BEFORE UPDATING:
-HAVE YOU CHECKED YOUR CODE?
-IS IT STABLE/FUNCTIONAL?
-INCLUDE WHAT CHANGES YOU MADE TO CHANGE ITS FUNCTION AS A COMMENT AT THE TOP BEFORE THE CODE

Notes - Game Code - Anda's Job: game logic & structure
18/05 - Basics, scanner/bufferedreader
24/05 - Change from hardcoding to classes, only have names done right now
01/06 - Refined classes and functions, now working on playing certain cards and their functions
04/06 - Finished game structure/progression - game can now start and end without any hiccups
05/06 - Game logic finalized. Progression working and card logic works. Now need to program AI
06/06 - GAME LOGIC FINISHED

GAME CODE

```
import java.io.*;
import java.util.*;

class Main {
  private static BufferedReader x = new BufferedReader(new InputStreamReader(System.in));
  private static ArrayList <Player> currentPlayers;
  private static ArrayList <String> names = new ArrayList <String> (Arrays.asList("Dr. FeelsGood", "Duke Of Prance", "Elocakes", "iAmanda", "ChaosFlame", "StacyLynn", "Tian Bo Guang", "Hook&Pull", "Linghu Chong", "East the INVINCIBLE", "Brazil", "Nigeria", "ayyylmao", "xxIslesOfSolitudexx", "Ren the UNSTOPPABLE", "United", "RillKate"));

  public static void main(String[] args) throws IOException{
    //INTRO MESSAGE (MENU)
   System.out.println("Welcome to Love Letter!\nWhat is your name?");
   currentPlayers = new ArrayList<>(Arrays.asList(new Player(x.readLine(), true), new Player(aiNames(), false), new Player(aiNames(), false), new Player(aiNames(), false)));
   System.out.println("Welcome " + currentPlayers.get(0) + "! Your opponents are " + currentPlayers.get(1) +" (AI 1), " + currentPlayers.get(2) + " (AI 2), and " + currentPlayers.get(3) +" (AI 3)!");

    //Create the master deck and set up the game by removing a random card (to make it harder to deduce)
    MasterDeck deck  = new MasterDeck();
    int leftover = deck.deal();
    //Note: the leftover card is used when someone doesn't have a card at the last turn due to prince, and should obtain another card (but there are no cards left in the deck), so the leftover is given to them

    // //Deal cards to each player
```

```
    int add;
    for(int i = 0 ; i < currentPlayers.size(); i++){
      add = deck.deal();
      currentPlayers.get(i).dealTo(add);
    }

    //Game
    boolean currentTie = false;
    //Game plays while there are still cards in the deck and there is more than 1 player standing
    while(currentPlayers.size() > 1 || deck.remaining() > 0){
      for(int i = 0; i < currentPlayers.size(); i++){

        //Will reset the counter if there more players to continue to do their turn
        if(i==currentPlayers.size())
          i=0;

        //Saves value of which player is currently playing, and what to deal to the player in order to start their turn, add is the card that the player will get at the start of their turn
        int m = i;
        add = deck.deal();

        //Player's turn
        boolean isPlayer = currentPlayers.get(i).isPlayer(); //This figures out if the current player is actually a real player or an AI
        boolean stepBackCounter = turn(m, add, deck, leftover, isPlayer); //forwards the following: who is playing?, what card to deal to the player?, what's the state of the deck (used for a certain tye of card)?, what is the leftover card (mentioned above, for a corner case)?, is this a real player or AI?

        //If there was a player that was eliminated, stepback the counter so the turns progress in the correct order, but only if the eliminated player's position is before the current player's position.
        if(stepBackCounter && i!=0){
          i--;
        }

        //Win case (main), when there is only 1 player left
        if (currentPlayers.size() == 1){
          System.out.println("\n" + currentPlayers.get(i) + " has won!");
          currentPlayers.clear();
          deck.clear();
          break;
        }
```

```
        //Tie case #1, more than 1 player remains but no cards left
        else if(currentPlayers.size() > 1 && deck.remaining() == 0){
          currentTie = true;
          break;
        }
      }
      //When there is a Tie (Tie case #1), compare the cards of the remaining players and
break the tie
      if(currentTie){
        //compare current people and their cards
        int m = 0;
        int temp = 0;
        boolean tie = false;
        //Compare cards to find either a win or a tie
        for(int i = 0; i < currentPlayers.size(); i++){
          if(currentPlayers.get(i).cardDisplay(0) > temp){
            m = i;
            temp = currentPlayers.get(i).cardDisplay(0);
            tie = false;
          }
          else if(currentPlayers.get(i).cardDisplay(0) == temp){
            tie = true;
          }
        }
        //Find winner's card and announce that they won with that card
        if(!tie){
          String winningCard;
          if(temp == 1)
            winningCard = "Guard";
          else if(temp == 2)
            winningCard = "Preist";
          else if(temp == 3)
            winningCard = "Baron";
          else if(temp == 4)
            winningCard = "Handmaid";
          else if(temp == 5)
            winningCard = "Prince";
          else if(temp == 6)
          winningCard = "King";
          else if(temp == 7)
            winningCard = "Countess";
          else
            winningCard = "Princess";
          System.out.println("\nThe winner is " + currentPlayers.get(m) + " with a " +
winningCard +"!");
```

```
        }
        //Game tie
        else{
          System.out.println("\nGame tie! No winner!");
        }
        break; //ends the game
      }
    }

  } //End of main method (the game execution)

  //RANDOM NAME GENERATOR FUNCTION
  static String aiNames(){
    //Generates random names for the AI based on a list of predetermined names
    int ranGenNames = (int)(Math.random()*names.size());
    String ai = names.get(ranGenNames);
    names.remove(ranGenNames);
    return ai;
  }

  //PLAYER'S TURN FUNCTION
  static boolean turn(int m, int add, MasterDeck deck, int leftover, boolean isplayer) throws
IOException{
    boolean stepBack = false;

    //Deal card to player and make player targetable again
    currentPlayers.get(m).targetable();
    currentPlayers.get(m).dealTo(add);

    //Display player's cards, only if player is a real player
    if(isplayer){
      System.out.println("\n" + currentPlayers.get(m).toString() + ", you have: ");
      for(int i = 0; i < currentPlayers.get(m).numCards(); i++){
        System.out.print(currentPlayers.get(m).cardDisplay(i) + " ");
      }
    }

    //Countess Check, countess is force used (regardless of player or AI)
    if(currentPlayers.get(m).cardDisplay(0) == 7 && (currentPlayers.get(m).cardDisplay(1) ==
5 || currentPlayers.get(m).cardDisplay(1) == 6)){
      currentPlayers.get(m).useCard(0);
      if(isplayer){
        System.out.println("\nThe countess was used because you had the King/Prince.");
      }
      else{
```

```
        System.out.println("\n" + currentPlayers.get(m).toString() + " used a countess.");
      }
    }
    //Countess Check, countess is force used (regardless of player or AI)
    else if(currentPlayers.get(m).cardDisplay(1) == 7 &&
(currentPlayers.get(m).cardDisplay(0) == 5 || currentPlayers.get(m).cardDisplay(0) == 6)){
      currentPlayers.get(m).useCard(1);
      if(isplayer){
        System.out.println("\nThe countess was used because you had the King/Prince.");
      }
      else{
        System.out.println("\n" + currentPlayers.get(m).toString() + " used a countess.");
      }
    }

    //If countess isn't forced played, turn goes as normal.
    else{
      int use;
      //Real player: get's their turn by displaying what they have and what they want to use
for their turn
      if(isplayer){
        System.out.println("\nPick a card to use!");
        //Input player's choice of card to do
        use = currentPlayers.get(m).cardDisplay(Integer.parseInt(x.readLine()));
        for(int i = 0; i < currentPlayers.get(m).numCards(); i++){
          if(use==currentPlayers.get(m).cardDisplay(i)){
            currentPlayers.get(m).useCard(i);
            break;
          }
        }
      }
      //AI: they just generate a random card to use for their turn
      else{
        int carduse = (int)(Math.random()*2);
        use = currentPlayers.get(m).cardDisplay(carduse);
        currentPlayers.get(m).useCard(carduse);
      }

      //Different types of cards. Note that countess (#7) was already covered in the
beginning
      if(use==1){
        if(isplayer == false){
          System.out.println("\n" + currentPlayers.get(m).toString() + " used a guard.");
        }
        stepBack = guard(m, isplayer); //Do guard function
```

```
    }

    else if(use==2){
      if(isplayer == false){
        System.out.println("\n" + currentPlayers.get(m).toString() + " used a preist.");
      }
      preist(m, isplayer); //Do preist function
    }

    else if(use==3){
      if(isplayer == false){
        System.out.println("\n" + currentPlayers.get(m).toString() + " used a baron.");
      }
      stepBack = baron(m, isplayer); //Do baron function
    }

    else if(use==4){
      if(isplayer == false){
        System.out.println("\n" + currentPlayers.get(m).toString() + " used a handmaid.");
      }
      handmaid(m); //Do handmaid function
    }

    else if(use==5){
      if(isplayer == false){
        System.out.println("\n" + currentPlayers.get(m).toString() + " used a prince.");
      }
      stepBack = prince(m, isplayer); //Do prince function

      //Replenishes target's card up to one (what they should have when it's not their
turn), as they were force discarded
      for(int i = 0; i < currentPlayers.size(); i ++){
        if(deck.remaining() > 1 && currentPlayers.get(i).numCards() < 1){
          int addLeftover = deck.deal();
          currentPlayers.get(i).dealTo(addLeftover);
        }
        else if(deck.remaining() == 0 && currentPlayers.get(i).numCards() == 0){
          currentPlayers.get(i).dealTo(leftover);
        }
      }
    }

    else if(use==6){
      if(isplayer == false){
        System.out.println("\n" + currentPlayers.get(m).toString() + " used a king.");
```

```
        }
        king(m, isplayer); //Do king function
      }

      else if(use == 7){
        if(isplayer == false){
          System.out.println("\n" + currentPlayers.get(m).toString() + " used a countess.");
        }
        //Countess function was already covered in the beginning, if it gets used here, its
simply discarded
      }

      else{
        if(isplayer == false){
          System.out.println("\n" + currentPlayers.get(m).toString() + " used a princess.");
        }
        stepBack = princess(m); //Do princess function
      }
    }
    return stepBack; //If a player was eliminated, this fixes the counter so that the turns
are still in order
  }

  //CARD TYPE FUNCTIONS
  static boolean guard(int m, boolean isplayer) throws IOException{
    int target = targetPlayer(m, isplayer);
    boolean eliminatedPlayer = false;
    //Guard: if there is a target, choose a target and guess their card. If correct, they are
eliminated.
    //If no target available
    if(target == 1337){
      if(isplayer){
        System.out.println("No target available! Discard card.");
      }
      else{
        System.out.println(currentPlayers.get(m).toString() + " discards their card. Nothing
happens (no available targets).");
      }
    }
    //If a real player
    else if(isplayer & target != 1337){
      System.out.println("Pick a card to guess: Preist, Baron, Handmaid, Prince, King,
Countess, Princess");
      int guess = Integer.parseInt(x.readLine());
      if(guess == currentPlayers.get(target).cardDisplay(0)){
```

```
        currentPlayers.remove(target);
        System.out.println("You guessed correctly!");
        if(target < m){
          eliminatedPlayer = true;
        }
      }
    }
    //If an AI
    else{
      int guess = (int)(Math.random()*7);
      if(guess == currentPlayers.get(target).cardDisplay(0)){
        System.out.println(currentPlayers.get(m).toString() + " guessed correctly!");
        currentPlayers.remove(target);
        if(target < m){
          eliminatedPlayer = true;
        }
      }
    }
    return eliminatedPlayer;
  }

  static void preist(int m, boolean isplayer)throws IOException{
    int target = targetPlayer(m, isplayer);
    //Preist: if there is a target, choose a target and look at what they have in their hand
(no other player knows)
    //If no target available
    if(target == 1337){
      if(isplayer){
        System.out.println("No target available! Discard card.");
      }
      else{
        System.out.println(currentPlayers.get(m).toString() + " discards their card. Nothing
happens (no available targets).");
      }
    }
    //If a real player
    else if(isplayer && target != 1337){
        System.out.println(currentPlayers.get(target) + " has " +
currentPlayers.get(target).cardDisplay(0) + ".");
    }
    //If an AI
    else{
      //AI will use prince, but doesn't get the information to use in the next rounds, so it
acts as if the card was discarded
    }
```

```
  }

  static boolean baron(int m, boolean isplayer)throws IOException{
    int target = targetPlayer(m, isplayer);
    boolean eliminatedPlayer = false;
    //Baron: if there is a target available, choose a target and compare cards. The one with the lower value card is eliminated, otherwise if a tie, nothing happens
    //If no target available
    if(target == 1337){
      if(isplayer){
        System.out.println("No target available! Discard card.");
      }
      else{
        System.out.println(currentPlayers.get(m).toString() + " discards their card. Nothing happens (no available targets).");
      }
    }
    //Comparison if there is a available target
    else{
      if(currentPlayers.get(m).cardDisplay(0) > currentPlayers.get(target).cardDisplay(0)){
        System.out.println(currentPlayers.get(target) + " was eliminated!");
        currentPlayers.remove(target);
        if(target < m){
          eliminatedPlayer = true;
        }
      }
      else if(currentPlayers.get(m).cardDisplay(0) < currentPlayers.get(target).cardDisplay(0)){
        System.out.println(currentPlayers.get(m) + " was eliminated!");
        currentPlayers.remove(m);
        if(target < m){
          eliminatedPlayer = true;
        }
      }
      else{
        System.out.println("Tie! No one was eliminated!");
      }
    }
    return eliminatedPlayer;
  }

  static void handmaid(int m){
    //Handmaid: gives invulnerability, cannot be affected by other cards
    currentPlayers.get(m).handmaid();
  }
```

```
  static boolean prince(int m, boolean isplayer)throws IOException{
    //Prince: picks a target (can include themselves), and force discards their card, and
replenishes target with a new card, unless if the discarded card was a princess
    int target = targetPlayer(currentPlayers.get(m).toString(), isplayer);
    boolean eliminatedPlayer = false;
    int forceDiscard = currentPlayers.get(target).cardDisplay(0);
    currentPlayers.get(target).useCard(0);
    //If discarded card was a princess, that player is eliminated
    if(forceDiscard == 8){
      System.out.println(currentPlayers.get(target) + " was eliminated!");
      currentPlayers.remove(target);
      if(target < m){
        eliminatedPlayer = true;
      }
    }
    return eliminatedPlayer;
  }

  static void king(int m, boolean isplayer)throws IOException{
    int target = targetPlayer(m, isplayer);
    //King: if there are available targets, pick one and swap cards with them
    //If no targets available
    if(target == 1337){
      if(isplayer){
        System.out.println("No target available! Discard card.");
      }
      else{
        System.out.println(currentPlayers.get(m).toString() + " discards their card. Nothing
happens (no available targets).");
      }
    }
    //Else, will swap cards
    else{
      int tempPlayer = currentPlayers.get(m).cardDisplay(0);
      int tempTarget = currentPlayers.get(target).cardDisplay(0);
      currentPlayers.get(m).useCard(0);
      currentPlayers.get(target).useCard(0);
      currentPlayers.get(m).dealTo(tempTarget);
      currentPlayers.get(target).dealTo(tempPlayer);
    }
  }

  static boolean princess(int m){
```

```
    //Princess: When discarded, player is eliminated from play (you shouldn't have done
that!!)
    System.out.println(currentPlayers.get(m) + " eliminated themselves.");
    currentPlayers.remove(m);
    if(m == 0){ //If player #0 eliminates themselves, the counter should stay at 0 (because
players get shifted forwards)
      return false;
    }
    else{ //Otherwise shift the counter
      return true;
    }
  }

  //TARGETING PLAYER SYSTEM
  static int targetPlayer(int m, boolean isplayer) throws IOException{
    //If it is a real player's turn: display the available targets
    if(isplayer){
      System.out.print("Pick a target: ");
      int numTargets = 0;
      for(int i = 0; i < currentPlayers.size(); i++){
        if(i!=m && currentPlayers.get(i).target()){
          System.out.print(currentPlayers.get(i) + ", ");
          numTargets++;
        }
      }
      System.out.println(); //Spacer
      if(numTargets > 0){ //Searches for which player they selected to use the card on
        String target = x.readLine();
        int i = 0;
        for(; i < currentPlayers.size(); ++i){
          if(target.equals(currentPlayers.get(i).toString())){
            break;
          }
        }
        return i;
      }
      else{ //If 0 targets, their turn is void (they just simply discard their card)
        return 1337;
      }
    }
    //If an AI, randomly slects a target
    else{
      int numInvul = 0;
      int targetOfAI;
      for(int i = 0; i < currentPlayers.size(); i++){
```

```
        if(!(currentPlayers.get(i).target()) ){
          numInvul++; //Keeps track of how many targets are not available
        }
      }
      if(currentPlayers.size()-1 == numInvul){  //If 0 available targets, then their turn is
void
          return 1337;
      }
      //Otherwise, if targets available, search for which target they selected
      else{
        while(true){
          targetOfAI = (int)(Math.random()*currentPlayers.size());
          if(targetOfAI!=m && currentPlayers.get(targetOfAI).target()){
            break;
          }
        }
        String target = currentPlayers.get(targetOfAI).toString();
        int k = 0;
        for(; k < currentPlayers.size(); k++){
          if(target.equals(currentPlayers.get(k).toString())){
            System.out.println(currentPlayers.get(m).toString() + " targets " +
currentPlayers.get(k).toString());
            break;
          }
        }
        return k; //Returns the index of that target
      }
    }
  }

  static int targetPlayer(String curPl, boolean isplayer) throws IOException{
    //In this case, there is always 1 available target, so there is no need to search for if
there are any targets available (you can target yourself with the prince)
    //If it is a real player's turn, display the cards they have and ask for an input for
their choice (same as above)
    if(isplayer){
      System.out.print("Pick a target: ");
      for(int i = 0; i < currentPlayers.size(); i++){
        if(currentPlayers.get(i).target()){
          System.out.print(currentPlayers.get(i) + ", ");
        }
      }
      System.out.println();
      String target = x.readLine();
      int i = 0;
```

```
      for(; i < currentPlayers.size(); i++){
        if(target.equals(currentPlayers.get(i).toString())){
          break;
        }
      }
      return i;
    }
    //If player is an AI, make them randomly find a target (same as above)
    else{
      int targetOfAI;
      while(true){
        targetOfAI = (int)(Math.random()*currentPlayers.size());
        if(currentPlayers.get(targetOfAI).target()){
          break;
        }
      }
      String target = currentPlayers.get(targetOfAI).toString();
      int k = 0;
      for(; k < currentPlayers.size(); k++){
        if(target.equals(currentPlayers.get(k).toString())){
          System.out.println(curPl + " targets " + currentPlayers.get(k).toString());
          break;
        }
      }
      return k;
    }
  }

} //End of Main class

class Player {
  //Every player has the following properties: name (String), hand of cards (ArrayList), a status of targetability (boolean), and a real/AI player status (boolean)
  private String nameP;
  private ArrayList <Integer> handP;
  private boolean targetable = true;
  private boolean isPlayer;

  //Player enters their name - from buffered reader, and also get a new hand or AI get their name randomly generated and also get a new hand, and also get assigned a real/AI player status
  Player(String Name, boolean player){
    nameP = Name;
    handP = new ArrayList <Integer>();
    isPlayer = player;
```

```
}

//Returns the string value instead of the address of the object.
public String toString(){
  return nameP;
}

//Adds the randomly gen'd card to player's hand
void dealTo(int add){
  handP.add(add);
}

//Finds the amount of cards in the player's hand
int numCards(){
  int siz = handP.size();
  return siz;
}

//Returns the value of the card that the player has at a certain index
int cardDisplay(int n){
  int nTh = handP.get(n);
  return nTh;
}

//Uses up the player's card by discarding it
void useCard(int i){
  handP.remove(i);
}

//Makes the player targetable once again (on start of their turn)
void targetable(){
  targetable = true;
}

//Returns the player's targetability status
boolean target(){
  return targetable;
}

//Makes the player not targetable
void handmaid(){
  targetable = false;
}

//Returns player's real/AI player status
```

```
  boolean isPlayer(){
    return isPlayer;
  }
} //End of Player class

class MasterDeck{
  private ArrayList <Integer> masterDeck = new ArrayList <Integer>(Arrays.asList(1, 1, 1, 1,
1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 7, 8)); //This is the custom deck
  int deckLength = masterDeck.size();
  //Core functions to randomly generate numbers and deal the cards to the players
  int random(){
    return (int)(Math.random()*masterDeck.size());
  }
  void remove(){
    int random = random();
    masterDeck.remove(random);
  }
  void remove(int in){
    masterDeck.remove(in);
  }
  int deal(){
    int idx = random();
    int val = masterDeck.get(idx);
    remove(idx);
    return val;
  }

  //Finds the current amount of cards remaining in the deck
  int remaining(){
    int i = masterDeck.size();
    return i;
  }
  void clear(){ //Clears the entire deck
    masterDeck.clear();
  }
} //End of MasterDeck class
```

Added music player
Added guard's popup options
Added most other card options
Added CPU Targeting for card options
Music Swap option added
All card functions completed
Turn completed
Game Runs
Goodbye
Exception errors removed
Discard list added
Finished

GUI CODE

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.*;
import sun.audio.*;
import java.io.*;
import java.lang.*;
import java.util.concurrent.TimeUnit;
public class Love_Letter extends JFrame implements ActionListener{
     //declare all variables to be used
          private static BufferedReader x = new BufferedReader(new
InputStreamReader(System.in));
          private static ArrayList <Player> currentPlayers;
          private static ArrayList <String> names = new ArrayList <String>
(Arrays.asList("Dr. FeelsGood", "Duke Of Prance", "Elocakes", "iAmanda",
"ChaosFlame", "StacyLynn", "Tian Bo Guang", "Hook&Pull", "Linghu Chong",
"East the INVINCIBLE", "Brazil", "Nigeria", "ayyylmao",
"Xx_IslesOfSolitude_xX", "Ren the UNSTOPPABLE", "United", "RillKate"));

          private JButton buttons[] = new JButton [4];      //number of
buttons
          private String name [] = {"Instructions", "Start Turn", "Log",
"Music"};   //button text
          public static JLabel
player[],cpu1[],cpu2[],cpu3[],pw,c1w,c2w,c3w,result,plh,hnd1,hnd2,hnd3,dis
card[];//variables
          private Container c = getContentPane ();
          private JPanel buttonPanel, cardP,scoreP;
          public static String log="",allLog="";
          private static boolean startTurn=false;
          public static ArrayList discards = new ArrayList();

public Love_Letter()throws IOException{
     //INTRO MESSAGE (MENU)
```

```
    String Uname = JOptionPane.showInputDialog(null, "Welcome to Love
Letter!\nWhat is your name?","uvuvwevwevwe onyetenyevwe ugwemubwem
ossas",JOptionPane.INFORMATION_MESSAGE);
     if(Uname==null||Uname.equals("")){
          JOptionPane.showMessageDialog(null,"Since you so eloquently
named yourself o \"Beautifully\", we shall spare the other player\'s form
being in awe of your \"Magnificent\" name by replacing it
with\"Kkwazzawazzakkwaquikkwalaquaza ?'* Zzabolazza\"","Nice
one",JOptionPane.ERROR_MESSAGE);
          Uname="Kkwazzawazzakkwaquikkwalaquaza ?\'* Zzabolazza";
     }
     currentPlayers = new ArrayList<>(Arrays.asList(new Player(Uname,
true), new Player(aiNames(), false), new Player(aiNames(), false), new
Player(aiNames(), false)));
    JOptionPane.showMessageDialog(null,"Welcome " + currentPlayers.get(0)
+ "! Your opponents are " + currentPlayers.get(1) +" (AI 1), " +
currentPlayers.get(2) + " (AI 2), and " + currentPlayers.get(3) +" (AI
3)!");

    //Create the master deck and set up the game by removing a random card
(to make it harder to deduce)
    MasterDeck deck  = new MasterDeck();
    int leftover = deck.deal();
    //Note: the leftover card is used when someone doesn't have a card at
the last turn due to prince, and should obtain another card (but there are
no cards left in the deck), so the leftover is given to them

    // //Deal cards to each player
    int add;
    for(int i = 0 ; i < currentPlayers.size(); i++){
      add = deck.deal();
      currentPlayers.get(i).dealTo(add);
    }
     //Set up images for game window
          player= new JLabel[5];   //player card hands
     cpu1= new JLabel[5];
     cpu2= new JLabel[5];
     cpu3= new JLabel[5];

     plh= new JLabel(currentPlayers.get(0).toString()); //name tags
          hnd1= new JLabel(currentPlayers.get(1).toString());
          hnd2= new JLabel(currentPlayers.get(2).toString());
          hnd3= new JLabel(currentPlayers.get(3).toString());


     buttonPanel = new JPanel();   //buttons
         buttonPanel.setLayout(
```

```
          new GridLayout (1, buttons.length));

     c = getContentPane ();
     setContentPane(c);

     for (int i=0; i<2; i++) {
               cpu1[i]=new JLabel(new ImageIcon("CardBack.jpg"));
//arrays of cpu1 hand
               cpu2[i]=new JLabel(new ImageIcon("CardBack.jpg"));
//arrays of cpu2 hand
               cpu3[i]=new JLabel(new ImageIcon("CardBack.jpg"));
//arrays of cpu3 hand
               player[i]=new JLabel(new ImageIcon("CardBack.jpg"));
//arrays of player hand
          }

          cardP=new JPanel();
         cardP.setLayout(new GridLayout(2,4));
          cardP.add(hnd1);
          for (int i=0; i<2; i++) {   //display cards cpu1
               cardP.add(cpu1[i]);
          }
          cardP.add(hnd2);
          for (int i=0; i<2; i++) {   //display cards player
               cardP.add(cpu2[i]);
          }
          cardP.add(plh);
          for (int i=0; i<2; i++) {   //display cards cpu12
               cardP.add(player[i]);
          }

          cardP.add(hnd3);
          for (int i=0; i<2; i++) {   //display cards cpu3
               cardP.add(cpu3[i]);
          }

     for (int i = 0 ; i < buttons.length ; i++)
     {
         buttons [i] = new JButton (name [i]);
         buttonPanel.add (buttons [i]);
         buttons[i].addActionListener( this );
     }
          pw= new JLabel("");   //header text setup
          c1w= new JLabel("");
          c2w= new JLabel("");
          c3w= new JLabel("");
          result= new JLabel("");
```

```
     scoreP=new JPanel();
     scoreP.add(result);

     c.add (scoreP, BorderLayout.NORTH);   //JLable placement formating
    c.add (cardP, BorderLayout.CENTER);
     c.add (buttonPanel, BorderLayout.SOUTH);

     setSize (900, 900);   //window dimentions
     show ();
     //Game
    boolean currentTie = false;
    //Game plays while there are still cards in the deck and there is more
than 1 player standing
     while((currentPlayers.size() > 1 || deck.remaining() > 0)){
          System.out.print("");
          if(startTurn==true){
               int  i=0;
               for(; i < currentPlayers.size(); i++){

                    //Will reset the counter if there more players to
continue to do their turn
                    if(i==currentPlayers.size())
                         i=0;

                    //Saves value of which player is currently playing,
and what to deal to the player in order to start their turn, add is the
card that the player will get at the start of their turn
                    int m = i;
                    add = deck.deal();

                    //Player's turn
                    boolean isPlayer = currentPlayers.get(i).isPlayer();
//This figures out if the current player is actually a real player or an
AI
                    boolean stepBackCounter = turn(m, add, deck, leftover,
isPlayer); //forwards the following: who is playing?, what card to deal to
the player?, what's the state of the deck (used for a certain tye of
card)?, what is the leftover card (mentioned above, for a corner case)?,
is this a real player or AI?
                    result.setText("List of discarded cards:
"+discards.toString()); //displays discard arraylist

                    //If there was a player that was eliminated, stepback
the counter so the turns progress in the correct order, but only if the
eliminated player's position is before the current player's position.
                    if(stepBackCounter && i!=0){
```

```
                    i--;
                }

                //Win case (main), when there is only 1 player left
                if (currentPlayers.size() == 1){
                    System.out.println("\n" + currentPlayers.get(i) +
" has won!");

JOptionPane.showMessageDialog(null,currentPlayers.get(i) + " has
won!","WINNER WINNER CHICKEN DINNER",JOptionPane.WARNING_MESSAGE);
                    currentPlayers.clear();
                    deck.clear();
                    break;
                }
                //Tie case #1, more than 1 player remains but no cards
left
                else if(currentPlayers.size() > 1 && deck.remaining()
== 0){
                    currentTie = true;
                    break;
                }
                if(i!=currentPlayers.size()-1){
                    log+="\n";
                }
            }
            log+="_____________________________\n";
            JOptionPane.showMessageDialog(null,
                log,"Battle.log: Adventure Quest",
                JOptionPane.QUESTION_MESSAGE);
            allLog+=log;//add log to master log
            log="";//reset masterlog
            if(!currentTie&&currentPlayers.size() != 1){
                System.out.println("\n\nPress \"Start Turn\" to
continue.");
                startTurn =false;
            }
            else{
                //When there is a Tie (Tie case #1), compare the cards
of the remaining players and break the tie
                if(currentTie){
                    //compare current people and their cards
                    int m = 0;
                    int temp = 0;
                    boolean tie = false;

                    //Compare cards to find either a win or a tie
                    for(int j = 0; j < currentPlayers.size(); j++){
```

```
                              if(currentPlayers.get(j).cardDisplay(0) >
temp){
                              m = j;
                              temp = currentPlayers.get(j).cardDisplay(0);
                              tie = false;
                              }
                              else if(currentPlayers.get(j).cardDisplay(0)
== temp){
                                   tie = true;
                              }
                         }
                         //Find winner's card and announce that they won
with that card
                         if(!tie){
                              String winningCard;
                              if(temp == 1)
                                   winningCard = "Guard";
                              else if(temp == 2)
                                   winningCard = "priest";
                              else if(temp == 3)
                                   winningCard = "Baron";
                              else if(temp == 4)
                                   winningCard = "Handmaid";
                              else if(temp == 5)
                                   winningCard = "Prince";
                              else if(temp == 6)
                                   winningCard = "King";
                              else if(temp == 7)
                                   winningCard = "Countess";
                              else
                                   winningCard = "Princess";
                              System.out.println("\nThe winner is " +
currentPlayers.get(m) + " with a " + winningCard +"!");
                              log+="\nThe winner is " +
currentPlayers.get(m) + " with a " + winningCard +"!";
                              JOptionPane.showMessageDialog(null,"The
winner is " + currentPlayers.get(m) + " with a " + winningCard
+"!","WINNER WINNER CHICKEN DINNER",JOptionPane.WARNING_MESSAGE);
                              break;//ends the game
                         }

                         //Game tie
                         else{
                              System.out.println("\nGame tie! No
winner!");
                              log+="\nGame tie! No winner!";
```

```
                              JOptionPane.showMessageDialog(null,"Game
tie! No winner!","...DISAPOINTED!!!!!1!",JOptionPane.WARNING_MESSAGE);
                         }
                         break; //ends the game
                    }
               }
          }
     }
     try{// thread to sleep for 1000 milliseconds
        Thread.sleep(1000);
          }
     catch (Exception e) {
            System.out.println(e);
        }
     JOptionPane.showMessageDialog(null,"The world, bound by
chains,\nshall turn to dust \nYet the cycle knows no end\nand all shall be
born anew.\nFor now, all who are lost shall\nvanish into the boundless
void.","HaVe A nIcE dAy...",JOptionPane.WARNING_MESSAGE);
     System.exit (0);
}

public void actionPerformed( ActionEvent e ){
     if ( e.getSource() == buttons[ 0 ] ){
          //Show    instructions popup when pressed
          JOptionPane.showMessageDialog(null,
               "Your goal is to get your love letter into Princess
Annette's hands while deflecting the letters from competing suitors.
\nFrom a deck with only sixteen cards, each player starts with only one
card in hand; one card is removed from play. \nOn a turn, you draw one
card, and play one card, trying to expose others and knock them from the
game. \nPowerful cards lead to early gains, but make you a target. \nRely
on weaker cards for too long, however, and your letter may be tossed in
the fire!",
               "Instructions",     JOptionPane.WARNING_MESSAGE);
          //Show the popup for card types and explanation
          JOptionPane.showMessageDialog(null,
               "Strength - Type: #of, Description \n1 - Guard (5): Name a
non-Guard card, if target player has card, he/she is out of the round.\n2
- Priest (2): Look at a player's hand.\n3 - Baron (2): Compare hands,
lower rank is out of the round.\n4 - Handmaid (2): Ignore all effects
targeting you until your next turn.\n5 - Prince (2): Target player (can
include yourself) discards hand, draws new card.\n6 - King (1): Change
hands with target player.\n7 - Countess (1): If you have this card and the
King/Prince in your hand, you must discard this card.\n8 - Princess (1):
If you discard this card, you are out of the round.",
               "Card Values",JOptionPane.ERROR_MESSAGE);
     }
```

```
     else if ( e.getSource() == buttons[ 1 ] ){
          startTurn=true;
     }
     else if ( e.getSource() == buttons[ 2 ] ){
          //prints master log when pressed
          JOptionPane.showMessageDialog(null,
               allLog,"Battle.log: Adventure Quest",
               JOptionPane.QUESTION_MESSAGE);
     }
    else if ( e.getSource() == buttons[ 3 ] ){
          //gets random music file name folder
          String song[]=new String []
{"HEY","Shia","Edge","Rickroll","Flyers ","September","Big","Awaken","Gas
Gas Gas","soup","SMO","Killer","You Say Run"};
          int rng=((int)(Math.random()*song.length));
          String filename=(song[rng]+".wav");
          System.out.println(song[rng]);
          InputStream music;//creates Inputstream for music
          try{
               music = new FileInputStream(new File(filename));//gets
music file
               AudioStream audio= new AudioStream(music);//creates
audiostream of music file
               AudioPlayer.player.start(audio);//plays audio
               //confirmation that song is acceptable
               int sng = JOptionPane.showConfirmDialog(null,
                    "Are you fine with this song?\n{You won't be able to
stop it until it ends naturally!}?",
                    "Do you have good taste?",JOptionPane.YES_NO_OPTION);

                    if(sng!=0){
                         AudioPlayer.player.stop(audio);    //stops audio
if user inputs that song is not acceptable
                    }
               }
          catch(Exception g){//displays error message if music is not
found/corrupted
               JOptionPane.showMessageDialog(null,"ERROR");
          }
    }
}
//RANDOM NAME GENERATOR FUNCTION
static String aiNames(){
    //Generates random names for the AI based on a list of predetermined
names
    int ranGenNames = (int)(Math.random()*names.size());
    String ai = names.get(ranGenNames);
```

```
    names.remove(ranGenNames);
    return ai;
}

//PLAYER'S TURN FUNCTION
static boolean turn(int m, int add, MasterDeck deck, int leftover, boolean
isplayer) throws IOException{
     boolean stepBack = false;

    //Deal card to player and make player targetable again
    currentPlayers.get(m).targetable();
    currentPlayers.get(m).dealTo(add);

    //Display player's cards, only if player is a real player
    if(isplayer){
      System.out.println("\n" + currentPlayers.get(m).toString() + ", you
have: ");
      for(int i = 0; i < currentPlayers.get(m).numCards(); i++){
        System.out.print(currentPlayers.get(m).cardDisplay(i) + " ");
          //displays player's cards and covers all cpu cards if previously
revealed by priest
          player[i].setIcon(new
ImageIcon(currentPlayers.get(0).cardDisplay(i)+".jpg"));
          cpu1[i].setIcon(new ImageIcon("CardBack.jpg"));
          cpu2[i].setIcon(new ImageIcon("CardBack.jpg"));
          cpu3[i].setIcon(new ImageIcon("CardBack.jpg"));
      }
    }
    //Countess Check, countess is force used (regardless of player or AI)
    if(currentPlayers.get(m).cardDisplay(0) == 7 &&
(currentPlayers.get(m).cardDisplay(1) == 5 ||
currentPlayers.get(m).cardDisplay(1) == 6)){
          currentPlayers.get(m).useCard(0);
          if(isplayer){
               System.out.println("\nThe countess was used because you had
the King/Prince.");
               JOptionPane.showMessageDialog(null,"The countess was used
because you had the King/Prince.");
               log+=currentPlayers.get(m).toString() + " used a
countess.\n";//add to turn log
          }
          else{
               System.out.println("\n" + currentPlayers.get(m).toString()
+ " used a countess.");
               log+=currentPlayers.get(m).toString() + " used a
countess.\n";//add to turn log
          }
```

```
          discards.add(7);
     }
    //Countess Check, countess is force used (regardless of player or AI)
    else if(currentPlayers.get(m).cardDisplay(1) == 7 &&
(currentPlayers.get(m).cardDisplay(0) == 5 ||
currentPlayers.get(m).cardDisplay(0) == 6)){
          currentPlayers.get(m).useCard(1);
          if(isplayer){
               System.out.println("\nThe countess was used because you had
the King/Prince.");
               JOptionPane.showMessageDialog(null,"The countess was used
because you had the King/Prince.");
               log+=currentPlayers.get(m).toString() + " used a
countess.\n";//add to turn log
          }
          else{
               System.out.println("\n" + currentPlayers.get(m).toString()
+ " used a countess.");
               log+=currentPlayers.get(m).toString() + " used a
countess.\n";//add to turn log
          }
          discards.add(7);
    }
    //If countess isn't forced played, turn goes as normal.
    else{
          int use,do_it=-1;
          //Real player: get's their turn by displaying what they have and
what they want to use for their turn
          if(isplayer){
               while(do_it==-1){
                    Object[] options = {"Card 1","Card 2"};
                         do_it = JOptionPane.showOptionDialog(null,
                         "Which card will you play?",//body text
                         "The wheel of fate is turning - Heaven or Hell:
Let\'s Rock",//title text
                         JOptionPane.YES_NO_OPTION,//input type
                         JOptionPane.QUESTION_MESSAGE,//icon type
                         null, options, options[0]); //number of uptions
               }
               use=currentPlayers.get(m).cardDisplay(do_it);
               for(int i = 0; i < currentPlayers.get(m).numCards(); i++){
                    if(use==currentPlayers.get(m).cardDisplay(i)){
                         currentPlayers.get(m).useCard(i);
                         break;
                    }
               }
```

```
                player[do_it].setIcon(new ImageIcon("CardBack.jpg"));//flip
the used card
          }
          //AI: they just generate a random card to use for their turn
          else{
                int carduse = (int)(Math.random()*2);
                use = currentPlayers.get(m).cardDisplay(carduse);
                currentPlayers.get(m).useCard(carduse);
          }
          discards.add(use);//add used card to discard list
          //Different types of cards. Note that countess (#7) was already
covered in the beginning
          if(use==1){
                System.out.println("\n" + currentPlayers.get(m).toString()
+ " used a guard.");
                log+=currentPlayers.get(m).toString() + " used a guard.\n";
                stepBack = guard(m, isplayer); //Do guard function
          }
          else if(use==2){
                System.out.println("\n" + currentPlayers.get(m).toString()
+ " used a priest.");
                log+=currentPlayers.get(m).toString() + " used a
priest.\n";
                priest(m, isplayer); //Do priest function
          }
          else if(use==3){
                System.out.println("\n" + currentPlayers.get(m).toString()
+ " used a baron.");
                log+=currentPlayers.get(m).toString() + " used a baron.\n";
                stepBack = baron(m, isplayer); //Do baron function
          }
          else if(use==4){
                System.out.println("\n" + currentPlayers.get(m).toString()
+ " used a handmaid.");
                log+=currentPlayers.get(m).toString() + " used a
handmaid.\n";
                handmaid(m); //Do handmaid function
          }
          else if(use==5){
                System.out.println("\n" + currentPlayers.get(m).toString()
+ " used a prince.");
                log+=currentPlayers.get(m).toString() + " used a
prince.\n";
                stepBack = prince(m, isplayer); //Do prince function
                //Replenishes target's card up to one (what they should
have when it's not their turn), as they were force discarded
                for(int i = 0; i < currentPlayers.size(); i ++){
```

```
                    if(deck.remaining() > 1 &&
currentPlayers.get(i).numCards() < 1){
                    int addLeftover = deck.deal();
                    currentPlayers.get(i).dealTo(addLeftover);
                    }
                    else if(deck.remaining() == 0 &&
currentPlayers.get(i).numCards() < 0){ //CHANGED LAST MIN
                         currentPlayers.get(i).dealTo(leftover);
                    }
               }
          }
          else if(use==6){
               System.out.println("\n" + currentPlayers.get(m).toString()
+ " used a king.");
               log+=currentPlayers.get(m).toString() + " used a king.\n";
               king(m, isplayer); //Do king function
          }
          else if(use == 7){
               System.out.println("\n" + currentPlayers.get(m).toString()
+ " used a countess.");
               log+=currentPlayers.get(m).toString() + " used a
countess.\n";
               //Countess function was already covered in the beginning,
if it gets used here, its simply discarded
          }
          else{
               System.out.println("\n" + currentPlayers.get(m).toString()
+ " used a princess.");
               log+=currentPlayers.get(m).toString() + " used a
princess.\n";
               stepBack = princess(m); //Do princess function
          }
    }
    return stepBack; //If a player was eliminated, this fixes the counter
so that the turns are still in order
}

//CARD TYPE FUNCTIONS
static boolean guard(int m, boolean isplayer) throws IOException{
     int target = targetPlayer(m, isplayer);
    boolean eliminatedPlayer = false;
    //Guard: if there is a target, choose a target and guess their card.
If correct, they are eliminated.
    //If no target available
    if(target == 1337){
          if(isplayer){
               System.out.println("No target available! Discard card.");
```

```
                JOptionPane.showMessageDialog(null,"No target available!
Discard card.");
           }
           else{
                System.out.println(currentPlayers.get(m).toString() + "
discards their card. Nothing happens (no available targets).");
           }
     }

     //If a real player
     else if(isplayer & target != 1337){
      Object selectedValue=null;
      int gogo=1;
           String[] possibleValues = {"Priest", "Baron",
"Handmaid","Prince","King","Countess","Princess"};
           while(gogo==1){
                selectedValue = JOptionPane.showInputDialog(null,
                     "Choose one to Eliminate", "The Guard's Options",
                     JOptionPane.INFORMATION_MESSAGE, null,
                     possibleValues,
possibleValues[possibleValues.length-2]);
                //keep requesting input if canceled for somereason
                if(selectedValue!=null)
                     gogo=0;
           }
           int guess=0;
           String Convert=selectedValue.toString();//convert string options
into integers
           if(Convert.equals("Priest")){
                guess=2;
           }
           else if(Convert.equals("Baron")){
                guess=3;
           }
           else if(Convert.equals("Handmaid")){
                guess=4;
           }
           else if(Convert.equals("Prince")){
                guess=5;
           }
           else if(Convert.equals("King")){
                guess=6;
           }
           else if(Convert.equals("Countess")){
                guess=7;
           }
           else if(Convert.equals("Princess")){
```

```
                guess=8;
          }
          //if guess was correct then eliminate player
          if(guess == currentPlayers.get(target).cardDisplay(0)){
                log+=currentPlayers.get(target) + " was
eliminated!\n";//add to turn log
                currentPlayers.remove(target);
                System.out.println("You guessed correctly!");
                JOptionPane.showMessageDialog(null,
                     "You guessed correctly!",
                     "OHHHH SNIPED GET RKT M8",
                     JOptionPane.INFORMATION_MESSAGE);
                if(target < m){
                     eliminatedPlayer = true;
                }
          }
    }
    //If an AI
    else{
          int guess = (int)(Math.random()*7+2);
          if(guess == currentPlayers.get(target).cardDisplay(0)){
                log+=currentPlayers.get(target) + " was
eliminated!\n";//add to turn log
                System.out.println(currentPlayers.get(m).toString() + "
guessed correctly!");
                JOptionPane.showMessageDialog(null,
                     currentPlayers.get(target) + " was eliminated!",
                     "DEATH COMES", JOptionPane.WARNING_MESSAGE);
                currentPlayers.remove(target);
                if(target < m){
                     eliminatedPlayer = true;

                }
          }
    }
    return eliminatedPlayer;
  }

static void priest(int m, boolean isplayer)throws IOException{
    int target = targetPlayer(m, isplayer);
    //priest: if there is a target, choose a target and look at what they
have in their hand (no other player knows)
    //If no target available
    if(target == 1337){
          if(isplayer){
                System.out.println("No target available! Discard card");
```

```
                JOptionPane.showMessageDialog(null,"No target available!
Discard card.");
          }
          else{
                System.out.println(currentPlayers.get(m).toString() + "
discards their card. Nothing happens (no available targets).");
          }
    }

    //If a real player
    else if(isplayer && target != 1337){
        System.out.println(currentPlayers.get(target) + " has " +
currentPlayers.get(target).cardDisplay(0) + ".");
          log+=currentPlayers.get(target) + " has " +
currentPlayers.get(target).cardDisplay(0) + ".\n";//add to turn log
          if(target==1){
                cpu1[0].setIcon(new
ImageIcon(currentPlayers.get(1).cardDisplay(0)+".jpg"));
          }
          else if(target==2){
                cpu2[0].setIcon(new
ImageIcon(currentPlayers.get(2).cardDisplay(0)+".jpg"));
          }
          else if(target==3){
                cpu3[0].setIcon(new
ImageIcon(currentPlayers.get(3).cardDisplay(0)+".jpg"));
          }
     }
    //If an AI
    //else{
          //AI will use prince, but doesn't get the information to use in
the next rounds, so it acts as if the card was discarded
    //}
}

static boolean baron(int m, boolean isplayer)throws IOException{
     int target = targetPlayer(m, isplayer);
    boolean eliminatedPlayer = false;
    //Baron: if there is a target available, choose a target and compare
cards. The one with the lower value card is eliminated, otherwise if a
tie, nothing happens
    //If no target available
    if(target == 1337){
          if(isplayer){
                System.out.println("No target available! Discard card.");
                JOptionPane.showMessageDialog(null,"No target available!
Discard card.");
```

```
          }
          else{
               System.out.println(currentPlayers.get(m).toString() + "
discards their card. Nothing happens (no available targets).");
          }
     }
     //Comparison if there is a available target
     else{
          if(currentPlayers.get(m).cardDisplay(0) >
currentPlayers.get(target).cardDisplay(0)){
               System.out.println(currentPlayers.get(target) + " was
eliminated!");
               JOptionPane.showMessageDialog(null,
                    currentPlayers.get(target) + " was eliminated!",
                    "DEATH COMES", JOptionPane.WARNING_MESSAGE);
               log+=currentPlayers.get(target) + " was
eliminated!\n";//add to turn log
               currentPlayers.remove(target);
               if(target < m){
               eliminatedPlayer = true;//eliminate player
               }
          }
          else if(currentPlayers.get(m).cardDisplay(0) <
currentPlayers.get(target).cardDisplay(0)){
               System.out.println(currentPlayers.get(target) + " was
eliminated!");
               JOptionPane.showMessageDialog(null,
                    currentPlayers.get(m) + " was eliminated!",
                    "DEATH COMES", JOptionPane.WARNING_MESSAGE);
               log+=currentPlayers.get(m) + " was eliminated!\n";//add to
turn log
               currentPlayers.remove(m);
               if(target < m){
               eliminatedPlayer = true;//eliminate player
               }
          }
          else{
               System.out.println("Tie! No one was eliminated!");
               JOptionPane.showMessageDialog(null,
                    "Tie! No one was eliminated!",
                    "...How Disappointing",
                    JOptionPane.ERROR_MESSAGE);
               log+="Tie! No one was eliminated!\n";//add to turn log
          }
     }
     return eliminatedPlayer;
}
```

```
static void handmaid(int m){
    //Handmaid: gives invulnerability, cannot be affected by other cards
    currentPlayers.get(m).handmaid();
}

static boolean prince(int m, boolean isplayer)throws IOException{
    //Prince: picks a target (can include themselves), and force discards
their card, and replenishes target with a new card, unless if the
discarded card was a princess
    int target = targetPlayer(currentPlayers.get(m).toString(), isplayer);
    boolean eliminatedPlayer = false;
    int forceDiscard = currentPlayers.get(target).cardDisplay(0);
     discards.add(forceDiscard);
    currentPlayers.get(target).useCard(0);
    //If discarded card was a princess, that player is eliminated
    if(forceDiscard == 8){
          System.out.println(currentPlayers.get(target) + " was
eliminated!");
          JOptionPane.showMessageDialog(null,
               currentPlayers.get(target) + " was eliminated!",
               "DEATH COMES", JOptionPane.WARNING_MESSAGE);
          log+=currentPlayers.get(target) + " was eliminated!\n";//add to
turn log
          currentPlayers.remove(target);
          if(target < m){
               eliminatedPlayer = true;//eliminate player
          }
    }
    return eliminatedPlayer;
}

static void king(int m, boolean isplayer)throws IOException{
    int target = targetPlayer(m, isplayer);
    //King: if there are available targets, pick one and swap cards with
them
    //If no targets available
    if(target == 1337){
          if(isplayer){
               System.out.println("No target available! Discard card.");
               JOptionPane.showMessageDialog(null,"No target available!
Discard card.");
          }
          else{
               System.out.println(currentPlayers.get(m).toString() + "
discards their card. Nothing happens (no available targets).");
          }
```

```
    }
    //Else, will swap cards
    else{
          int tempPlayer = currentPlayers.get(m).cardDisplay(0);
          int tempTarget = currentPlayers.get(target).cardDisplay(0);
          currentPlayers.get(m).useCard(0);
          currentPlayers.get(target).useCard(0);
          currentPlayers.get(m).dealTo(tempTarget);
          currentPlayers.get(target).dealTo(tempPlayer);
    }
}

static boolean princess(int m){
    //Princess: When discarded, player is eliminated from play (you
shouldn't have done that!!)
    System.out.println(currentPlayers.get(m) + " eliminated themselves.");
     JOptionPane.showMessageDialog(null,
          currentPlayers.get(m) + " eliminated themselves.",
          "HAHAAHHAHAHAHAHAHAHAHAHHAHAHA!!1!",
          JOptionPane.WARNING_MESSAGE);
     log+=currentPlayers.get(m) + " eliminated themselves.\n";//add to
turn log
    currentPlayers.remove(m);//liminate self
    if(m == 0){ //If player #0 eliminates themselves, the counter should
stay at 0 (because players get shifted forwards)
          return false;
    }
    else{ //Otherwise shift the counter
          return true;
    }
}

//TARGETING PLAYER SYSTEM
static int targetPlayer(int m, boolean isplayer) throws IOException{
    //If it is a real player's turn: display the available targets
    if(isplayer){
          System.out.print("Pick a target: ");
          int numTargets = 0;
          for(int i = 0; i < currentPlayers.size(); i++){
               if(i!=m && currentPlayers.get(i).target()){
                    System.out.print(currentPlayers.get(i) + ", ");
                    numTargets++;
               }
          }

          System.out.println(); //Spacer
```

```
          if(numTargets > 0){ //Searches for which player they selected to
use the card on
               String [] victims= new String[numTargets];
               int cnt=0;
               for(int i = 0; i < currentPlayers.size(); i++){//add
targets into array to choose from
                    if(i!=m && currentPlayers.get(i).target()){
                         victims[cnt]=currentPlayers.get(i).toString();
                         cnt++;
                    }
               }

               int gogo=1;
               Object target=null;
               while(gogo==1){
                    target = JOptionPane.showInputDialog(null,
                         "Choose the Target", "Who shall be Judged",//body
text,title text
                         JOptionPane.INFORMATION_MESSAGE, null,//type of
icon
                         victims, victims[victims.length-1]);//optins
choice
                         //keep asking if canceled
                    if(target!=null)
                         gogo=0;
               }
               int i = 0;
               for(; i < currentPlayers.size(); ++i){
                    if(target.equals(currentPlayers.get(i).toString())){
                    log+=currentPlayers.get(m).toString() + " targets " +
currentPlayers.get(i).toString()+"\n";
                    break;
                    }
               }
               return i;
          }
          else{ //If 0 targets, their turn is void (they just simply
discard their card)
               return 1337;
          }
    }
    //If an AI, randomly slects a target
    else{
          int numInvul = 0;
          int targetOfAI;
          for(int i = 0; i < currentPlayers.size(); i++){
               if(!(currentPlayers.get(i).target()) ){
```

```
                numInvul++; //Keeps track of how many targets are not
available
                }
           }
           if(currentPlayers.size()-1 == numInvul){  //If 0 available
targets, then their turn is void
                return 1337;
           }
           //Otherwise, if targets available, search for which target they
selected
           else{
                while(true){
                     targetOfAI =
(int)(Math.random()*currentPlayers.size());
                     if(targetOfAI!=m &&
currentPlayers.get(targetOfAI).target()){
                          break;
                     }
                }
                String target = currentPlayers.get(targetOfAI).toString();
                int k = 0;
                for(; k < currentPlayers.size(); k++){
                     if(target.equals(currentPlayers.get(k).toString())){

System.out.println(currentPlayers.get(m).toString() + " targets " +
currentPlayers.get(k).toString());
                          log+=currentPlayers.get(m).toString() + " targets
" + currentPlayers.get(k).toString()+"\n";
                          break;
                     }
                }
                return k; //Returns the index of that target
           }
      }
}

static int targetPlayer(String curPl, boolean isplayer) throws
IOException{
    //In this case, there is always 1 available target, so there is no
need to search for if there are any targets available (you can target
yourself with the prince)
    //If it is a real player's turn, display the cards they have and ask
for an input for their choice (same as above)
    if(isplayer){
           int numTargets=0;
           System.out.print("Pick a target: ");
           //find number of targets
```

```
          for(int i = 0; i < currentPlayers.size(); i++){
               if(currentPlayers.get(i).target()){
                    System.out.print(currentPlayers.get(i) + ", ");
                    numTargets++;
               }
          }
          System.out.println();
          String [] victims= new String[numTargets];//list of targets
          int cnt=0;
          //add targets to array of targets
          for(int i = 0; i < currentPlayers.size(); i++){//add targets
into array to choose from
               if(currentPlayers.get(i).target()){
                    victims[cnt]=currentPlayers.get(i).toString();
                    cnt++;
               }
          }
          int gogo=1;
          Object target=null;
          //keep asking for target if canceled
          while(gogo==1){
               target = JOptionPane.showInputDialog(null,
                    "Choose the Target", "Who shall be Judged",
                    JOptionPane.INFORMATION_MESSAGE, null,
                    victims, victims[victims.length-1]);
               if(target!=null){
                    gogo=0;
               }
          }
          int i = 0;
          for(; i < currentPlayers.size(); i++){
               if(target.equals(currentPlayers.get(i).toString())){
                    log+=curPl + " targets " +
currentPlayers.get(i).toString()+"\n";
                    break;
               }
          }
          return i;
     }
     //If player is an AI, make them randomly find a target (same as above)
     else{
          int targetOfAI;
          while(true){
               targetOfAI = (int)(Math.random()*currentPlayers.size());
               if(currentPlayers.get(targetOfAI).target()){
                    break;
               }
```

```
          }
          String target = currentPlayers.get(targetOfAI).toString();
          int k = 0;
          for(; k < currentPlayers.size(); k++){
               if(target.equals(currentPlayers.get(k).toString())){
                    System.out.println(curPl + " targets " +
currentPlayers.get(k).toString());
                    log+=curPl + " targets " +
currentPlayers.get(k).toString()+"\n";//add to turn log
                    break;
               }
          }
          return k;
     }

}
//main
public static void main (String args [])throws IOException{
     Love_Letter app = new Love_Letter ();//call game code

     app.addWindowListener (   //add window
          new WindowAdapter ()  {
               public void windowClosing (WindowEvent e){//close window
when top close button is pressed
                    System.exit (0);  //exits and closes window
               }
          }
     );
}

}//end of class

class Player {
  //Every player has the following properties: name (String), hand of
cards (ArrayList), a status of targetability (boolean), and a real/AI
player status (boolean)
  private String nameP;
  private ArrayList <Integer> handP;
  private boolean targetable = true;
  private boolean isPlayer;

  //Player enters their name - from buffered reader, and also get a new
hand or AI get their name randomly generated and also get a new hand, and
also get assigned a real/AI player status
  Player(String Name, boolean player){
    nameP = Name;
    handP = new ArrayList <Integer>();
```

```
  isPlayer = player;
}

//Returns the string value instead of the address of the object.
public String toString(){
  return nameP;
}

//Adds the randomly gen'd card to player's hand
void dealTo(int add){
  handP.add(add);
}

//Finds the amount of cards in the player's hand
int numCards(){
  int siz = handP.size();
  return siz;
}

//Returns the value of the card that the player has at a certain index
int cardDisplay(int n){
  int nTh = handP.get(n);
  return nTh;
}

//Uses up the player's card by discarding it
void useCard(int i){
  handP.remove(i);
}

//Makes the player targetable once again (on start of their turn)
void targetable(){
  targetable = true;
}

//Returns the player's targetability status
boolean target(){
  return targetable;
}

//Makes the player not targetable
void handmaid(){
  targetable = false;
}

//Returns player's real/AI player status
boolean isPlayer(){
```

```
    return isPlayer;
  }
} //End of Player class

class MasterDeck{
  private ArrayList <Integer> masterDeck = new ArrayList
<Integer>(Arrays.asList(1, 1, 1, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 7, 8));
//This is the custom deck
  int deckLength = masterDeck.size();
  //Core functions to randomly generate numbers and deal the cards to the
players
  int random(){
    return (int)(Math.random()*masterDeck.size());
  }
  void remove(){
    int random = random();
    masterDeck.remove(random);
  }
  void remove(int in){
    masterDeck.remove(in);
  }
  int deal(){
    int idx = random();
    int val = masterDeck.get(idx);
    remove(idx);
    return val;
  }

  //Finds the current amount of cards remaining in the deck
  int remaining(){
    int i = masterDeck.size();
    return i;
  }
  void clear(){ //Clears the entire deck
    masterDeck.clear();
  }
} //End of MasterDeck class
```