# Requirements

1. Enables the user to set up a 7-by-6 Connect Four frame
2. The frame should include two different coloured discs (blue and red), with one on each side of the frame
3. At the start of the game, no discs should be entered in the frame
4. The order of play should be determined randomly
5. The user will be able to choose to start a new game
6. When all discs the user wants to place have been placed in the frame, the system should analyze whether the current state is possible or not (if there is no disc beneath a disc, it can only be supported if it’s position is at the bottom of the board), and all errors will be highlighted on the screen (including an unbalanced number of blue discs in comparison with red discs)
7. The final state of this frame cannot include any winning connected “line” of 4 discs of the same colour

# Design

The first class is called Model. This class is responsible for linking all the methods; it initializes the action listener and the game panels. It calls all of the classes and passes the appropriate data into them so that they may execute with the appropriate arguments.

Another class, Disk, constructs an abstract data type “Disk”. This class allows us to create objects of type Disk that are made up of two integer values: x and y. These are the x and y coordinates of each disk object, which are used to represent the location of each disk on the game board when a game is being played.

Another class used in the design of the Connect4 game is View. This class is responsible for creating the buttons, the labels and the background images. This class essentially sets up the images for the graphical user interface, setting up each panel in the game (the game board, the start screen) when they are called.

Lastly, we implemented a class “Control” that is responsible for obtaining the location of each mouse click and dealing with it appropriately. In this class, we implemented an action listener that, depending on the panel the program is currently in, executes the corresponding action.

We chose to decompose the project into these classes to prevent coupling while allowing for maximum cohesion; with only four classes, each class is mostly dependent on its own methods, gathering minimal data from other classes and therefore avoiding problematic dependencies by reducing dependencies on other classes.

# Classes and Modules

Model Class

* 4.2 Public Entities/Methods
  + Model(): a no argument constructor that allows a static method to call a non-static variable or non-static method from the Model class
  + Model(JPanel game\_panel, boolean developermode): initializes game panel, calls developer
  + mouseCicked, mouseEntered, mouseExited, mousePressed: all take in MouseEvent e and are required, override methods used to allow mouse clicks
  + mouseReleasted(MouseEvent e): when the mouse is released, uses the mouse function method in control
  + actionPerformed(ActionEvent e): is a button listener, which calls the button function in control
  + main(): calls appropriate methods in order to create the main window, and displays the panel
* 4.4 Traceback
  + satisfies requirement 5, allows user to enter game screen when they click button “Start Game”
* 4.5 Private Entities/Methods
  + createFrame(JFrame main\_frame): sets up the location of the window as well its size
  + The variable dev\_mode is used to check if the game needs to be run in developer mode or not. This variable is then used again in control when the user clicks the developer mode button.
  + The variable card\_layout determines how the panels are shown. This variable is used in view to create what it looks like and control to display it
  + The variable deck\_panel is the main panel in which the card layouts are displayed. This variable is used in view to add the panels to the deck and control to display it when the user clicks the appropriate button.
  + The variable main\_frame makes a new JFrame type named "Connect Four" (This will be the name of the window) this variable is used in view to create the size and control to display it when the user clicks.
  + The variable check\_disk checks the disk for if there is a piece on the position that the user has clicked. This is used in control when the user clicks a spot on the board

Disk

* 4.2 Public Entities/Methods
  + Disk(): default constructor that prevents an error from occurring if disk is called without any arguments
  + Disk(int x, int y): constructor to set the x and y coordinates for a Disk object
  + setX(int x): sets the value of the x coordinate in a Disk object
  + setY(int y): sets the value of the y coordinate in a Disk object
  + getX(): returns the x coordinate of the Disk object
  + getY(): returns the y coordinate of the Disk object
* 4.4 Traceback
  + an abstract data type that allows game pieces to be created, contributing to satisfying requirements 6 and 7
* 4.5 Private Entities/Methods
  + All of the variables in this are private (expand more)

View

* 4.2 Public/No Modifier Entities/Methods
  + titleScreen(): creates title screen, sets up the locations of the buttons
  + infoScreen(): displays the instruction screen, loads the buttons for this screen
  + gameScreen(): displays the game board, as well as 2 disk pieces; this method also loads the buttons for the game board
  + developerScreen(): creates the buttons for the developer screen and sets up the screen for developer mode
* 4.4 Trace back
  + this class satisfies requirements 1, 2 and 3 by setting up the game board and displaying disks

• 4.5 Private Entities/Methods

* + the variable game\_panel is being used in the class control, this variable holds the current game panel and control takes this panel and displays it when the mouse is clicked
  + The rest of the view

Control

* 4.2 Public Entities/Methods
  + buttonFunction(ActionEvent e, JPanel panel, boolean dev\_mode): contains all functions of each button. When a button is clicked, this method finds the name of the button in order to decide which action should be executed (for example, if exit is clicked, a confirmation popup will appear to confirm whether or not the user wishes to close
  + mouseFunction(MouseEvent e, JPanel panel, boolean dev\_model): this method is called when the mouse is clicked. It checks if that click is within the range of a the of the position its clicked it then sets the x and y values of the position it clicked it. It then checks if dev\_model(developer mode) is on or off, if it is on then it places the disk in the position of the board that is clicked. If dev\_model is turned off then it places a disk at next available position of the column.
* 4.4 Traceback
  + This class satisfies requirements 1, 4, 6 and 7 by allowing a random player to go first, allowing the user to place a disk on the game board. In developer mode, this class allows the game **board to place disks anywhere on the game board.**
* 4.5 Private Entities/Methods
  + blueDisk(int x, int y, JPanel panel): sets the location of the blue disk, as well as the panel it is located on
  + redDisk(int x, int y, JPanel panel): sets the location of the red disk, as well as the panel it is located on
  + playerNameSet(JPanel panel): prompts the user to enter each player’s name, displays the name in the proper locations of the game board
  + **pieceAir(): determines whether or not there is an empty space underneath the location the user tries to put ­­a disk. If there is no empty space (if there is either a disk underneath or the piece has reached the bottom of the board), this method returns true**
  + win(): in developer mode, checks whether or not 4 pieces have been placed in a row vertically, horizontally or diagonally; **if so, …** Otherwise, the method returns false. This method is used for determining whether or not a player has won
  + All the variables in this class in this class are private and used within in each method (add more)