Class Descriptions and Code

Taxman Class

This class is what makes the game work. It contains all the current information in the game (current player score, taxman score, remaining numbers etc) and also contains the mechanisms for how the game works. It contains a variety of methods which keep the game moving and working correctly according to the given rules.

Taxman Class Code:

```
import java.util.Iterator;
import java.util.TreeSet;

public class Taxman {

    // member attributes
    private int upperLimit;
    private TreeSet<Integer> numbers;
    private int playerScore = 0;
    private int taxmanScore = 0;

    // given N, update `upperLimit` and
    // fill `numbers` with the necessary numbers.
    public Taxman(int n) {
            numbers = new TreeSet<Integer>();
            this.upperLimit = n; // Set the upper limit to n.
            for (int i = 1; i <= n; i++) {
                    this.numbers.add(i); // Add all numbers previous to n.
            }
    }

    // upperLimit getter
    public int getUpperLimit() {
            return this.upperLimit;
    }

    // playerScore getter
    public int getPlayerScore() {
            return this.playerScore;
    }

    // taxmanScore getter
    public int getTaxmanScore() {
            return this.taxmanScore;
    }
```

```
// return a deep copy of `numbers`
public TreeSet<Integer> getAvailableNumbers() {
        TreeSet<Integer> deepCopy = new TreeSet<Integer>();
        // Deep copy the numbers
        deepCopy.addAll(this.numbers);
        return deepCopy;
}

// return true if there are divisors of `a` in `numbers` (not including `a`)
// return false otherwise
public boolean hasDivisorsOf(int a) {
        Iterator<Integer> iter = this.numbers.iterator();
        while (iter.hasNext()) {
                int temp = (int) iter.next();
                // Return true if there is a divisor of a.
                if (a % temp == 0 && temp != a) {
                        return true;
                }
        }
        // There are no divisors.
        return false;
}

// return true if there are no longer any legal choices for player to make
public boolean gameOver() {
        Iterator<Integer> iter = this.numbers.iterator();
        while (iter.hasNext()) {
                int temp = (int) iter.next();
                // There are divisors and therefore there is a legal choice.
                if (hasDivisorsOf(temp) == true) {
                        return false;
                }
        }
        // Since the method didn't return false, add all remaining numbers in
        // numbers to the taxMan's score.
        if (numbers.size() > 0) {
                Iterator<Integer> iter2 = numbers.iterator();
                while (iter2.hasNext()) {
                        int temp = (int) iter2.next();
                        this.taxmanScore = this.taxmanScore + temp;
                }
        }
        // There are no divisors left, the game is over.
        return true;
}

// remove all the divisors of `a` in `numbers` (not including `a`)
// return a TreeSet of all the numbers you removed
private TreeSet<Integer> removeDivisorsOf(int a) {
        TreeSet<Integer> removedNumbers = new TreeSet<Integer>();
        Iterator<Integer> iter = this.numbers.iterator();
        // Count the iteration numbers incase we need to go back to a previous
        // item
        // after an element has been removed.
        int itercounter = 0;
        // Go through every item in numbers.
        while (iter.hasNext()) {
                int temp = (int) iter.next();
                // If a number is a divisor of a then AND not a itself then:
                if (a % temp == 0 && temp != a) {
                        // Add the divisor to the removed numbers treeset.
                        removedNumbers.add(temp);
                        // remove the element from numbers.
                        numbers.remove(temp);
                        // When removing an item the iterator must go back an element.
                        // Create a new iter for numbers starting from the start.
                        Iterator<Integer> newIter = this.numbers.iterator();
                        // Reset the iter.
                        iter = newIter;
                        int i = 0;
                        // Move the iter to the correct positioning.
                        while (i < itercounter) {
                                iter.next();
                                i++;
                        }
                }
                // If nothing is removed simply go to the next element in numbers.
                itercounter++;
        }
        return removedNumbers;
}

// the player chooses the number `a` from the set.
// the taxman then takes some numbers, as per the rules of the game.
// return `true` if `a` was a valid choice.
// return `false` if `a` was an invalid choice.
// this method should also update scores where necessary.
// if the game is over, the taxman should take all remaining numbers.
public boolean choose(int a) {
        // If it is a valid choice then:
        if (hasDivisorsOf(a) == true && numbers.contains(a) == true) {
                // Add the players choice to his score.
                playerScore = playerScore + a;
                TreeSet<Integer> taxMansNumbers = removeDivisorsOf(a);
```

```
			Iterator<Integer> tmnIterator = taxMansNumbers.iterator();
			// Add the taxman numbers to his score.
			while (tmnIterator.hasNext()) {
				taxmanScore = taxmanScore + tmnIterator.next();
			}
			// Remove 'a' from numbers.
			numbers.remove(a);
			// Return true as it was a valid choice.
			return true;
		} else
			return false;
	}

	// a helper function for displaying the current game state.
	public void displayGameState() {
		System.out.print("Remaining nums: ");
		for (int i = 0; i <= upperLimit; i++)
			if (this.numbers.contains(i))
				System.out.print(i + " ");
		System.out.println("\n" + "\nThe scores are now:");
		System.out.println("Player: " + playerScore + "\n" + "Taxman: " + taxmanScore +
"\n");
		if (gameOver()) {
			// Player wins
			if (playerScore > taxmanScore) {
				System.out.println("Game Over! The winner is: Player!");
			}
			// Taxman wins
			if (taxmanScore > playerScore) {
				System.out.println("Game Over! The winner is: Taxman");
			}
			// Tie
			if (taxmanScore == playerScore) {
				System.out.println("Game Over! It's a tie!");
			}
			// Final Score Report
			System.out.print("Player: " + playerScore + "\n" + "Taxman: " +
taxmanScore);
		}
	}
}
```

NaivePlayer Class

This class chooses the highest number from a selection of numbers. That is the only function of the class. It picks the highest possible number in the selection given and is therefore not a very effective player class for getting a good score.

NaivePlayer Class Code:

```
import java.util.Iterator;
import java.util.TreeSet;

public class NaivePlayer implements Player {

    @Override
    // This method will find the highest legal choice that the player can take
    // each turn.
    public int chooseNum(TreeSet<Integer> avail) {
        // Implement a Naive strategy for choosing which number the
        // player will take. return which number you choose for this turn.
        int bestChoice = 0;
        Iterator<Integer> iter = avail.iterator();
        while (iter.hasNext()) {
            int temp = (int) iter.next();
            if (hasDivisorsOf(temp, avail) == true && temp > bestChoice) {
                bestChoice = temp;
            }
        }
        return bestChoice;
    }

    // Check if 'a' has divisors in x.
    public boolean hasDivisorsOf(int a, TreeSet<Integer> inputSet) {
        // Check every number in input set to see if it is a divisor of a.
        Iterator<Integer> iter = inputSet.iterator();
        while (iter.hasNext()) {
            int temp = (int) iter.next();
            // Return true if there is a divisor of a.
            if (a % temp == 0 && temp != a) {
                return true;
            }
        }
        // There are no divisors.
        return false;
    }
}
```

GreedyPlayer Class

This class picks the absolute best immediate option from the input numbers. This is done by selecting the number which gives the player the highest score and the taxman the lowest score in comparison. It goes through every choice option from the input numbers and plays out what the scores would be if that choice is taken. Then it compares the player score and the taxman score to find out which has the largest score difference in the player's favour. Therefore finding the best immediate score.

GreedyPlayer Class Code:

```
import java.util.Iterator;
import java.util.TreeSet;

public class GreedyPlayer implements Player {

    @Override
    public int chooseNum(TreeSet<Integer> avail) {
            // This greedy strategy finds the element which has the greatest
            // increase in score, compared to the score that the taxman will get
            // from the divisors total.
            Iterator<Integer> iter = avail.iterator();
            // This will play the role of the best possible choice. It will be
            // updated as calculations commence until it is finally used to return
            // the best choice.
            int bestChoice = 0;
            // Start of with a low value so that the scoreGap will always be changed
            // on the first instance of the algorithm(no matter how small the
            // difference, it will always be larger than -100)
            int scoreGap = -100;
            int taxmanPoints = 0;
            int playerPoints = 0;
            // Go through the list
            while (iter.hasNext()) {
                    int temp = (int) iter.next();
                    // Does the number even have divisors (is it an option)
                    if (hasDivisorsOf(temp, avail) == true) {
                            // Value of the element.
                            playerPoints = temp;
                            // Add up taxmanPoints
                            taxmanPoints = getDivisorTotal(temp, avail);
                            // If this scoregap is larger than the previous score gap, then
                            // it is a better option.
                            if ((playerPoints - taxmanPoints) > scoreGap) {
                                    // The new best element for this selection.
                                    bestChoice = temp;
                                    // Update the best scoregap
```

```
                        scoreGap = playerPoints - taxmanPoints;
                    }
                }
            }
            // Return the element which has the best possible immediate outcome for
            // the human player.
            return bestChoice;
    }

    // Check if 'a' has divisors in x.
    public boolean hasDivisorsOf(int a, TreeSet<Integer> x) {
            Iterator<Integer> iter = x.iterator();
            while (iter.hasNext()) {
                int temp = (int) iter.next();
                // Return true if there is a divisor of a.
                if (a % temp == 0 && temp != a) {
                    return true;
                }
            }
            // There are no divisors.
            return false;
    }

    // Returns an integer value for all the divisors of 'a' in TreeSet 'x' added
    // together.
    private int getDivisorTotal(int a, TreeSet<Integer> x) {
            TreeSet<Integer> divisors = new TreeSet<Integer>();
            Iterator<Integer> iter = x.iterator();
            int total = 0;
            // Go through every item in numbers.
            while (iter.hasNext()) {
                int temp = (int) iter.next();
                // If a number is a divisor of a then AND not a itself then:
                if (a % temp == 0 && temp != a) {
                    // Add the divisor to the divisor numbers treeset.
                    divisors.add(temp);
                }
            }
            // Go through all the divisors and add it to the running total of the
            // divisors.
            Iterator<Integer> divIter = divisors.iterator();
            while (divIter.hasNext()) {
                int divTemp = (int) divIter.next();
                total = total + divTemp;
            }
            // Return all the divisors added up.
            return total; }}
```

OptimalPlayer Class

This class chooses the best overall choice for this set of numbers. It does this by searching through every single choice path possible in this set of numbers and returns with the path that gives the player the best score and the taxman the lowest score in comparison. It uses a recursive method to go through the choices of the input numbers.

OptimalPlayer Class Code:

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.TreeSet;

public class OptimalPlayer implements Player {

    // This choice tree will hold the series of choices which gives the best
    // possible player score, in comparison to the TaxMans score.
    public static ChoiceTree bestChoiceTree = new ChoiceTree();
    // This array list will hold the values of the best choice list. It will
    // then be used to feed integers into the chooseNum method.
    public static ArrayList<Integer> bestChoiceList = new ArrayList<Integer>();
    // This boolean switch is used to notify chooseNum if the getBestTree
    // algorithm has been run yet or not (False = hasn't been run. True = has
    // been run)
    public boolean TreeFound = false;

    @Override
    public int chooseNum(TreeSet<Integer> avail) {

            // If the best choice tree finding algorithm hasn't been run. Run it.
            if (TreeFound == false) {

                    // Send through blank tree and blank choice list, as no choices have
                    // been made yet.
                    ChoiceTree blankTree = new ChoiceTree();
                    ArrayList<Integer> blankList = new ArrayList<Integer>();
                    getBestTree(avail, blankTree, blankList);
            }
            return getNextChoice(bestChoiceList);
    }

    // This recursive method inputs 3 variables:
    // - A TreeSet of Integers. This holds the selection of numbers available
    // for this turns choice.
    // - A ChoiceTree object. This holds the current scores so far in this
    // series of turns.
```

```
   // - An array of previous choices. This is what the program uses to
   // calculate the best possible set of choices for the player.
   // This method goes through every single possible choice path from the
   // original input of numbers and finds the one that ends with the largest
   // difference in players score and taxman score.
   public void getBestTree(TreeSet<Integer> inputNumbers, ChoiceTree inputTree,
ArrayList<Integer> choiceList) {

        // Base Case (If there are no more legal choices left with the inputed
        // numbers)
        if (gameOver(inputNumbers, inputTree) == true) {

                // If this series of choices ended up having a larger score
                // difference than the previous largest. Then make this series the
                // new best.
                if (inputTree.getScoreDifference() >= bestChoiceTree.getScoreDifference()) {
                        bestChoiceTree = inputTree;
                        bestChoiceList = choiceList;
                }
        }

        // If the game is not over (meaning there are still more paths to go
        // down)
        else {

                // Create a TreeSet of all the possible legal choices in
                // inputNumbers.
                TreeSet<Integer> legalChoices = new TreeSet<Integer>();

                // Go through all the input numbers.
                Iterator<Integer> inputNumbersIterator = inputNumbers.iterator();
                while (inputNumbersIterator.hasNext()) {
                        int selectedChoice = (int) inputNumbersIterator.next();

                        // If it's a legal choice:
                        if (hasDivisorsOf(inputNumbers, selectedChoice) == true) {

                                // Add it to the legal choice TreeSet.
                                legalChoices.add(selectedChoice);
                        }
                }

                // Go through every legal choice and simulate what happens if you
                // select each on as the choice.
                Iterator<Integer> legalChoicesIterator = legalChoices.iterator();
                while (legalChoicesIterator.hasNext()) {
                        // The current selected choice from legal choices.
                        int currentChoice = (int) legalChoicesIterator.next();

                        // Make a deep copy of the input numbers so that they can be
                        // altered without effecting the rest of the legal choice
                        // iterations.
                        TreeSet<Integer> currentChoiceNumbers = new TreeSet<Integer>();
                        currentChoiceNumbers = copyNumbers(inputNumbers);

                        // Make a deep copy of the inputed tree. This is so the tree
                        // can break off into different states without altering the
                        // original inputed tree.
                        ChoiceTree treeCopy = new ChoiceTree();
                        treeCopy = copyTree(inputTree);

                        // Make a deep copy of the inputed choice list. This is so it
                        // can remain its own encapsulated list as it continues to go
                        // down a path.
                        ArrayList<Integer> copyList = new ArrayList<Integer>();
                        copyList.addAll(choiceList);

                        // Update the information in this new tree based on the
                        // selection of this iteration.
                        TreeSet<Integer> taxMansNumber =
removeDivisorsOf(currentChoiceNumbers, currentChoice);
                        treeCopy.addChoice(currentChoice, taxMansNumber);
                        copyList.add(currentChoice);
                        currentChoiceNumbers.remove(currentChoice);

                        // Make a recursive call now with the new set of numbers and the
                        // tree.
                        getBestTree(currentChoiceNumbers, treeCopy, copyList);
                }
        }

        // After every single path option has been tested. Change the boolean
        // switch (TreeFound) to true as this algorithm has been run.
        TreeFound = true;
   }

   // Return true if there are no longer any legal choices for player to make
   // with the inputed series of numbers. Additionally input the choice tree
   // so that if the game is over, the remaining numbers in the list can be
   // added to the TaxMans score.
   public boolean gameOver(TreeSet<Integer> numbers, ChoiceTree tree) {
        Iterator<Integer> iter = numbers.iterator();
        // Go through every number.
        while (iter.hasNext()) {
            int selectedNumber = (int) iter.next();
            // There are divisors and therefore there is a legal choice left.
            if (hasDivisorsOf(numbers, selectedNumber) == true) {
                return false;
            }
        }
        // Since the game has in fact ended. The remaining numbers are added to
        // the taxmans score (If there are any numbers left).
        if (numbers.size() > 0) {
            tree.updateTaxmanTotal(numbers);
        }
        // There are no divisors(therefore choices) left, the game is over.
        return true;
    }

    // return true if there are divisors of `a` in `numbers` (not including `a`)
    // return false otherwise
    public boolean hasDivisorsOf(TreeSet<Integer> numbers, int a) {
        Iterator<Integer> iter = numbers.iterator();
        while (iter.hasNext()) {
            int temp = (int) iter.next();
            // Return true if there is a divisor of a.
            if (a % temp == 0 && temp != a) {
                return true;
            }
        }
        // There are no divisors.
        return false;
    }

    // return a deep copy of `numbers`
    public TreeSet<Integer> copyNumbers(TreeSet<Integer> numbers) {
        TreeSet<Integer> deepCopy = new TreeSet<Integer>();
        // Deep copy the numbers
        deepCopy.addAll(numbers);
        return deepCopy;
    }

    // return a deep copy of 'choiceTree'
    public ChoiceTree copyTree(ChoiceTree x) {
        ChoiceTree copy = new ChoiceTree();
        // Deep copy values
        copy.setPlayerTotal(x.getPTotal());
        copy.setTaxManTotal(x.getTMTotal());
        return copy;
    }
    // remove all the divisors of `a` in `numbers` (not including `a`)
    // return a TreeSet of all the numbers you removed
    private TreeSet<Integer> removeDivisorsOf(TreeSet<Integer> numbers, int a) {
        TreeSet<Integer> removedNumbers = new TreeSet<Integer>();
        Iterator<Integer> iter = numbers.iterator();
        // Count the iteration numbers incase we need to go back to a previous
        // item after an element has been removed.
        int itercounter = 0;
        // Go through every item in numbers.
        while (iter.hasNext()) {
            int temp = (int) iter.next();
            // If a number is a divisor of a then AND not 'a' itself then:
            if (a % temp == 0 && temp != a) {
                // Add the divisor to the removed numbers treeset.
                removedNumbers.add(temp);
                // remove the element from numbers.
                numbers.remove(temp);
                // When removing an item the iterator must go back an element.
                // Create a new iter for numbers starting from the start.
                Iterator<Integer> newIter = numbers.iterator();
                // Reset the iter.
                iter = newIter;
                int i = 0;
                // Move the iter to the correct positioning.
                while (i < itercounter) {
                    iter.next();
                    i++;
                }
            }
            // If nothing is removed simply go to the next element in numbers.
            itercounter++;
        }
        return removedNumbers;
    }

    // Return the next choice in this choice tree. Then remove it from the list
    // so that the next element can be accessed.
    public int getNextChoice(ArrayList<Integer> x) {
        int temp = x.get(0);
        x.remove(0);
        return temp;
    }
}
```

ChoiceTree Class

This class is a helper class for the class OptimalPlayer. It keeps track of simulated game scores within OptimalPlayer class so that they can be compared later in order to find the best possible choice path.

ChoiceTree Class Code:

```
import java.util.Iterator;
import java.util.TreeSet;

//The point of this class is to create objects which hold specific game scores related to a
choice list (outside this class object).
public class ChoiceTree {
    public int taxManTotal;
    public int playerTotal;

    // Constructor
    public ChoiceTree() {
        this.taxManTotal = 0;
        this.playerTotal = 0;
    }

    // Update the totals based on the input.
    public void addChoice(int selected, TreeSet<Integer> divisors) {
        // Update the totals.
        updatePlayerTotal(selected);
        updateTaxmanTotal(divisors);
    }

    // Updates the players total score
    public void updatePlayerTotal(int x) {
        this.playerTotal = this.playerTotal + x;
    }

    // Updates the TaxMans total score
    public void updateTaxmanTotal(TreeSet<Integer> x) {
        Iterator<Integer> numbersIterator = x.iterator();
        while (numbersIterator.hasNext()) {
            this.taxManTotal = this.taxManTotal + numbersIterator.next();
        }
    }

    // Getter for TaxManTotal
    public int getTMTotal() {
        return this.taxManTotal;
    }

    // Setter for TaxManTotal
    public void setTaxManTotal(int x) {
            this.taxManTotal = x;
    }

    // Getter for playerTotal
    public int getPTotal() {
            return this.playerTotal;
    }

    // Setter for playerTotal
    public void setPlayerTotal(int x) {
            this.playerTotal = x;
    }

    // Calculates the difference between playerTotal and TaxManTotal.
    public int getScoreDifference() {
            return this.playerTotal - this.taxManTotal;
    }

}
```

Class Outputs

InputPlayer Output

Output for upper limit 6. Various choice paths

```
Please choose an upper limit between 2 and 25. 6
Which number do you choose? The choices are:
[1, 2, 3, 4, 5, 6]
6

Player chose 6
Remaining nums: 3 4 5

The scores are now:
Player: 6
Taxman: 3

Game Over! The winner is: Taxman
Player: 6
Taxman: 15
```

```
Please choose an upper limit between 2 and 25. 6
Which number do you choose? The choices are:
[1, 2, 3, 4, 5, 6]
4

Player chose 4
Remaining nums: 3 5 6

The scores are now:
Player: 4
Taxman: 3

Which number do you choose? The choices are:
[3, 5, 6]
6

Player chose 6
Remaining nums: 5

The scores are now:
Player: 10
Taxman: 6

Game Over! The winner is: Taxman
Player: 10
Taxman: 11
```

```
Please choose an upper limit between 2 and 25. 6
Which number do you choose? The choices are:
[1, 2, 3, 4, 5, 6]
5

Player chose 5
Remaining nums: 2 3 4 6

The scores are now:
Player: 5
Taxman: 1

Which number do you choose? The choices are:
[2, 3, 4, 6]
4

Player chose 4
Remaining nums: 3 6

The scores are now:
Player: 9
Taxman: 3

Which number do you choose? The choices are:
[3, 6]
6

Player chose 6
Remaining nums:

The scores are now:
Player: 15
Taxman: 6

Game Over! The winner is: Player!
Player: 15
Taxman: 6
```

Output for upper limit 11. Various choice paths

```
Please choose an upper limit between 2 and 25. 11
Which number do you choose? The choices are:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
10

Player chose 10
Remaining nums: 3 4 6 7 8 9 11

The scores are now:
Player: 10
Taxman: 8

Which number do you choose? The choices are:
[3, 4, 6, 7, 8, 9, 11]
8

Player chose 8
Remaining nums: 3 6 7 9 11

The scores are now:
Player: 18
Taxman: 12

Which number do you choose? The choices are:
[3, 6, 7, 9, 11]
6

Player chose 6
Remaining nums: 7 9 11

The scores are now:
Player: 24
Taxman: 15

Game Over! The winner is: Taxman
Player: 24
Taxman: 42
```

```
Please choose an upper limit between 2 and 25. 11
Which number do you choose? The choices are:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
8

Player chose 8
Remaining nums: 3 5 6 7 9 10 11

The scores are now:
Player: 8
Taxman: 7
```

```
Which number do you choose? The choices are:
[3, 5, 6, 7, 9, 10, 11]
6

Player chose 6
Remaining nums: 5 7 9 10 11

The scores are now:
Player: 14
Taxman: 10

Which number do you choose? The choices are:
[5, 7, 9, 10, 11]
10

Player chose 10
Remaining nums: 7 9 11

The scores are now:
Player: 24
Taxman: 15

Game Over! The winner is: Taxman
Player: 24
Taxman: 42
```

```
Please choose an upper limit between 2 and 25. 11
Which number do you choose? The choices are:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
4

Player chose 4
Remaining nums: 3 5 6 7 8 9 10 11

The scores are now:
Player: 4
Taxman: 3

Which number do you choose? The choices are:
[3, 5, 6, 7, 8, 9, 10, 11]
10

Player chose 10
Remaining nums: 3 6 7 8 9 11

The scores are now:
Player: 14
Taxman: 8

Which number do you choose? The choices are:
[3, 6, 7, 8, 9, 11]
9
```

```

Player chose 9
Remaining nums: 6 7 8 11

The scores are now:
Player: 23
Taxman: 11

Game Over! The winner is: Taxman
Player: 23
Taxman: 43
```

Output for upper limit 15. Various choice paths

```
Please choose an upper limit between 2 and 25. 15
Which number do you choose? The choices are:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
15

Player chose 15
Remaining nums: 2 4 6 7 8 9 10 11 12 13 14

The scores are now:
Player: 15
Taxman: 9

Which number do you choose? The choices are:
[2, 4, 6, 7, 8, 9, 10, 11, 12, 13, 14]
10

Player chose 10
Remaining nums: 4 6 7 8 9 11 12 13 14

The scores are now:
Player: 25
Taxman: 11

Which number do you choose? The choices are:
[4, 6, 7, 8, 9, 11, 12, 13, 14]
8

Player chose 8
Remaining nums: 6 7 9 11 12 13 14

The scores are now:
Player: 33
Taxman: 15

Which number do you choose? The choices are:
[6, 7, 9, 11, 12, 13, 14]
12

Player chose 12
Remaining nums: 7 9 11 13 14

The scores are now:
Player: 45
Taxman: 21

Which number do you choose? The choices are:
[7, 9, 11, 13, 14]
14

Player chose 14
Remaining nums: 9 11 13
```

```
The scores are now:
Player: 59
Taxman: 28

Game Over! The winner is: Taxman
Player: 59
Taxman: 61
```

```
Please choose an upper limit between 2 and 25. 15
Which number do you choose? The choices are:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
14

Player chose 14
Remaining nums: 3 4 5 6 8 9 10 11 12 13 15

The scores are now:
Player: 14
Taxman: 10

Which number do you choose? The choices are:
[3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 15]
9

Player chose 9
Remaining nums: 4 5 6 8 10 11 12 13 15

The scores are now:
Player: 23
Taxman: 13

Which number do you choose? The choices are:
[4, 5, 6, 8, 10, 11, 12, 13, 15]
12

Player chose 12
Remaining nums: 5 8 10 11 13 15

The scores are now:
Player: 35
Taxman: 23

Which number do you choose? The choices are:
[5, 8, 10, 11, 13, 15]
15

Player chose 15
Remaining nums: 8 10 11 13

The scores are now:
Player: 50
Taxman: 28
```

```
Game Over! The winner is: Taxman
Player: 50
Taxman: 70
```

```
Please choose an upper limit between 2 and 25. 15
Which number do you choose? The choices are:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
2

Player chose 2
Remaining nums: 3 4 5 6 7 8 9 10 11 12 13 14 15

The scores are now:
Player: 2
Taxman: 1

Which number do you choose? The choices are:
[3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
6

Player chose 6
Remaining nums: 4 5 7 8 9 10 11 12 13 14 15

The scores are now:
Player: 8
Taxman: 4

Which number do you choose? The choices are:
[4, 5, 7, 8, 9, 10, 11, 12, 13, 14, 15]
8

Player chose 8
Remaining nums: 5 7 9 10 11 12 13 14 15

The scores are now:
Player: 16
Taxman: 8

Which number do you choose? The choices are:
[5, 7, 9, 10, 11, 12, 13, 14, 15]
10

Player chose 10
Remaining nums: 7 9 11 12 13 14 15

The scores are now:
Player: 26
Taxman: 13

Which number do you choose? The choices are:
[7, 9, 11, 12, 13, 14, 15]
14
```

```
Player chose 14
Remaining nums: 9 11 12 13 15

The scores are now:
Player: 40
Taxman: 20

Game Over! The winner is: Taxman
Player: 40
Taxman: 80
```

Output for upper limit 25. Various choice paths

```
Please choose an upper limit between 2 and 25. 25
Which number do you choose? The choices are:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25]
25

Player chose 25
Remaining nums: 2 3 4 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

The scores are now:
Player: 25
Taxman: 6

Which number do you choose? The choices are:
[2, 3, 4, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]
10

Player chose 10
Remaining nums: 3 4 6 7 8 9 11 12 13 14 15 16 17 18 19 20 21 22 23 24

The scores are now:
Player: 35
Taxman: 8

Which number do you choose? The choices are:
[3, 4, 6, 7, 8, 9, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]
16

Player chose 16
Remaining nums: 3 6 7 9 11 12 13 14 15 17 18 19 20 21 22 23 24

The scores are now:
Player: 51
Taxman: 20

Which number do you choose? The choices are:
[3, 6, 7, 9, 11, 12, 13, 14, 15, 17, 18, 19, 20, 21, 22, 23, 24]
9

Player chose 9
Remaining nums: 6 7 11 12 13 14 15 17 18 19 20 21 22 23 24

The scores are now:
Player: 60
Taxman: 23

Which number do you choose? The choices are:
[6, 7, 11, 12, 13, 14, 15, 17, 18, 19, 20, 21, 22, 23, 24]
22

Player chose 22
Remaining nums: 6 7 12 13 14 15 17 18 19 20 21 23 24
```

```

The scores are now:
Player: 82
Taxman: 34

Which number do you choose? The choices are:
[6, 7, 12, 13, 14, 15, 17, 18, 19, 20, 21, 23, 24]
24

Player chose 24
Remaining nums: 7 13 14 15 17 18 19 20 21 23

The scores are now:
Player: 106
Taxman: 52

Which number do you choose? The choices are:
[7, 13, 14, 15, 17, 18, 19, 20, 21, 23]
14

Player chose 14
Remaining nums: 13 15 17 18 19 20 21 23

The scores are now:
Player: 120
Taxman: 59

Game Over! The winner is: Taxman
Player: 120
Taxman: 205
```

```
Please choose an upper limit between 2 and 25. 25
Which number do you choose? The choices are:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25]
10

Player chose 10
Remaining nums: 3 4 6 7 8 9 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

The scores are now:
Player: 10
Taxman: 8

Which number do you choose? The choices are:
[3, 4, 6, 7, 8, 9, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25]
8

Player chose 8
Remaining nums: 3 6 7 9 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

The scores are now:
Player: 18
Taxman: 12
```

```
Which number do you choose? The choices are:
[3, 6, 7, 9, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25]
22

Player chose 22
Remaining nums: 3 6 7 9 12 13 14 15 16 17 18 19 20 21 23 24 25

The scores are now:
Player: 40
Taxman: 23

Which number do you choose? The choices are:
[3, 6, 7, 9, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 23, 24, 25]
18

Player chose 18
Remaining nums: 7 12 13 14 15 16 17 19 20 21 23 24 25

The scores are now:
Player: 58
Taxman: 41

Which number do you choose? The choices are:
[7, 12, 13, 14, 15, 16, 17, 19, 20, 21, 23, 24, 25]
24

Player chose 24
Remaining nums: 7 13 14 15 16 17 19 20 21 23 25

The scores are now:
Player: 82
Taxman: 53

Which number do you choose? The choices are:
[7, 13, 14, 15, 16, 17, 19, 20, 21, 23, 25]
14

Player chose 14
Remaining nums: 13 15 16 17 19 20 21 23 25

The scores are now:
Player: 96
Taxman: 60

Game Over! The winner is: Taxman
Player: 96
Taxman: 229
```

```
Please choose an upper limit between 2 and 25. 25
Which number do you choose? The choices are:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25]
14
```

```
Player chose 14
Remaining nums: 3 4 5 6 8 9 10 11 12 13 15 16 17 18 19 20 21 22 23 24 25

The scores are now:
Player: 14
Taxman: 10

Which number do you choose? The choices are:
[3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25]
20

Player chose 20
Remaining nums: 3 6 8 9 11 12 13 15 16 17 18 19 21 22 23 24 25

The scores are now:
Player: 34
Taxman: 29

Which number do you choose? The choices are:
[3, 6, 8, 9, 11, 12, 13, 15, 16, 17, 18, 19, 21, 22, 23, 24, 25]
22

Player chose 22
Remaining nums: 3 6 8 9 12 13 15 16 17 18 19 21 23 24 25

The scores are now:
Player: 56
Taxman: 40

Which number do you choose? The choices are:
[3, 6, 8, 9, 12, 13, 15, 16, 17, 18, 19, 21, 23, 24, 25]
12

Player chose 12
Remaining nums: 8 9 13 15 16 17 18 19 21 23 24 25

The scores are now:
Player: 68
Taxman: 49

Which number do you choose? The choices are:
[8, 9, 13, 15, 16, 17, 18, 19, 21, 23, 24, 25]
18

Player chose 18
Remaining nums: 8 13 15 16 17 19 21 23 24 25

The scores are now:
Player: 86
Taxman: 58

Which number do you choose? The choices are:
```

```
[8, 13, 15, 16, 17, 19, 21, 23, 24, 25]
16

Player chose 16
Remaining nums: 13 15 17 19 21 23 24 25

The scores are now:
Player: 102
Taxman: 66

Game Over! The winner is: Taxman
Player: 102
Taxman: 223
```

Greedy vs Optimal Output Comparison Table

N	**Strategy (Greedy)**	**Winning Margin**	**Strategy (Optimal)**	**Winning Margin**
2	2	1	2	1
3	3	0	3	0
4	3,4	4	3,4	4
5	5,4	3	5,4	3
6	5,4,6	9	5,4,6	9
7	7,4,6	6	7,4,6	6
8	7,4,6	-2	7,8,6	6
9	7,9,6,8	15	7,9,6,8	15
10	7,9,6,10,8	25	7,9,6,10,8	25
11	11,9,6,10,8	22	11,9,6,10,8	22
12	11,9,6,12,10	18	11,10,9,8,12	22
13	13,9,6,12,10	9	13,10,9,8,12	13
14	19,14,9,10,8,12	27	13,14,10,9,8,12	27
15	13,15,10,14,8,12	24	13,9,15,10,14,8, 12	42
16	13,15,10,14,8,12	8	13,9,15,10,16,14,1 2	42
17	17,15,10,14,8,12	-1	17,9,15,10,16,14,1 2	33
18	17,15,10,14,8,12,1 8	17	17,9,15,10.18,14,1 2,16	51
19	19,15,10,14,8,12,1 8	2	19,9,15,10,18,14,1 2,16	36
20	19,15,10,20,16,14, 12,18	38	19,15,10,20,16,14, 12,18	34
21	19,21,14,15,20,16, 12,18	39	19,9,21,15,14,18,1 2,20,16	57

22	12,21,14,22,15,20,16,12,18	61	19,9,21,15,14,22,18,12,20,16	79
23	23,21,14,22,15,20,16,12,18	46	23,9,21,15,14,22,18,12,20,16	64
24	23,21,14,22,15,20,16,12,18	22	23,9,21,15,14,22,20,18,16,24	64
25	23,25,15,21,14,22,20,16,12,18	47	23,25,15,21,14,24,22,20,18,16	71

Design Decision Discussion

TaxMan

The design decisions that went into the Taxman class were definitely premeditated. I reviewed the details given in the assignment outline and created a more succinct set of rules which I could apply to the template given. Once the rules were concreted in front of me, writing code to produce these rules was very easy. An example of rules or details that were transformed into code could be the gameOver() method. There is a specific state that the game has to be in to check if the game is over or not, and that is: Are there any more legal choices for the player to make? Therefore creating this method was as simple as checking if number list had any numbers had divisors in the rest of the list.

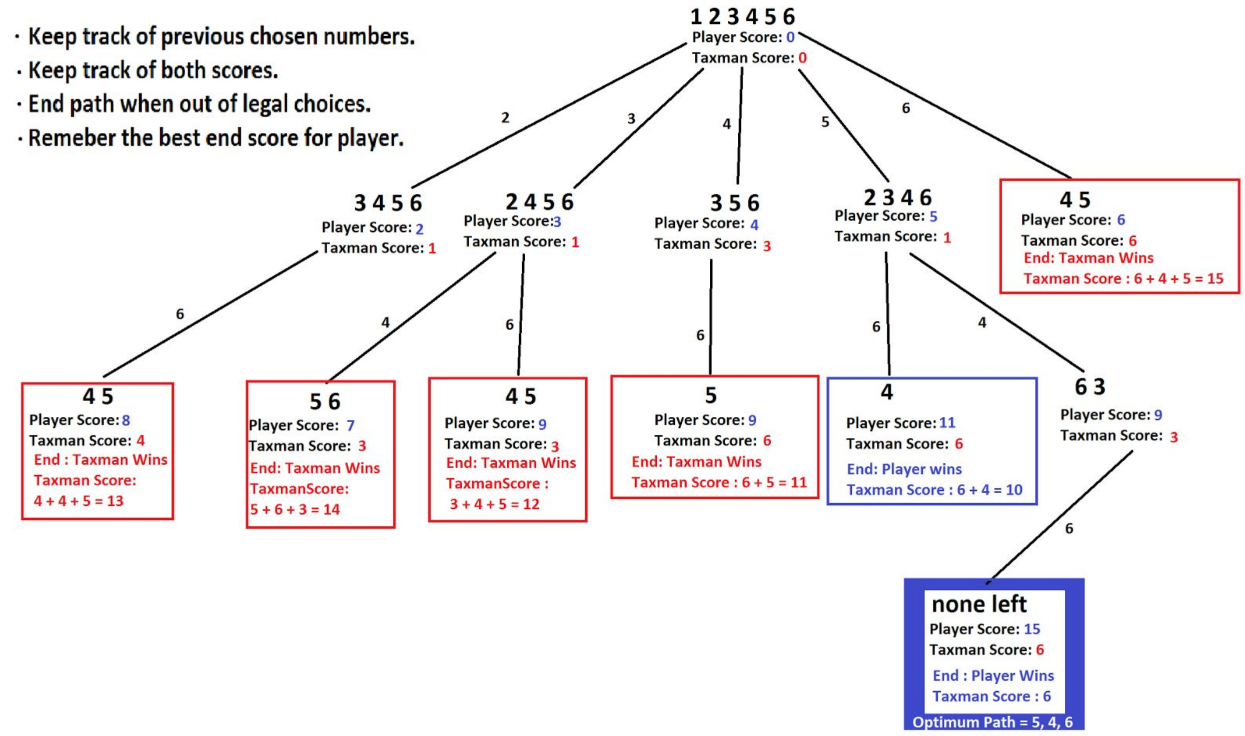
Naive

The Naive Strategy was a good way to get started on the understanding of choice optimization algorithms. This strategy had quite a simple idea which is 'pick the highest legal number'. Whilst this isn't necessarily an effecting class, it is nonetheless a simple autopilot which can sometimes yield effective results. The obvious way to make this class was to compare all items in the inputted selection and return the highest legal choice. This was quite easy with the TreeSet data structure due to the ability to iterate through every item.

Greedy

The point of the greedy strategy is to receive the best possible immediate score. The method is not designed to look through future choices made that end up with the best outcome (like optimal). It is instead supposed to look directly into the next turn only and find out which choice will yield the best result. The best result is the largest score difference between the taxman and player in the favour of the player. The design of this class was as simple as figuring out what the two scores are if you pick a certain choice. Then going through every single choice and comparing them at the end to find the ultimate greedy choice.

Optimal
The point of the Optimal strategy was to search through every single choice path that a player could make with the inputted set of numbers. This is in order to find out which immediate selection will end up producing the highest winning margin for the player. This class was quite difficult as it involved recursion. The hardest part was figuring out how to keep the information which was necessary to find the best choice path. I created an image of a tree which described all possible paths if the upper limit was 6. This was extremely useful as it notified me with which information was essential to keep for each individual path.



I decided to go with the input of three variables into the recursive getBestTree() method:

- A TreeSet of Integers. This holds the selection of numbers available for this turns choice.
- A ChoiceTree object. This holds the current scores so far in this series of turns.
- An array of previous choices. This is what the program uses to calculate the best possible set of choices for the player.

With these three inputs it became much easier to test the state of a specific choice list. Those three inputs where all my program needed to compare choice paths and find the optimal path. The basecase for the method getBestTree was simply checking if the inputted information was at an end of a path or not. If it is, then the path is compared to the previous best path. If it is in fact better (yield a better winning margin) then it becomes the new best path. Otherwise the method keeps iterating through other options in the inputted numbers tree and recursively calls itself to check again if the path is over after the numbers have been modified according to the choice made. This methodology allows the algorithm to search through every single possible path.