# Timed High-Card Game

******This is a good program to add to your portfolio.*****

**Understand the Classes and Problem**

Last week we made a GUI card game.  Now it is time to modify it even further.  Redesign Phase 3 from the last module by adding a timer and put it into a design pattern.

Here are the phases:

1. Change the program into a Model-View-Controller Design Pattern.
2. Add a new part to the High-Card game by putting a timer on the side of the screen.  It will be on a timer to update every second, but in order for you to still play the game, you will need to use multithreading.  (Timer class)
3. Design a new game.
4. Redraw the UML diagram so that it represents your new structure.

**Phase 1: Model-View-Controller Design Pattern**

You need to redesign last week's code into the Model-View-Controller Design Pattern.

There is a sample for #3 on pg 709.

This will help you as you take a look at what code needs to be moved around.

Make sure that the game still works as it did last week.

**Phase 2: Multithreading Implimentation**

Multithreading is nice to implement inside of a GUI program, so let's give it a try.

Time to add a timer to the High-Card game.  It should sit on the side of the screen and count up from 0:00, updating every second.

Of course the use multithreading will be needed to make sure you can still play the game with the timer running.

**Timer class (extends Thread)**

Overrides the run() method.  Put all of the needed timer code in the run() method.

- Display the timer box and numbers
- Create start and stop buttons to control the timer. (extra challenge: merge the two buttons into one start/stop button)
- Make a call to a doNothing() method that will use the sleep() method of the Thread class.

Note: The method Thread.sleep can throw an InterruptedException , which is a checked exception— that is, it must be either caught in a catch block or declared in a throws clause. The InterruptedException has to do with one thread interrupting another thread. The book simply notes that an InterruptedException may be thrown by Thread.sleep and so must be accounted for.  The example uses a simple catch block. The class InterruptedException is in the java. lang package and so requires no import statement.

You will need an actionPerformed() method in the main() class to create an object of the Timer class and call start().

**Phase 3: Make a new Game!**

Time to create a new game called "BUILD".  The timer created above will not directly impact the game, but will just be running on the side.

Here is how the game works:

1. Take turns with the computer putting a card on one of two stacks in the middle of the table.
2. You can put on a card that is one value higher or one value lower. (6 on a 5 OR 4 on a 5, Q on a J OR T on a J, etc.)
3. After you play, you get another card from the deck in your hand.
4. Keep going until the single deck is out of cards.
5. If you cannot play, click a button that says "I cannot play". The the computer gets a second turn. Same for you, if the computer cannot play. If neither of you can play, then the deck puts a new card on each stack in the middle of the table.
6. Who ever has the least number of "cannot plays", is the winner. Declare this at the end, when the deck is exhausted.

Have the game end when there are no more cards to draw from the deck. You will still have cards in your hand, but don't worry about that.

First use playCard() to get the card you want to play from playerIndex and at the cardIndex location. Then use the takeCard() to get a new card for end of the array. You will then need to reorder the labels by using setIcon().

In order for playCard() and takeCard() to work in the cardGameFramework class, add the following to the Hand class. This will remove the card at a location and slide all of the cards down one spot in the myCards array.

```
public Card playCard(int cardIndex)
{
  if ( numCards == 0 ) //error
  {
    //Creates a card that does not work
    return new Card('M', Card.Suit.spades);
  }
  //Decreases numCards.
```

```
    Card card = myCards[cardIndex];

    numCards--;
    for(int i = cardIndex; i < numCards; i++)
    {
      myCards[i] = myCards[i+1];
    }

    myCards[numCards] = null;

    return card;
  }
```

### Phase 4: Create the UML diagram

Redraw the UML diagram so that it represents your new structure.

******This is a good program to add to your portfolio.*****

### Submission

- Turn in one .txt file with your new phase 3 code that includes the timer from phase 2.  No output need be included, like last week, since it is GUI.
- UML diagram file.

- **Peer and Self Evaluation**

    - 5% of your grade will be based on your completion of an evaluation of your peers for this assignment and a self evaluation. Put your responses in the box below.
    - Communication -- Them with you?  You with them?
    - Who decided what each person would do?  Did they do their part? Did you do your part?

- How much effort did YOU put into the assignment?
- How hard was this assignment?
- In your own words, describe how your code solved the assignment. This should not be the same as any other team members explanation.