = 0; i < numGenerationsVal; i++) {
            newComponent.runRouletteElite(crossoverOption.isSelected(), mutationRateVal,
                    elitismNum);
        }
```

After refactoring

```
public void run(boolean crossoverOption, int mutate, int numElites, SelectionType selectionStrategy)
{
    evoLoop.changeSelectionStrategy(selectionStrategy);
    evoLoop.selection();

    if (crossoverOption)
    {
        evoLoop.flipMutation(mutate);
    }
    else if(numElites == 0)
    {
        evoLoop.crossoverMutation(50);
    }

    if(numElites != 0)
    {
        evoLoop.elitism(mutate, numElites);
    }
    this.highFit.add(evoLoop.returnHighestAverage());
    this.averageFit.add(evoLoop.returnAverage());
    this.lowFit.add(evoLoop.returnLowestAverage());
    this.fittest.add(evoLoop.returnFittest());
}
```

```
SelectionType selectionStrategy = SelectionType.TRUNCATION; //Default value
int numElites = 0;

if (selectionType.equals("Truncation")) {
    selectionStrategy = SelectionType.TRUNCATION;
}
else if (selectionType.equals("Roulette")) {
    selectionStrategy = SelectionType.ROULETTE;
}
else if (selectionType.equals("Rank")) {
    selectionStrategy = SelectionType.RANK;
}
else if(selectionType.equals("Tournament"))
{
    selectionStrategy = SelectionType.TOURNAMENT;
}

if(evolveType.equals("Elitism"))
{
    numElites = elitismNum;
}

for (int i = 0; i < numGenerationsVal; i++) {
    newComponent.run(crossoverOption.isSelected(), mutationRateVal, numElites, selectionStrategy);
}
```

The vast amounts of code duplication found within the process for running a specified selection type have been removed and replaced with only a few methods and simple processes that maintain the same functionality.

Speculative generality:

```
public class EvolutionLoop {

    private Population population;
    private ArrayList<Chromosome> updatedSet;
    private ArrayList<Chromosome> curPop;
    private int genomeLengthVal, mutationRateVal, numPop;

    public EvolutionLoop(int numPop, int genomeSize) {
        this.population = new Population(numPop, genomeSize);
        this.updatedSet = new ArrayList<Chromosome>();
        this.curPop = new ArrayList<Chromosome>();
        this.genomeLengthVal = genomeSize;
        this.numPop = population.returnPopSize();

    }
```

```java
public int calculateBorderFitness() {
    int count = 0;
    for (int i = 0; i < this.numberOfArrays; i++) {
        for (int j = 0; j < this.numberOfGenesInArray; j++) {
            if (i == 0 || j == 0 || i == this.numberOfArrays - 1 || j == this.numberOfGenesInArray - 1) {
                count += this.geneticCode[i][j];
            }
        }
    }
    return count;
}

/**
 * Quantifies the difference between two Chromosome's genetic code.
 *
 * @author akamahwa
 *
 */
public int calculateHammingDistance(Chromosome otherChromosome) {
    int hammingDistance = 0;
    for (int j = 0; j < this.numberOfArrays; j++) {
        for (int k = 0; k < this.numberOfGenesInArray; k++) {
            if (this.geneticCode[j][k] != otherChromosome.geneticCode[j][k]) {
                hammingDistance++;
            }
        }
    }
    return hammingDistance;
}
```

```java
/**
 * Quantifies the similarity between two Chromosome's genetic code.
 *
 * @author akamahwa
 *
 */
int calculateInverseHammingDistance(Chromosome otherChromosome) {
    int inverseHammingDistance = 0;
    for (int j = 0; j < this.numberOfArrays; j++) {
        for (int k = 0; k < this.numberOfGenesInArray; k++) {
            if (this.geneticCode[j][k] == otherChromosome.geneticCode[j][k]) {
                inverseHammingDistance++;
            }
        }
    }
    return inverseHammingDistance;
}

public int calculateFitnessMatchOtherChromosome(Chromosome perfectSpecimen) {
    return 100 * this.calculateHammingDistance(perfectSpecimen) / this.numberOfGenes();
}
```

After refactoring

```java
public class EvolutionLoop {

    private Population population;
    private ArrayList<Chromosome> updatedSet;
    private ArrayList<Chromosome> curPop;
    private int numPop;

    public EvolutionLoop(int numPop, int genomeSize) {
        this.population = new Population(numPop, genomeSize);
        this.updatedSet = new ArrayList<Chromosome>();
        this.curPop = new ArrayList<Chromosome>();
        this.numPop = population.returnPopSize();

    }
```

```java
// Calculates fitness based on how many consecutive bits are 1's
// ex: 100110110111 has a fitness of 7
public int calculateInARowFitness() {
    int count = 1;
    int curCount = 0;
    String rawGene = this.asStringRaw();
    int i = 0;
    while (i < rawGene.length() - 1) {
        if (rawGene.charAt(i) == '1' && rawGene.charAt(i + 1) == '1') {
            count++;
        }
        i++;
    }
    if (rawGene.charAt(rawGene.length() - 2) == '1' && rawGene.charAt(rawGene.length() - 1) == '1') {
        count++;
    }

    if (count == 1) {
        return 0;
    }

    this.fitness = count;
    return count;

}

private boolean checkIfPerfectSquare(int n) {
    int roundedSquareRoot = (int) Math.sqrt(n);
    return Math.pow(roundedSquareRoot, 2) == n;
}
```

The fields genomeLengthVal and mutationRateVal were never used for any computation, and so were safely deleted from the code to remove speculative generality. The same goes for the functions calculateBorderFitness(), calculateHammingDistance(), calculateInverseHammingDistance(), and calculateFitnessMatchOtherChromosome() in the Chromosome class.

These refactorings made implementing the relevant feature, adding more selection types, much easier than it would have been in the initial codebase.

This refactoring does introduce a small switch case to determine which selectionType is used during a run, due to how the GUI interfaces with the backend, but due to the mass volume of code that was refactored through this change and the better form of the Strategy Pattern that it enables, this was deemed acceptable.

The tests developed for this milestone can be found under testing/SelectionTesting, and they test valid combinations of all present selection types.

Bryson:

1. Temporary Field:

```
public ChromosomeViewer() throws FileNotFoundException {

public ChromosomeViewer(Chromosome chromosome, double mutationRate) {

public ChromosomeViewer(Chromosome chromosome) {
```

After refactoring:

```
public ChromosomeViewer(Chromosome inputChromosome, double mutationRate) {
        Chromosome chromosome;
        if(inputChromosome == null){
                chromosome = getTypedChromosome();
        }
        else{
                chromosome = inputChromosome;
        }
```

ChromosomeViewer has three separate constructors that are only used situationally. Each constructor shares similar functionality but also has slightly different logic. These were refactored into a single constructor that used both arguments, which was needed with my refactor to include a default mutationRate. The mutationRate is required, but was not previously passed in all constructors and the inputChromosome is called internally with null when it is not needed to prevent the need for another constructor.

2. Primitive Obsession

```
genomeLengthVal = Integer.parseInt(genomeLength.getText());
numGenerationsVal = Integer.parseInt(numGenerations.getText());
populationVal = Integer.parseInt(population.getText());
mutationRateVal = Integer.parseInt(mutationRate.getText());
selectionType = cb.getSelectedItem().toString();
evolveType = cb2.getSelectedItem().toString();
elitismNum = Integer.parseInt(elitismNumButton.getText());
newComponent.genSize = numGenerationsVal;
newComponent.startUp(populationVal, genomeLengthVal);
```

After refactoring:

```
private void setDefaults(HashMap<JComponent, String> componentToText) {
        for (JComponent component : componentToText.keySet()) {
                if (component instanceof JTextField) {
                        String text = componentToText.get(component);
                        JTextField textField = (JTextField) component;
                        if(!textField.getText().equals(text) && !textField.getText().equals("")) {
                                componentToDefaultVal.put(textField, Integer.valueOf(textField.getText()));
                        }
                }
        }
}
```

EvolutionViewer used a method of manually parsing and setting values. This prevented the implementation of my feature because it was inflexible when trying to check what the default value of a button should be. This was demonstrated by an obsession with using raw magic Strings and raw buttons instead of an object or data structure to link them. When implementing my feature (having buttons have default values), it was necessary to know what the default button string was (which was hardcoded at the time). Thus, I refactored them into another class, named “EvoViewerSwingComponents” and mapped a String of the default value with the button. Using this HashMap, we can iterate over each element instead of calling each element one by one.

3. Data Clumps

```
JTextField mutationRate = new JTextField("Enter Mutation Rate", 0);
JTextField numGenerations = new JTextField("Enter Number of Generations", 0);
JTextField population = new JTextField("Enter Population", 0);
JTextField genomeLength = new JTextField("Enter Genome Length", 0);

JButton enterButton = new JButton("Start");
JTextField elitismNumButton = new JTextField("Number of Elites", 0);
JCheckBox terminateAtMaxButton = new JCheckBox("Terminate at Max Fitness?");
JCheckBox crossoverOption = new JCheckBox("Crossover?");

JButton seeFitChrom = new JButton("Show Fittest Chromosome");

String[] selectionChoices = { "Truncation", "Roulette", "Rank" };

final JComboBox<String> cb = new JComboBox<String>(selectionChoices);

String[] evolveChoices = { "Regular", "Elitism" };

final JComboBox<String> cb2 = new JComboBox<String>(evolveChoices);

// Add things to frame
inputPanel.add(enterButton);
inputPanel.add(genomeLength);
inputPanel.add(population);
inputPanel.add(numGenerations);
inputPanel.add(mutationRate);
inputPanel.add(elitismNumButton);
inputPanel.add(cb);
inputPanel.add(cb2);
inputPanel.add(terminateAtMaxButton);
inputPanel.add(crossoverOption);
inputPanel.add(seeFitChrom);

frame.add(inputPanel, BorderLayout.SOUTH);
frame.add(newComponent, BorderLayout.CENTER);
```

```
genomeLengthVal = Integer.parseInt(genomeLength.getText());
numGenerationsVal = Integer.parseInt(numGenerations.getText());
populationVal = Integer.parseInt(population.getText());
mutationRateVal = Integer.parseInt(mutationRate.getText());
selectionType = cb.getSelectedItem().toString();
evolveType = cb2.getSelectedItem().toString();
elitismNum = Integer.parseInt(elitismNumButton.getText());
newComponent.genSize = numGenerationsVal;
newComponent.startUp(populationVal, genomeLengthVal);
```

After refactoring:

```
private void removeTextForComponents(HashMap<JComponent, String> componentToText) {
    for (JComponent component : componentToText.keySet()) {
        if (component instanceof JTextField) {
            JTextField textField = (JTextField) component;
            removeTextOnClick(textField);
        }
    }
}

private void centerAlignTextFields(HashMap<JComponent, String> componentToText) {
    for (JComponent component : componentToText.keySet()) {
        if (component instanceof JTextField) {
            JTextField textField = (JTextField) component;
            textField.setHorizontalAlignment(JTextField.CENTER);
        }
    }
}

private void addComponentsToPanel(HashMap<JComponent, String> componentToText, JPanel inputPanel) {
    inputPanel.add(enterButton);
            inputPanel.add(genomeLengthButton);
            inputPanel.add(populationButton);
            inputPanel.add(numGenerationsButton);
            inputPanel.add(mutationRateButton);
            inputPanel.add(numEliteIndivButton);
            inputPanel.add(selectionBox);
            inputPanel.add(fitnessBox);
            inputPanel.add(evolveTypeBox);
            inputPanel.add(terminateAtMaxButton);
            inputPanel.add(crossoverButton);
    inputPanel.add(evolveTypeBox);
            inputPanel.add(showFittestButton);
}
```

```
private void initializeComponentToText(){
    componentToText.put(enterButton, enterText);
    componentToText.put(genomeLengthButton, genomeLengthText);
    componentToText.put(populationButton, populationText);
    componentToText.put(numGenerationsButton, numGenerationsText);
    componentToText.put(mutationRateButton, mutationRateText);
    componentToText.put(numEliteIndivButton, numEliteIndiv);
    componentToText.put(selectionBox, "Selection Method");
    componentToText.put(fitnessBox, "Fitness Function");
    componentToText.put(evolveTypeBox, "Evolution Type");
    componentToText.put(terminateAtMaxButton, terminateAtMaxText);
    componentToText.put(crossoverButton, crossoverText);
    componentToText.put(showFittestButton, showFittestText);
}
```

```
private void setDefaults(HashMap<JComponent, String> componentToText) {
        for (JComponent component : componentToText.keySet()) {
                if (component instanceof JTextField) {
                        String text = componentToText.get(component);
                        JTextField textField = (JTextField) component;
                        if(!textField.getText().equals(text) && !textField.getText().equals("")) {
                                componentToDefaultVal.put(textField, Integer.valueOf(textField.getText()));
                        }
                }
        }
}
```

Java components seemed to be clumped and required the entire group to be copy pasted for each new addition for my feature. These clumps of data prevent elegant designs that iterate over all buttons, forcing the entire clump to be copy-pasted. To fix it, I created a HashMap of the buttons and refactored them into a separate object, making it easy to add a new one or work with operations that use all buttons. Since inputPanels are ordered and HashMaps are not, in order to get the display we want, the inputPanel required manual ordering, however, functionality like center aligning and removing text was able to be iteratively done over the HashMap containing JComponents.

Abby

1. God Class

```java
/**
 * @author Nnesachia Akanah & Abbx Walorsh
 *
 */

public class Chromosome {

    public static final Color PHENOTYPE_OF_1_GENE = new Color(37, 244, 137);
    public static final Color PHENOTYPE_OF_0_GENE = new Color(242, 235, 226);
    public static final int PHENOTYPE_SIDE_LENGTH_1 = 50;
    public static final int PHENOTYPE_SIDE_LENGTH_2 = 5;
    public int fitness;

    private int[][] geneticCode;
    private int numberOfArrays, numberOfGenesInArray;

    public Chromosome() {
        Random random = new Random();

        this.numberOfArrays = 10;
        this.numberOfGenesInArray = 10;
        int[][] randomGeneticCode = new int[numberOfArrays][numberOfGenesInArray];
        for (int j = 0; j < numberOfArrays; j++) {
            for (int k = 0; k < numberOfGenesInArray; k++) {
                randomGeneticCode[j][k] = random.nextInt(2);
            }
        }
        this.geneticCode = randomGeneticCode;
        this.fitness = this.calculateInARowFitness();
        System.out.println(this.fitness);
    }

    public Chromosome(int genomeSize) {
        if (genomeSize == 20) {
            Random random = new Random();
            this.numberOfArrays = 4;
            this.numberOfGenesInArray = 5;
            int[][] randomGeneticCode = new int[this.numberOfArrays][this.numberOfGenesInArray];
            for (int j = 0; j < this.numberOfArrays; j++) {
                for (int k = 0; k < this.numberOfGenesInArray; k++) {
                    randomGeneticCode[j][k] = random.nextInt(2);
                }
            }
            this.geneticCode = randomGeneticCode;
        }

        else if (checkIfPerfectSquare(genomeSize)) {
            Random random = new Random();
            this.numberOfArrays = (int) (Math.sqrt(genomeSize));
            this.numberOfGenesInArray = this.numberOfArrays;
            int[][] randomGeneticCode = new int[this.numberOfArrays][this.numberOfGenesInArray];
            for (int j = 0; j < this.numberOfArrays; j++) {
                for (int k = 0; k < this.numberOfGenesInArray; k++) {
                    if (random.nextBoolean()) {
                        randomGeneticCode[j][k] = 1;
                    } else {
                        randomGeneticCode[j][k] = 0;
                    }

                }
            }
            this.geneticCode = randomGeneticCode;
        }

        else {
            throw new IllegalArgumentException();
        }

        this.fitness = this.calculateFitnessV1();
        System.out.println();
    }

    public Chromosome(int[][] geneticCode) {
        this.geneticCode = new int[geneticCode.length][geneticCode[0].length];
        this.numberOfArrays = this.geneticCode.length;
        this.numberOfGenesInArray = this.geneticCode[0].length;
        for (int j = 0; j < this.numberOfArrays; j++) {
            for (int k = 0; k < this.numberOfGenesInArray; k++) {
                this.geneticCode[j][k] = geneticCode[j][k];
            }
        }
        this.fitness = this.calculateFitnessV1();
    }

    public int[][] geneticCode() {
        return this.geneticCode;
    }

    public int numberOfArrays() {
        return this.numberOfArrays;
    }

    public int numberOfGenesInArray() {
        return this.numberOfGenesInArray;
    }

    public int numberOfGenes() {
        return this.numberOfArrays * numberOfGenesInArray;
    }

    public void setGeneticCode(int[][] geneticCode) {
        this.geneticCode = geneticCode;
    }
```

```
 */
public String asString() {
    String geneticCodeString = "";
    for (int[] array : this.geneticCode) {
        for (int gene : array) {
            geneticCodeString += gene + " ";
        }
        geneticCodeString += "\n              ";
    }
    return "Genetic Code: " + geneticCodeString;
}

public String asStringRaw() {
    String geneticCodeString = "";
    for (int[] array : this.geneticCode) {
        for (int gene : array) {
            geneticCodeString += gene;
        }
    }
    return geneticCodeString;
}

/**
 * Treats a Chromosome's genetic code as a binary number and converts it into a
 * decimal number.
 *
 * @author akamahwa
 *
 */
public long getDecimal() {
    long decimalRep = 0;
    for (int j = 0; j < this.numberOfArrays; j++) {
        for (int k = 0; k < this.numberOfGenesInArray; k++) {
            decimalRep += this.geneticCode[j][k] * Math.pow(2, j * this.numberOfArrays + k);
        }
    }
    return decimalRep;
}

public void drawOn(int option, JPanel panel) {
    if (option == 1) {
        for (int j = 0; j < this.geneticCode.length; j++) {
            for (int k = 0; k < this.geneticCode[0].length; k++) {
                if (this.geneticCode[j][k] == 0) {
                    JButton geneButton = new JButton();
                    geneButton.setPreferredSize(new Dimension(PHENOTYPE_SIDE_LENGTH_1, PHENOTYPE_SIDE_LENGTH_1));
                    geneButton.setBackground(PHENOTYPE_OF_0_GENE);
                    geneButton.addActionListener(
                            new GeneButtonListener(geneButton, PHENOTYPE_OF_0_GENE, PHENOTYPE_OF_1_GENE));
                    panel.add(geneButton);
                }

                else {
                    JButton geneButton = new JButton();
                    geneButton.setPreferredSize(new Dimension(PHENOTYPE_SIDE_LENGTH_1, PHENOTYPE_SIDE_LENGTH_1));
                    geneButton.setBackground(PHENOTYPE_OF_1_GENE);
                    geneButton.addActionListener(
                            new GeneButtonListener(geneButton, PHENOTYPE_OF_0_GENE, PHENOTYPE_OF_1_GENE));
                    panel.add(geneButton);
                }
            }
        }
    }

    else {
        for (int j = 0; j < this.geneticCode.length; j++) {
            for (int k = 0; k < this.geneticCode[0].length; k++) {
                if (this.geneticCode[j][k] == 0) {
                    JButton geneButton = new JButton();
                    geneButton.setPreferredSize(new Dimension(PHENOTYPE_SIDE_LENGTH_2, PHENOTYPE_SIDE_LENGTH_2));
                    geneButton.setBackground(PHENOTYPE_OF_0_GENE);
                    geneButton.addActionListener(
                            new GeneButtonListener(geneButton, PHENOTYPE_OF_0_GENE, PHENOTYPE_OF_1_GENE));
                    panel.add(geneButton);
                }

                else {
                    JButton geneButton = new JButton();
                    geneButton.setPreferredSize(new Dimension(PHENOTYPE_SIDE_LENGTH_2, PHENOTYPE_SIDE_LENGTH_2));
                    geneButton.setBackground(PHENOTYPE_OF_1_GENE);
                    geneButton.addActionListener(
                            new GeneButtonListener(geneButton, PHENOTYPE_OF_0_GENE, PHENOTYPE_OF_1_GENE));
                    panel.add(geneButton);
                }
            }
        }
    }

}

public Chromosome mutatedOffspring(int n) {
    Chromosome offspring = new Chromosome(this.geneticCode);
    for (int k = 0; k < n; k++) {
        Random random = new Random();
        int randomArray = random.nextInt(offspring.numberOfArrays);
        int randomGeneInArray = random.nextInt(offspring.numberOfGenesInArray);
        if (offspring.geneticCode()[randomArray][randomGeneInArray] == 0) {
            offspring.geneticCode()[randomArray][randomGeneInArray] = 1;
        } else {
            offspring.geneticCode()[randomArray][randomGeneInArray] = 0;
        }
```

```java
package mainApp;

import java.awt.BasicStroke;

@SuppressWarnings("serial")
public class EvolutionComponent extends JComponent {

    public static final int LINE_WIDTH = 2;
    public static final int STATS_HEIGHT = 200;
    public static final int SIDE_OFFSET = 100;
    private EvolutionLoop evoLoop;
    public int genSize;
    public int i;
    public Chromosome fittestChrom;

    // Lists for three values to be graphed
    public ArrayList<Integer> highFit;
    public ArrayList<Integer> averageFit;
    public ArrayList<Integer> lowFit;
    public ArrayList<Chromosome> fittest;

//  public EvolutionComponent1(EvolutionLoop evoLoop, int numPop) {
//      this.setPreferredSize(new Dimension(EvolutionViewer.FRAME_WIDTH, STATS_HEIGHT));
//      //this.evoLoop = new EvolutionLoop(numPop);
//      this.lowFit = new ArrayList<Integer>();
//      this.averageFit = new ArrayList<Integer>();
//      this.highFit = new ArrayList<Integer>();
//      this.updated = new HashMap<Integer, Integer>();
//      //this.image = ImageIO.read(new File("darwin.jpg"));
//  }

    public EvolutionComponent(EvolutionLoop evoLoop2, int populationVal) {
        // TODO Auto-generated constructor stub
        this.setPreferredSize(new Dimension(EvolutionViewer.FRAME_WIDTH, STATS_HEIGHT));
        // this.evoLoop = new EvolutionLoop(numPop);
        this.lowFit = new ArrayList<Integer>();
        this.averageFit = new ArrayList<Integer>();
        this.highFit = new ArrayList<Integer>();
        this.fittest = new ArrayList<Chromosome>();
        this.i = 0;
    }

    public void setUpLoop(int numPop, int genomeSize) {
        this.evoLoop = new EvolutionLoop(numPop, genomeSize);

    }

    // Adds values from evolution loop
    public void startUp(int numPop, int genomeSize) {
        this.evoLoop = new EvolutionLoop(numPop, genomeSize);
        evoLoop.createPop();

    }

    public void runTruncation(boolean crossoverOption, int mutate) {
        if (crossoverOption) {
            System.out.println("truncate");
            evoLoop.truncationSelection();
            evoLoop.flipMutation(mutate);
            this.highFit.add(evoLoop.returnHighestAverage());
            this.averageFit.add(evoLoop.returnAverage());
            this.lowFit.add(evoLoop.returnLowestAverage());
            this.fittest.add(evoLoop.returnFittest());
        }

        else {
            System.out.println("truncate");
            evoLoop.truncationSelection();
            evoLoop.crossoverMutation(50);
            this.highFit.add(evoLoop.returnHighestAverage());
            this.averageFit.add(evoLoop.returnAverage());
            this.lowFit.add(evoLoop.returnLowestAverage());
            this.fittest.add(evoLoop.returnFittest());
        }
    }

    public void runRank(boolean crossoverOption, int mutate) {
        if (crossoverOption) {
            System.out.println("rank");
            evoLoop.rankSelection();
            evoLoop.crossoverMutation(50);
            this.highFit.add(evoLoop.returnHighestAverage());
            this.averageFit.add(evoLoop.returnAverage());
            this.lowFit.add(evoLoop.returnLowestAverage());
            this.fittest.add(evoLoop.returnFittest());
        }

        else {
            System.out.println("rank");
            evoLoop.rankSelection();
            evoLoop.flipMutation(mutate);
            this.highFit.add(evoLoop.returnHighestAverage());
            this.averageFit.add(evoLoop.returnAverage());
            this.lowFit.add(evoLoop.returnLowestAverage());
            this.fittest.add(evoLoop.returnFittest());
        }
    }
```

We had two God classes, Chromosome and EvolutionComponent. Chromosome created, mutated, and calculated the fitness of the Chromosome. So this was split into Chromosome and ChromosomeOperations. There were also test cases made for ChromosomeGenerator to make sure it had the correct behavior. This made the classes smaller and reduced how much functionality each class had to have. For EvolutionComponent, it took a loop and stored the data from that iteration as well as drawing the graph and fitness lines (see Long Method). It wasn't appropriate for the drawing functionality to be in this class so it was extracted to FitnessGraphPanel. This left EvolutionComponent to perform its data storing and sorting responsibilities.

After refactoring

```java
public class Chromosome {

    private int[][] geneticCode;
    private static int numberOfArrays;
    private static int numberOfGenesInArray;
    private ChromosomeOperations cO = new ChromosomeOperations();

    private static final Random RANDOM = new Random();

    public Chromosome(int genomeSize) {
        initializeChromosome(genomeSize);


    }

    public Chromosome(int[][] geneticCode) {
        initializeCustomChromosome(geneticCode);


    }

    private void initializeCustomChromosome(int[][] geneticCode) {
        this.geneticCode = geneticCode;
        this.numberOfArrays = geneticCode.length;
        this.numberOfGenesInArray = geneticCode[0].length;
    }

    private void initializeChromosome(int genomeSize) {
        if (checkIfPerfectSquare(genomeSize)) {
            Chromosome.numberOfArrays = (int) (Math.sqrt(genomeSize));
            Chromosome.numberOfGenesInArray = Chromosome.numberOfArrays;
            this.geneticCode = generateGeneticCode();
        }
        else {
            throw new IllegalArgumentException();
        }
    }

    private int[][] generateGeneticCode() {
        int[][] randomGeneticCode = new int[numberOfArrays][numberOfGenesInArray];
        for (int j = 0; j < numberOfArrays; j++) {
            for (int k = 0; k < numberOfGenesInArray; k++) {
                randomGeneticCode[j][k] = RANDOM.nextInt(2);
            }
        }
        return randomGeneticCode;
    }


    public int[][] geneticCode() {
        return this.geneticCode;
    }

    public int numberOfArrays() {
        return Chromosome.numberOfArrays;
    }

    public int numberOfGenesInArray() {
        return Chromosome.numberOfGenesInArray;
    }

    public int numberOfGenes() {
        return Chromosome.numberOfArrays * numberOfGenesInArray;
    }

    public void setGeneticCode(int[][] geneticCode) {
        this.geneticCode = geneticCode;
    }


    public String asString() {

    public String asStringRaw() {


    public long getDecimal() {




    private boolean checkIfPerfectSquare(int n) {

    public void mutate3(int n) {
        cO.mutate3(this, n);
    }

}
```

```java
public ChromosomeOperations() {


}



public Chromosome mutatedOffspring(Chromosome c, int n) {
    Chromosome offspring = new Chromosome(c.geneticCode());
    for (int k = 0; k < n; k++) {
        Random random = new Random();
        int randomArray = random.nextInt(offspring.numberOfArrays());
        int randomGeneInArray = random.nextInt(offspring.numberOfGenesInArray());
        if (offspring.geneticCode()[randomArray][randomGeneInArray] == 0) {
            offspring.geneticCode()[randomArray][randomGeneInArray] = 1;
        } else {
            offspring.geneticCode()[randomArray][randomGeneInArray] = 0;
        }
    }

    return offspring;
}

public ArrayList<Chromosome> crossoverWith(Chromosome mate1, Chromosome mate2, int crossoverPoint) {
    // Assumption: Parents have the same code size.
    ArrayList<Chromosome> offspring = new ArrayList<>();
    int[][] firstbornCode = new int[mate1.numberOfArrays()][mate1.numberOfGenesInArray()];
    int[][] secondbornCode = new int[mate1.numberOfArrays()][mate1.numberOfGenesInArray()];
    for (int j = 0; j < firstbornCode.length; j++) {
        for (int k = 0; k < firstbornCode[0].length; k++) {
            if (firstbornCode.length * j + k < crossoverPoint) {
                firstbornCode[j][k] = mate1.geneticCode()[j][k];
                secondbornCode[j][k] = mate2.geneticCode()[j][k];
            }

            else {
                firstbornCode[j][k] = mate2.geneticCode()[j][k];
                secondbornCode[j][k] = mate1.geneticCode()[j][k];
            }
        }
    }

    offspring.add(new Chromosome(firstbornCode));
    offspring.add(new Chromosome(secondbornCode));
    return offspring;
}

public int[][] mutate3(Chromosome c, int n) {
    Random random = new Random();
    int[][] newGeneCode = new int[c.geneticCode().length][c.geneticCode()[0].length];

    int rand = 0;
    for (int i = 0; i < c.geneticCode().length; i++) {
        for (int j = 0; j < c.geneticCode()[0].length; j++) {
            newGeneCode[i][j] = c.geneticCode()[i][j];
        }
    }

    for (int i = 0; i < newGeneCode.length; i++) {
        for (int j = 0; j < newGeneCode[0].length; j++) {
            rand = random.nextInt(100);
            if (rand < n) {
                if (newGeneCode[i][j] == 0) {
                    newGeneCode[i][j] = 1;
                }

                else if (newGeneCode[i][j] == 1) {
                    newGeneCode[i][j] = 0;
                }
            }
        }

    }
    // System.out.println(newGeneCode.toString());
    return newGeneCode;
}
```

2. Long Method

```java
@Override
protected void paintComponent(Graphics g) {

    Font font = new Font(null, Font.PLAIN, 10);
    super.paintComponent(g);
    Graphics2D g2 = (Graphics2D) g;
    g2.setFont(font);
    int offsetX = 100;
    int offsetY = this.getHeight() - 100;
    int tickSize = 3;
    g2.setColor(Color.LIGHT_GRAY);
    g2.drawRect(0, 0, this.getWidth(), this.getHeight());
    int textOffsetX = 15;
    int scale = 2;
    int textOffsetY = 5;
    HashMap<Integer, Integer> pointsHigh = new HashMap<Integer, Integer>();
    HashMap<Integer, Integer> pointsAve = new HashMap<Integer, Integer>();
    HashMap<Integer, Integer> pointsLow = new HashMap<Integer, Integer>();

    //// Draws X-axis and tics
    g2.setColor(Color.BLACK);
    g2.drawLine(offsetX, offsetY, this.getWidth() - offsetX, offsetY);
    g2.drawLine(offsetX, offsetY, offsetX, this.getHeight() - offsetY);
    for (int i = 20 + offsetX; i < this.getWidth() - offsetX; i += 20) {
        g2.drawLine(i, offsetY + tickSize, i, offsetY - tickSize);
        if ((i - offsetX) % 40 == 0) {
            g2.drawString(String.valueOf((i - offsetX) / scale), i - 10, offsetY + textOffsetX);
        }

    }

    // Draws Y-axis and tics
    for (int i = offsetY - 20; i > offsetX; i -= 20) {
        g2.drawLine(offsetX + tickSize, i, offsetX - tickSize, i);
    }
    for (int i = 40; i < 360; i += 40) {
        g2.drawString(String.valueOf(i), offsetX - textOffsetX * 2, offsetY - i + textOffsetY);
    }

    // Adds text to graph
    AffineTransform affineTransform = new AffineTransform();
    affineTransform.rotate(Math.toRadians(-90), 0, 0);
    Font font2 = new Font(null, Font.PLAIN, 20);
    Font rotatedFont = font2.deriveFont(affineTransform);
    g2.setFont(rotatedFont);
    g2.drawString("Fitness", offsetX - textOffsetX * 3, this.getHeight() / 2);
    g2.setFont(font2);
    g2.drawString("Number of Generations", this.getWidth() / 2 - offsetX, offsetY + textOffsetY * 9);

    if (highFit.size() > 0 && i < highFit.size()) {
        // Labels for values being graphed
//          g2.setColor(Color.GREEN);
//          g2.drawString("Lowest: " + lowFit.get(highFit.size() - 1), (this.getWidth() / 3) - 2 * offsetX, 90);
//
//          g2.setColor(Color.BLACK);
//          g2.drawString("Average: " + averageFit.get(averageFit.size() - 1), (2 * this.getWidth() / 3) - 2 * offsetX,
//                  90);
//
//          g2.setColor(Color.MAGENTA);
//          g2.drawString("Highest: " + highFit.get(lowFit.size() - 1), this.getWidth() - 2 * offsetX, 90);

        // Draws horizontal line at max fitness
        Font smallFont = new Font(null, Font.PLAIN, 10);
        g2.setFont(smallFont);
        g2.drawString("Max Fitness", offsetX + 20, offsetY - evoLoop.returnGeneSize());
        g2.drawLine(offsetX, offsetY - evoLoop.returnGeneSize(), this.getWidth() - offsetX,
                offsetY - evoLoop.returnGeneSize());
        // Graphs fitness
        int n = 0;
        for (int j = 0; j < i; j++) {

//          g2.setColor(Color.GREEN);
//          g2.drawString("Lowest: " + lowFit.get(j), (this.getWidth() / 3) - 2 * offsetX, 90);
//
//          g2.setColor(Color.BLACK);
//          g2.drawString("Average: " + averageFit.get(j), (2 * this.getWidth() / 3) - 2 * offsetX,
//                  90);
//
//          g2.setColor(Color.MAGENTA);
//          g2.drawString("Highest: " + highFit.get(j), this.getWidth() - 2 * offsetX, 90);

            BasicStroke brush = new BasicStroke(3);
            g2.setStroke(brush);
            g2.setColor(Color.BLACK);
            g2.drawLine(n + offsetX, offsetY - averageFit.get(j), n + 5 + offsetX, offsetY - averageFit.get(j + 1));
            g2.setColor(Color.GREEN);
            int low = lowFit.get(j);
            g2.drawLine(n + offsetX, offsetY - low, n + 5 + offsetX, offsetY - lowFit.get(j + 1));
            g2.setColor(Color.MAGENTA);
            g2.drawLine(n + offsetX, offsetY - highFit.get(j), n + 5 + offsetX, offsetY - highFit.get(j + 1));

            pointsAve.put(j + offsetX, offsetY - averageFit.get(j));
            pointsHigh.put(j + offsetX, offsetY - highFit.get(j));
            pointsLow.put(j + offsetX, offsetY - low);
            n += scale;

        }
        //
        i++;
    }
```

EvolutionComponent paintComponent() This is the paint method for evolution component. It draws the entire graph and fitness lines in one method. To fix this, I

extracted methods to draw labels, axis and ticks, and the fitness lines. The resulting classes were easier to read and the general method was easier to comprehend.

After refactoring

```
import javax.swing.*;

public class FitnessGraphPanel extends JComponent {

    private EvolutionComponent evolutionComponent;

    private int i = 0;


    public FitnessGraphPanel(EvolutionComponent evolutionComponent) {
        this.evolutionComponent = evolutionComponent;
        setOpaque(false);
    }

    public void setComponent(EvolutionComponent component) {
        this.evolutionComponent = component;
    }


    @Override
    protected void paintComponent(Graphics g) {
        Font font = new Font(null, Font.PLAIN, 10);
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D) g;
        g2.setFont(font);
        int offsetX = 100;
        int offsetY = this.getHeight() - 100;
        int tickSize = 3;
        g2.setColor(Color.LIGHT_GRAY);
        g2.drawRect(0, 0, this.getWidth(), this.getHeight());
        int textOffsetX = 15;
        int scale = 2;
        int textOffsetY = 5;
        HashMap<Integer, Integer> pointsHigh = new HashMap<Integer, Integer>();
        HashMap<Integer, Integer> pointsAve = new HashMap<Integer, Integer>();
        HashMap<Integer, Integer> pointsLow = new HashMap<Integer, Integer>();

        drawAxis(g2, offsetX, offsetY, tickSize, textOffsetX, scale, textOffsetY);

        addLabels(g2, offsetX, offsetY, textOffsetX, textOffsetY);

        if(evolutionComponent != null) {
            drawFitnessLines(g2, offsetX, offsetY, scale, pointsHigh, pointsAve, pointsLow);
        }

    }

    private void drawFitnessLines(Graphics2D g2, int offsetX, int offsetY, int scale,

    private void drawMaxFitness(Graphics2D g2, int offsetX, int offsetY, EvolutionLoop evoLoop) {

    private void addLabels(Graphics2D g2, int offsetX, int offsetY, int textOffsetX, int textOffsetY) {

    private void drawAxis(Graphics2D g2, int offsetX, int offsetY, int tickSize, int textOffsetX, int scale,



}
```

```java
public ChromosomeViewer() throws FileNotFoundException {
    this.rateInput = 0;

    // Asks for a filename to be entered.
    String filename = JOptionPane.showInputDialog("Enter Chromosome File:");

    // Converts the file of the given file name into a Chromosome.
    ChromosomeGenerator chromosomeCreator = new ChromosomeGenerator(filename);
    Chromosome chromosomeFromFile = chromosomeCreator.getChromosome();

    JFrame frame = new JFrame();
    JPanel chromosomePanel = new JPanel();
    JPanel buttonPanel = new JPanel();

    GridBagLayout grid = new GridBagLayout();

    // Sets up display grid.
    buttonPanel.setLayout(grid);
    chromosomePanel.setLayout(
            new GridLayout(chromosomeFromFile.numberOfArrays(), chromosomeFromFile.numberOfGenesInArray()));

    chromosomeFromFile.drawOn(1, chromosomePanel);

    JButton mutateButton = new JButton("Mutate");
    JButton loadButton = new JButton("Load");
    JButton saveButton = new JButton("Save");
    JButton enterRateButton = new JButton("Enter");
    JTextField rateInputBox = new JTextField(String.valueOf(this.rateInput), 10);
    JLabel label = new JLabel("Enter Mutation Rate  ");

    // ---------------------------------------------------------------------------------------
    // Where buttons are given functionality

    MutateListener mutateListener = new MutateListener(chromosomeFromFile, frame, rateInput);

    enterRateButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            String rateInputString = rateInputBox.getText();
            System.out.println(rateInputString);
            rateInput = Double.parseDouble(rateInputString);
            mutateListener.setMutationRate(rateInput);
            System.out.println(rateInput);
            label.setText("Rate Entered");
        }
    });

    mutateButton.addActionListener(mutateListener);
    loadButton.addActionListener(new LoadChromosomeFileListener());
    saveButton.addActionListener(new SaveChromosomeFileListener());
    // ---------------------------------------------------------------------------------------

    buttonPanel.add(label);
    buttonPanel.add(rateInputBox);
    buttonPanel.add(enterRateButton);
    buttonPanel.add(mutateButton);
    buttonPanel.add(loadButton);
    buttonPanel.add(saveButton);

    frame.setSize(CHROMOSOME_VIEWER_SIZE);
    frame.setTitle("Chromosome Viewer");

    frame.add(buttonPanel, BorderLayout.SOUTH);
    frame.add(chromosomePanel, BorderLayout.NORTH);

    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.pack();
    frame.setVisible(true);
}
```

ChromosomeViewer() had all its logic in the constructor so methods were extracted out

After refactoring

```java
public class ChromosomeViewer {

    public static final Dimension CHROMOSOME_VIEWER_SIZE = new Dimension(500, 500);
    private double rateInput;

    JFrame frame = new JFrame();
    JPanel chromosomePanel = new JPanel();
    JPanel buttonPanel = new JPanel();
    JButton mutateButton = new JButton("Mutate");
    JButton loadButton = new JButton("Load");
    JButton saveButton = new JButton("Save");
    JButton enterRateButton = new JButton("Enter");

    /**
     * This is the constructor of ChromosomeViewer which allows chromosomes defined
     * by the program to be displayed.
     *
     */
    public ChromosomeViewer(Chromosome inputChromosome, double mutationRate) {
        Chromosome chromosome = initializeChromosome(inputChromosome);
        this.rateInput = mutationRate;

        JLabel label = new JLabel("Enter Mutation Rate  ");
        setupGridLayout();
        initializeChromosomePanel(chromosome);
        JTextField rateInputBox = createRateInputBox();
        setupButtonPanel(label, rateInputBox);
        setupFrame();

        // Set up button functionality
        setupButtonListeners(label, rateInputBox, chromosome);
    }

    private Chromosome initializeChromosome(Chromosome inputChromosome) {
        if (inputChromosome == null) {
            return getTypedChromosome();
        } else {
            return inputChromosome;
        }
    }

    private void setupGridLayout() {
        GridBagLayout grid = new GridBagLayout();
        buttonPanel.setLayout(grid);
    }

    private void initializeChromosomePanel(Chromosome chromosome) {
        chromosomePanel.setLayout(new GridLayout(chromosome.numberOfArrays(), chromosome.numberOfGenesInArray()));
        ChromosomeDrawer.drawLongPhenotype(chromosome, chromosomePanel);
    }

    private JTextField createRateInputBox() {
        JTextField rateInputBox = new JTextField(String.valueOf(this.rateInput), 10);
        return rateInputBox;
    }

    private void setupButtonPanel(JLabel label, JTextField rateInputBox) {

    private void setupFrame() {

    private void setupButtonListeners(JLabel label, JTextField rateInputBox, Chromosome chromosome) {

    private void handleEnterRateAction(JTextField rateInputBox, JLabel label, MutateListener mutateListener) {

    private Chromosome getTypedChromosome() {
}
```

```java
public EvolutionViewer() {
    // Creates frames
    JFrame frame = new JFrame();
    frame.setPreferredSize(new Dimension(FRAME_WIDTH, FRAME_HEIGHT));
    // Creates instance of EvolutionComponent
    EvolutionComponent newComponent = new EvolutionComponent(new EvolutionLoop(populationVal, this.genomeLengthVal),
            populationVal);

    // Adding buttons and creating layout
    GridBagLayout grid = new GridBagLayout();

    JPanel inputPanel = new JPanel();
    inputPanel.setLayout(grid);

    JTextField mutationRate = new JTextField("Enter Mutation Rate", 0);
    JTextField numGenerations = new JTextField("Enter Number of Generations", 0);
    JTextField population = new JTextField("Enter Population", 0);
    JTextField genomeLength = new JTextField("Enter Genome Length", 0);

    JButton enterButton = new JButton("Start");
    JTextField elitismNumButton = new JTextField("Number of Elites", 0);
    JCheckBox terminateAtMaxButton = new JCheckBox("Terminate at Max Fitness?");
    JCheckBox crossoverOption = new JCheckBox("Crossover?");

    JButton seeFitChrom = new JButton("Show Fittest Chromosome");

    String[] selectionChoices = { "Truncation", "Roulette", "Rank" };

    final JComboBox<String> cb = new JComboBox<String>(selectionChoices);

    String[] evolveChoices = { "Regular", "Elitism" };

    final JComboBox<String> cb2 = new JComboBox<String>(evolveChoices);

    // Add things to frame
    inputPanel.add(enterButton);
    inputPanel.add(genomeLength);
    inputPanel.add(population);
    inputPanel.add(numGenerations);
    inputPanel.add(mutationRate);
    inputPanel.add(elitismNumButton);
    inputPanel.add(cb);
    inputPanel.add(cb2);
    inputPanel.add(terminateAtMaxButton);
    inputPanel.add(crossoverOption);
    inputPanel.add(seeFitChrom);

    frame.add(inputPanel, BorderLayout.SOUTH);
    frame.add(newComponent, BorderLayout.CENTER);

    JButton clear = new JButton("Clear All");

    // Add button panel on the right side

    // Starts the simulator
    Timer t = new Timer(DELAY, new ActionListener() {
        public int ticks = 0;

        @Override
        public void actionPerformed(ActionEvent arg0) {
            if (ticks == numGenerationsVal - 1) {
                return;
            }

            if (terminateAtMaxButton.isSelected()) {
                if (newComponent.highFit.get(ticks) == (Integer) genomeLengthVal) {
                    return;
                }

            }

            frame.repaint();
            newComponent.repaint();
            // new ChromosomeViewer(newComponent.fittest.get(ticks));
            ticks++;

        }

    });

    enterButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            clicked++;
            if (clicked == 1) {
                enterButton.setText("Pause");

                genomeLengthVal = Integer.parseInt(genomeLength.getText());
                numGenerationsVal = Integer.parseInt(numGenerations.getText());
                populationVal = Integer.parseInt(population.getText());
                mutationRateVal = Integer.parseInt(mutationRate.getText());
                selectionType = cb.getSelectedItem().toString();
                evolveType = cb2.getSelectedItem().toString();
                elitismNum = Integer.parseInt(elitismNumButton.getText());
                newComponent.genSize = numGenerationsVal;
                newComponent.startUp(populationVal, genomeLengthVal);

                if (selectionType.equals("Truncation")) {
                    if (evolveType.equals("Regular")) {
                        for (int i = 0; i < numGenerationsVal; i++) {
                            newComponent.runTruncation(crossoverOption.isSelected(), mutationRateVal);
                        }
```

EvolutionViewer() had all of its logic in the constructor so methods were extracted for getting components, making panels, and getting inputs.

After refactoring

```
public EvolutionViewer() {
    FitnessGraphPanel graphPanel = new FitnessGraphPanel(evolutionComponent);
    JFrame frame = setUpFrame(graphPanel);

    Timer timer = setUpTimer(frame, graphPanel);


    getViewerSwingComponents().enterButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            clicked++;
            if (clicked == 1) {
                getViewerComponents();
                graphPanel.setComponent(evolutionComponent);
                evolutionComponent.genSize = numGenerationsVal;
                evolutionComponent.startUp(populationVal, genomeLengthVal);
                getInputs();
                timer.start();
                clicked++;
            }
            else if (clicked % 2 == 1) {
                timer.stop();
                getViewerSwingComponents().enterButton.setText("Start");
            } else {
                timer.start();
                getViewerSwingComponents().enterButton.setText("Pause");
            }
        }

        private void getInputs() {

        private void getViewerComponents() {
            getViewerSwingComponents().enterButton.setText("Pause");
            setDefaults(getViewerSwingComponents().componentToText);
            extractMapValues();
            selectionType = getViewerSwingComponents().selectionBox.getSelectedItem().toString();
            fitnessType = getViewerSwingComponents().fitnessBox.getSelectedItem().toString();
            evolveType = getViewerSwingComponents().evolveTypeBox.getSelectedItem().toString();
            evolutionComponent = new EvolutionComponent(new EvolutionLoop(numGenerationsVal, genomeLengthVal), populationVal);
        }

    });

    getViewerSwingComponents().showFittestButton.addActionListener(new ActionListener() {
    frame.pack();
    frame.setVisible(true);
}

private JFrame setUpFrame(FitnessGraphPanel graphPanel) {
    JFrame frame = new JFrame();
    frame.add(graphPanel, BorderLayout.CENTER);
    GridBagLayout grid = new GridBagLayout();
    JPanel inputPanel = new JPanel();
    inputPanel.setLayout(grid);
    inputPanel.setPreferredSize(new Dimension(50, 50));
    viewerSwingComponents = new EvoViewerSwingComponents(frame, inputPanel);
    initializeTextFieldToDefault();
    return frame;
}

private Timer setUpTimer(JFrame frame, FitnessGraphPanel graphPanel) {
    Timer timer = new Timer(DELAY, new ActionListener() {
        public int ticks = 0;

        @Override
        public void actionPerformed(ActionEvent arg0) {
            if (ticks == numGenerationsVal - 1) {
                return;
            }
            if (getViewerSwingComponents().terminateAtMaxButton.isSelected()) {
                if (evolutionComponent.highFit.get(ticks) == (Integer) genomeLengthVal) {
                    return;
                }
            }
            frame.repaint();
            graphPanel.repaint();
            ticks++;
        }
    });
    return timer;
}
```

```
public PopulationViewer(ArrayList<Chromosome> population, JFrame frame) {
    this.frame = frame;
    this.population = population;

    JPanel populationPanel = new JPanel();
    GridBagLayout grid = new GridBagLayout();

    // Sets up display grid.
    populationPanel.setLayout(grid);
    populationPanel
            .setLayout(new GridLayout((int) (population.get(0).numberOfArrays() * Math.pow(population.size(), 0.5)),
                    (int) (population.get(0).numberOfGenes() * Math.pow(population.size(), 0.5))));

    for (int j = 0; j < 100; j++) {
        this.population.get(j).drawOn(2, populationPanel);
    }

    frame.setSize(POPULATION_VIEWER_SIZE);
    frame.setTitle("Test Population Viewer");

    frame.add(populationPanel, BorderLayout.NORTH);

    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.pack();
    frame.setVisible(true);

}
```

PopulationViewer() also had all the logic in the constructor. To fix this we extracted a method out to display the chromosome outside of the constructor.

After refactoring

```
public class PopulationViewer {
    public static final Dimension POPULATION_VIEWER_SIZE = new Dimension(500, 500);
    private ArrayList<Chromosome> population;
    private JFrame frame;

    public PopulationViewer(ArrayList<Chromosome> population, JFrame frame) {
        this.frame = frame;
        this.population = population;

        JPanel populationPanel = new JPanel();
        GridBagLayout grid = new GridBagLayout();

        // Sets up display grid.
        initializePanel(population, frame, populationPanel, grid);

    }

    private void initializePanel(ArrayList<Chromosome> population, JFrame frame, JPanel populationPanel,
            GridBagLayout grid) {
        populationPanel.setLayout(grid);
        populationPanel
                .setLayout(new GridLayout((int) (population.get(0).numberOfArrays() * Math.pow(population.size(), 0.5)),
                        (int) (population.get(0).numberOfGenes() * Math.pow(population.size(), 0.5))));

        for (int j = 0; j < 100; j++) {

            ChromosomeDrawer.drawShortPhenotype(this.population.get(j), populationPanel);
        }

        frame.setSize(POPULATION_VIEWER_SIZE);
        frame.setTitle("Test Population Viewer");

        frame.add(populationPanel, BorderLayout.NORTH);

        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.pack();
        frame.setVisible(true);
    }
```

```java
public ChromosomeGenerator(String filename) throws FileNotFoundException {
    this.filename = "ChromosomesText/" + filename + ".txt";
    FileReader reader = new FileReader(this.filename);
    Scanner scanner = new Scanner(reader);
    String rawGeneticCode = scanner.nextLine();

    // Debugger print
    System.out.println(rawGeneticCode);
    System.out.println();

    if (rawGeneticCode.length() == 20) {
        int numberOfArrays = 4;
        int lengthOfArray = 5;
        int[][] inputGeneticCode = new int[numberOfArrays][lengthOfArray];
        for (int j = 0; j < numberOfArrays; j++) {
            for (int k = 0; k < lengthOfArray; k++) {
                if (rawGeneticCode.charAt(numberOfArrays * j + k) == '0') {
                    inputGeneticCode[j][k] = 0;
                }

                else if (rawGeneticCode.charAt(numberOfArrays * j + k) == '1') {
                    inputGeneticCode[j][k] = 1;
                }

                else {
                    throw new IllegalArgumentException();
                }
            }
        }
        this.translatedChromosome = new Chromosome(inputGeneticCode);

        // Debugger print
        System.out.println(this.translatedChromosome.asString());
        System.out.println();
    }

    else if (checkIfPerfectSquare(rawGeneticCode.length())) {
        int numberOfArrays = (int) Math.sqrt(rawGeneticCode.length());
        int lengthOfArray = numberOfArrays;
        int[][] inputGeneticCode = new int[numberOfArrays][lengthOfArray];
        for (int j = 0; j < numberOfArrays; j++) {
            for (int k = 0; k < lengthOfArray; k++) {
                if (rawGeneticCode.charAt(numberOfArrays * j + k) == '0') {
                    inputGeneticCode[j][k] = 0;
                }

                else if (rawGeneticCode.charAt(numberOfArrays * j + k) == '1') {
                    inputGeneticCode[j][k] = 1;
                }

                else {
                    throw new IllegalArgumentException();
                }
            }
        }
        this.translatedChromosome = new Chromosome(inputGeneticCode);

        // Debugger print
        System.out.println(this.translatedChromosome.asString());
        System.out.println();
    }

    else {
        throw new IllegalArgumentException();
    }
}
```

ChromosomeGenerator() had all logic in the constructor. To fix this we removed some dead code that wasn't being used and extracted methods to parse incoming .txt files.

3. Lazy Class

```java
package mainApp;

public class EvolutionApp {

    public void runApp() {

        new EvolutionViewer();

    } // runApp

    /**
     * ensures: runs the application
     *
     * @param args unused
     */
    public static void main(String[] args) {
        MainApp mainApp = new MainApp();
        mainApp.runApp();
    } // main

}

```

We found a class in the project that was never used and caused a lot of issues when first testing the project as we didn't know it wasn't the correct class to run. This class runs just the evolution viewer, instead of giving the user the option to choose between evolution viewer and chromosome viewer. The class was deleted.

A note on testing: Because most of my refactorings had to do with GUI classes, I only made test for Chromosome generation.

Eli Granade:

1. Data Class

To begin with, the population class had very few methods, only amounting to holding an array of chromosomes that are then either accessed via getters and setters or inspected with the sporadic fields contained within.

```
private Population population;
private ArrayList<Chromosome> updatedSet;
private ArrayList<Chromosome> curPop;
private int genomeLengthVal, mutationRateVal, numPop;

public EvolutionLoop(int numPop, int genomeSize) {
    this.population = new Population(numPop, genomeSize);
    this.updatedSet = new ArrayList<Chromosome>();
    this.curPop = new ArrayList<Chromosome>();
    this.genomeLengthVal = genomeSize;
    this.numPop = population.returnPopSize();

}

class sortPop implements Comparator<Chromosome> {
    public int compare(Chromosome c1, Chromosome c2) {
        return c1.fitness - c2.fitness;
    }
}

public int returnGeneSize() {
    return this.population.popArray[0].numberOfArrays() * this.population.popArray[0].numberOfGenesInArray();
}

public void createPop() {
    for (int i = 0; i < this.numPop; i++) {
        curPop.add(population.popArray[i]);
    }

    for (int i = 0; i < curPop.size(); i++) {
        System.out.println(curPop.get(i).fitness);
    }
}
```

```
public void elitism(int mutate, int n) {
    if (n == population.popSize) {
        for (int i = 0; i < curPop.size(); i++) {
            updatedSet.add(curPop.get(i));
        }
    }

    else {
```

To ->

```
private Chromosome[] population;
private ArrayList<Chromosome> updatedSet;
private ArrayList<Chromosome> curPop;
private int numPop;

public EvolutionLoop(int numPop, int genomeSize) {
    this.numPop = numPop;
    this.population = new Chromosome[numPop];
    for (int i = 0; i < numPop; i++) {
        this.population[i] = new Chromosome(genomeSize);
    }
    this.updatedSet = new ArrayList<Chromosome>();
    this.curPop = new ArrayList<Chromosome>();
}

class sortPop implements Comparator<Chromosome> {
    public int compare(Chromosome c1, Chromosome c2) {
        return c1.fitness - c2.fitness;
    }
}

public int returnGeneSize() {
    return this.population[0].numberOfArrays() * this.population[0].numberOfGenesInArray();
}
```

```
public void elitism(int mutate, int n) {
    if (n == population.length) {
        for (int i = 0; i < curPop.size(); i++) {
            updatedSet.add(curPop.get(i));
        }
    }
```

Population is no longer a class in the project as a whole, and the remaining methods are no longer used.

2. Message Chain

This same data class was participating in a message chain since it would pass the chromosomes to the main class to then be operated on, with no other major changes. Therefore, removing the data class also removed this message chain.

3. Long Class

```java
public class Chromosome {

    public static final Color PHENOTYPE_OF_1_GENE = new Color(37, 244, 137);
    public static final Color PHENOTYPE_OF_0_GENE = new Color(242, 235, 226);
    public static final int PHENOTYPE_SIDE_LENGTH_1 = 50;
    public static final int PHENOTYPE_SIDE_LENGTH_2 = 5;
    public int fitness;

    private int[][] geneticCode;
    private int numberOfArrays, numberOfGenesInArray;

    public Chromosome() {
        Random random = new Random();

        this.numberOfArrays = 10;
        this.numberOfGenesInArray = 10;
        int[][] randomGeneticCode = new int[numberOfArrays][numberOfGenesInArray];
        for (int j = 0; j < numberOfArrays; j++) {
            for (int k = 0; k < numberOfGenesInArray; k++) {
                randomGeneticCode[j][k] = random.nextInt(2);
            }
        }
        this.geneticCode = randomGeneticCode;
        this.fitness = this.calculateInARowFitness();
        System.out.println(this.fitness);
    }
```

```java
public void drawOn(int option, JPanel panel) {
    if (option == 1) {
        for (int j = 0; j < this.geneticCode.length; j++) {
            for (int k = 0; k < this.geneticCode[0].length; k++) {
                if (this.geneticCode[j][k] == 0) {
                    JButton geneButton = new JButton();
                    geneButton.setPreferredSize(new Dimension(PHENOTYPE_SIDE_LENGTH_1, PHENOTYPE_SIDE_LENGTH_1));
                    geneButton.setBackground(PHENOTYPE_OF_0_GENE);
                    geneButton.addActionListener(
                            new GeneButtonListener(geneButton, PHENOTYPE_OF_0_GENE, PHENOTYPE_OF_1_GENE));
                    panel.add(geneButton);
                }

                else {
                    JButton geneButton = new JButton();
                    geneButton.setPreferredSize(new Dimension(PHENOTYPE_SIDE_LENGTH_1, PHENOTYPE_SIDE_LENGTH_1));
                    geneButton.setBackground(PHENOTYPE_OF_1_GENE);
                    geneButton.addActionListener(
                            new GeneButtonListener(geneButton, PHENOTYPE_OF_0_GENE, PHENOTYPE_OF_1_GENE));
                    panel.add(geneButton);
                }
            }
        }
    }

    else {
        for (int j = 0; j < this.geneticCode.length; j++) {
            for (int k = 0; k < this.geneticCode[0].length; k++) {
                if (this.geneticCode[j][k] == 0) {
                    JButton geneButton = new JButton();
                    geneButton.setPreferredSize(new Dimension(PHENOTYPE_SIDE_LENGTH_2, PHENOTYPE_SIDE_LENGTH_2));
                    geneButton.setBackground(PHENOTYPE_OF_0_GENE);
                    geneButton.addActionListener(
                            new GeneButtonListener(geneButton, PHENOTYPE_OF_0_GENE, PHENOTYPE_OF_1_GENE));
                    panel.add(geneButton);
                }

                else {
```

```
                    geneButton.setPreferredSize(new Dimension(PHENOTYPE_SIDE_LENGTH_1, PHENOTYPE_SIDE_LENGTH_1));
                    geneButton.setBackground(PHENOTYPE_OF_1_GENE);
                    geneButton.addActionListener(
                            new GeneButtonListener(geneButton, PHENOTYPE_OF_0_GENE, PHENOTYPE_OF_1_GENE));
                    panel.add(geneButton);
                }
            }
        }
    }

    else {
        for (int j = 0; j < this.geneticCode.length; j++) {
            for (int k = 0; k < this.geneticCode[0].length; k++) {
                if (this.geneticCode[j][k] == 0) {
                    JButton geneButton = new JButton();
                    geneButton.setPreferredSize(new Dimension(PHENOTYPE_SIDE_LENGTH_2, PHENOTYPE_SIDE_LENGTH_2));
                    geneButton.setBackground(PHENOTYPE_OF_0_GENE);
                    geneButton.addActionListener(
                            new GeneButtonListener(geneButton, PHENOTYPE_OF_0_GENE, PHENOTYPE_OF_1_GENE));
                    panel.add(geneButton);
                }

                else {
                    JButton geneButton = new JButton();
                    geneButton.setPreferredSize(new Dimension(PHENOTYPE_SIDE_LENGTH_2, PHENOTYPE_SIDE_LENGTH_2));
                    geneButton.setBackground(PHENOTYPE_OF_1_GENE);
                    geneButton.addActionListener(
                            new GeneButtonListener(geneButton, PHENOTYPE_OF_0_GENE, PHENOTYPE_OF_1_GENE));
                    panel.add(geneButton);
                }
            }
        }
    }

}
```

Chromosome was a nearly 400 line Long class. It had several troubling issues, such as duplicated code, fitness functions, mutations and crossover, in addition to data management. The main refactorings I had done was splitting the fitness functions into their own section so that they could be both standardized and improved on, as well as breaking the drawing knowledge into another class. In addition, several of the above improvements lowered the size of the class, where it sits to only manage the very complex data, and sits at around 125 lines.

These improvements came at the cost of two other issues however. By making the fitnesses their own functions, like the selection strategies, passing the data required a switch case, but this small, manageable switch case helps reduce code duplication and manage class sizes by making the separate class more understandable.