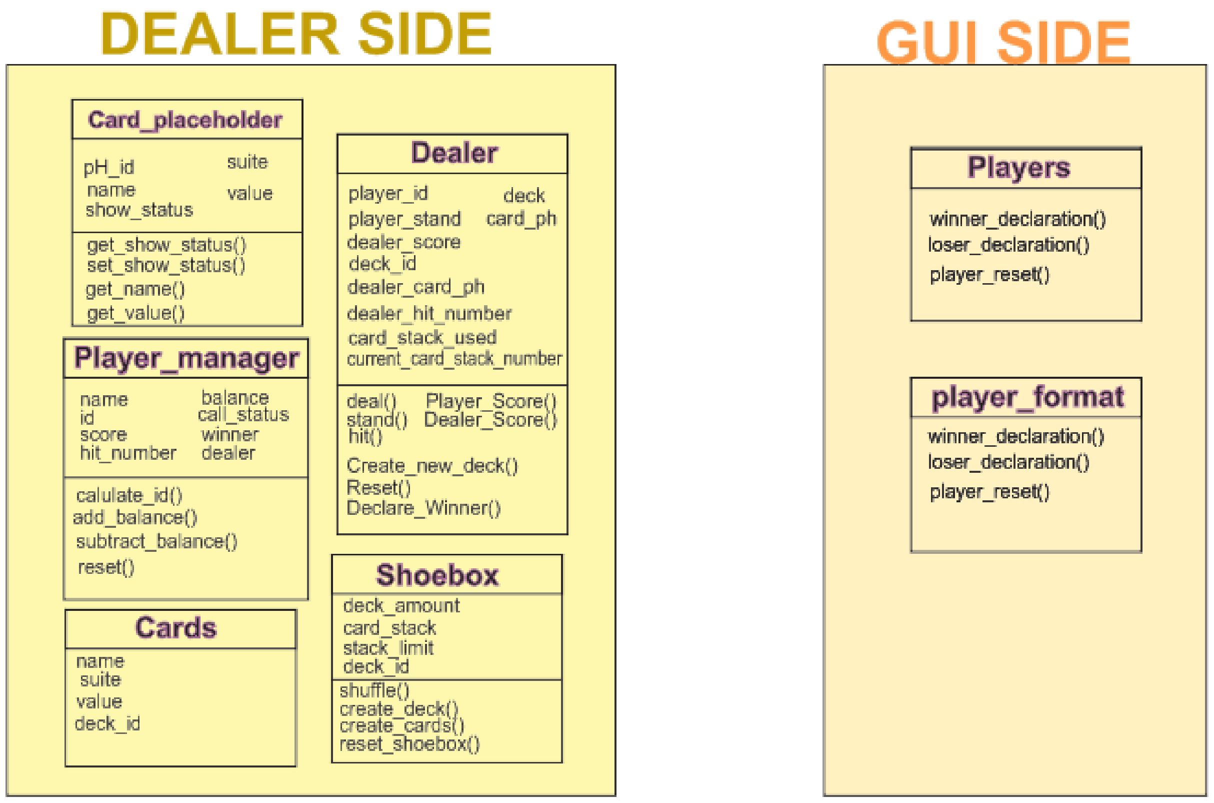
Black Jack

code by Neville Ekka

During the process of designing classes for the game, I have first taken into account of real-world scenario where various programmers looking to construct their own game with custom GUI will be utilizing my classes. Therefore to achieve universality, classes have been divided into two categories – Dealer side and GUI side. Following classes below are distributed to their respective sides.



For simplicity for programmers, any custom class (Players in this case) needs to created while inheriting from abstract class Player_Manager and implementing player_format interface to effectively interrupt game to announce winners. The dealer is class that manages all the other classes. Dealer is supposed to a singleton class but I have not implemented as such to keep it simple.

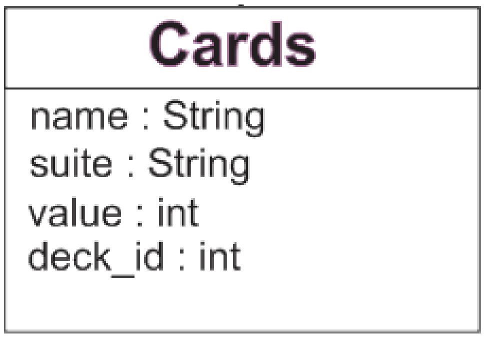
The **key features** of this Black Jack game are:

- Supports multiple players
- Shoebox that can mix multiple decks.
- Dealer and Player soft hand calculation

Black Jack

code by Neville Ekka

Class Cards



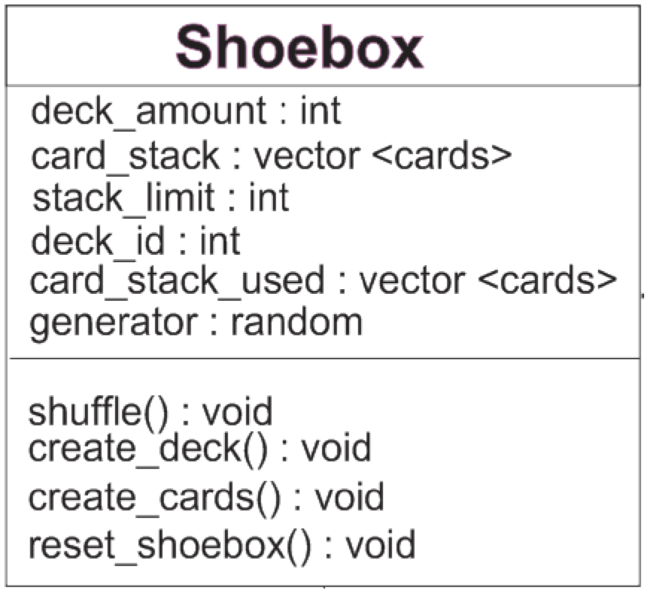
It is a simple member class that manages required fields for each instance card created. It would be better to keep cards nested inside so only the dealer can use (only the dealer class uses it however) it but to keep code organized for simplicity I have made it a separate member class.

```
public String name;  => name of the Card  2= 2, Q= queen, K= king, A=
                        ace...etc

public String suite; => name of suite. Can be used to store actual suite names
                        but for simple implementation. I have used number.
                        Only the GUI really needs to use this field.

public int value; => Value of the card according to black jack game. 1-9 cards
                     have their own name value. 10,J,K,Q had value of 10.
                     A,ace, however has value=0 to allow dynamic calculation.

public int deck_id; => which deck it belongs to.

 public int id; => ID of that individual card. Although not useful, value
                   assignment by shoebox class is key to shuffling method.
```

Black Jack

code by Neville Ekka

Class Shoebox



A member class that is manages all cards, followed by their decks. Its key function is shuffling and resetting all cards. It would be better to keep cards nested inside so only the dealer can use (only the dealer class uses it however) it but to keep code organized for simplicity I have made it a separate member class.

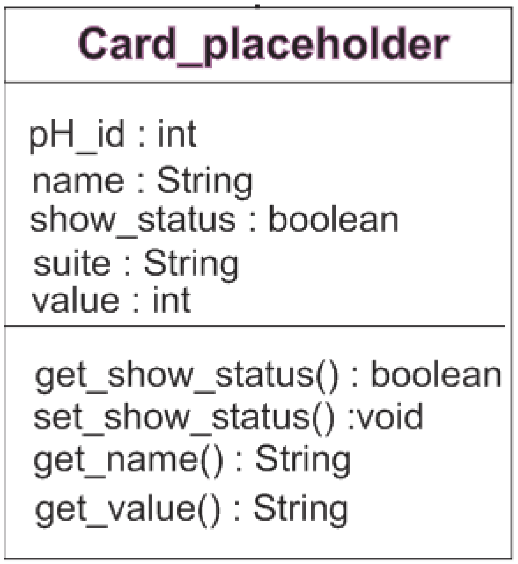
```
public int deck_id; => Used to store Ids for decks created. Amount of decks = deck_id

public int card_id=0; => This is required to keep track of cards created in a deck or
                         or multiple deck so that it can used to randomly pick cards
                         from their id.

public int stack_limit= 100; => To make sure the least amount of shuffled cards are
selected, enough to be used by maximum hit called by players and dealer, a limit is
set to conserve memory.

public Vector<cards> card_stack = new Vector<cards>(1);
=> A vector array used to store all the cards created even the ones created for other
multiple decks. It has been initialized to simply fill the index 0 in the array.

public Vector<cards> card_stack_used = new Vector<cards>(1);
=> A vector array used to store all the cards randomly picked from the main stack of
cards. This field is mainly used by shuffle() method. It has been initialized to
simply fill the index 0 in the array.


private Random generator = new Random(); => random generator for shuffling cards.
```

Black Jack

code by Neville Ekka

```
public void shuffle() => Cards from card_stack are transferred randomly to
card_stack_used with a limit put by stack_limit.

private void create_cards() => Create 13 x 4 cards with assigned names and values.
public void create_deck(int deck_id) => Creates new deck and assigns id.
```

Class Card_placeholder



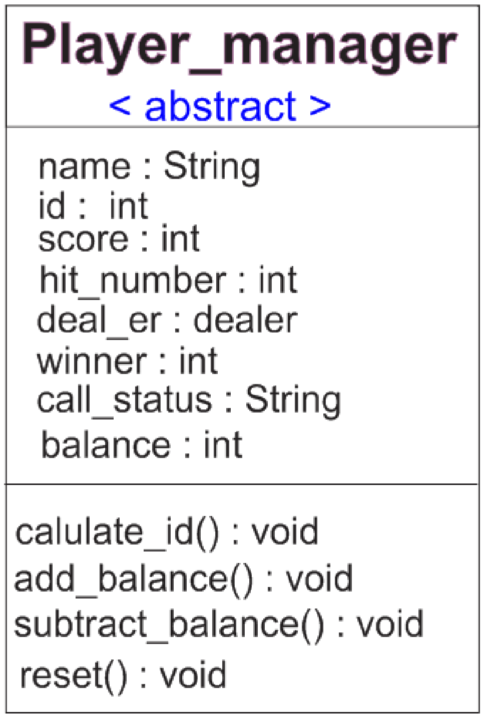
The objects of class Card_placeholder can be visualized as an exact area on the table where the cards are placed. Only one card be placed on that area. This class is used mainly for GUI to keep track of cards placed as well as used to calculate values of the cards in an organized manner. It has the function of flipping cards up or down. Each card dealt to either dealer or players is linked to the placeholder objects.

```
    public int ph_id; => Ids used to keep track of how many Placeholders created.

    private String name; => Used to store names of the linked cards

    private String suite; => Used to store suite of the linked cards

    private int value; => Used to store value of the linked cards

    private boolean show_status; => flip card up/down. It can be accessed or set
                                    anytime.
```

Black Jack

code by Neville Ekka

Class Player Manager



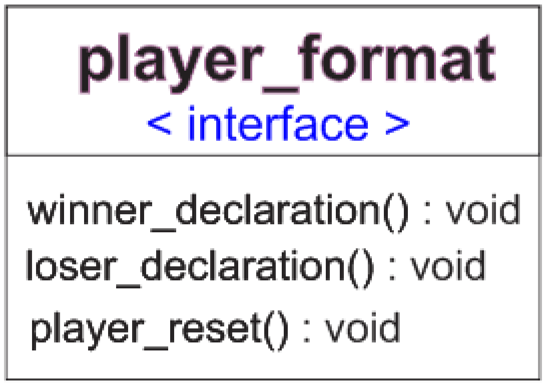
This is an abstract class that is supposed to be extended by the class players . This class keeps tracks of all the player information and plays key role in communication of information between GUI side and dealer side classes. It also protected fields to directly communicate with its sub classes i.e players.

```
public String name; => Stores player name

protected int id; => Stores player specific IDs. Not really used.

public int score; => Stores Player score

public int hit_number=0; => Stores Player number of hits

private int balance; => Stores Player current Balance

public String call_status=""; => Stores Player current Call status i.e hit &
                                 stand

public int winner; => This field is where the updated information about
                      players winning status is stored. Value of 0 = ingame,
                      1 = has won, 2= has lost

public static dealer deal_er; => Stores class dealer instance
```

Black Jack
code by Neville Ekka

```
public void calculate_id(dealer deal_er) => Checks the dealer instance for current
                                           number of players and then increments that
                                           value. It also assigns  that player a
                                           specific ID.

      ** Reset of the methods are self explanatory.
```

Interface player_format



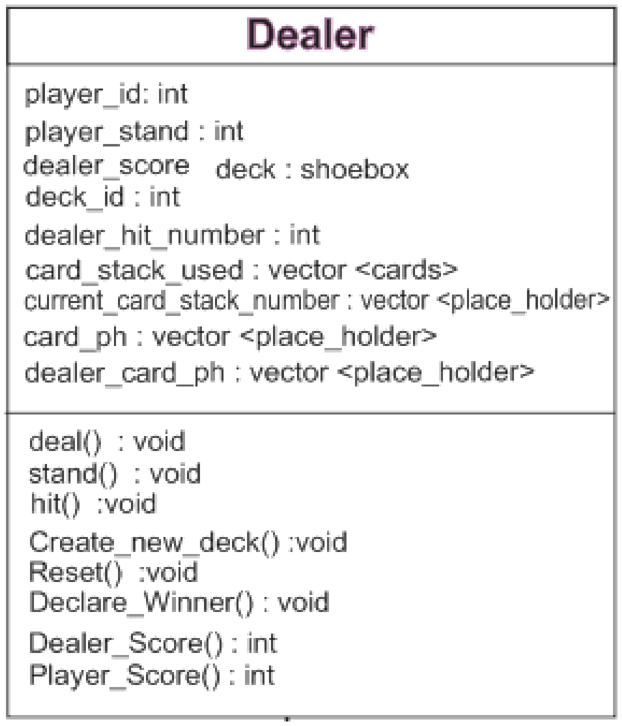
This is an interface that is implemented by class player. The functions implemented are mostly called by methods in dealer class to immediately trigger the implemented code in those 3 methods, if the player has won or lost. Interfacing these methods not only allow for immediate response but also allows user to execute their custom response.

```
public void on_winner_declaration();=> Action response on player win

public void on_loser_declaration();=>Action response on player lost

public void player_reset(); => Fields in player class that needs to be reset
                               the beginning of the game.
```

Black Jack

code by Neville Ekka

sClass Dealer



```
private static int players_id=0; => Used to keep track of all players. It is mainly u
                                    to detect if all players have “stand” call status
                                    so that the dealer can show his cards and
                                    conclude winner.

private int player_stand=0;   => Used to keep track of all players with “stand”
                                call status. Used in stand() to trigger winner
                                declaration.


private int dealer_score=0; => Keep track of Dealer score. It is always updated by
                               dealer_score();

private int deck_id=0; => Used to store IDs of all decks. It is accessed by class
                          Shoebox.

private int dealer_hit_number=0; => Used to store total number of times the dealer
                                     calls hit. It is triggered by all_stand() that
                                     is called by stand().

private int current_card_stack_number; => It is used to keep track of how many cards
                                          are taken out of shuffled stack. Used to
                                          ensure same cards are not give to place
                                          holders.
```

Black Jack

code by Neville Ekka

```
private shoebox deck = new shoebox();  => Stores the only copy of shoebox.

private Vector<cards> card_stack_used =new Vector<cards>(1);
 => Card stack used to draw shuffled cards from. It has been initialized to simply
fill the index 0 in the array.



private Vector <card_placeholder> card_ph =new Vector<card_placeholder>(1) ;
=> Stores all the card place holder objects that are used by players.  It has been
initialized to simply fill the index 0 in the array.


private Vector <card_placeholder> dealer_card_ph = new Vector<card_placeholder>(1);
Stores all the card place holder objects that is used by dealer.  It has been
initialized to simply fill the index 0 in the array.
```

```
public void deal(players player) => Deals card to specific player i.e on player
                                    specific placeholder. This method begins the game

-------------------------------
   reset();
       player.reset();
       player.player_reset();
       create_new_deck();
       deck.shuffle();
--------------------------------
Resets fields in dealer class and resets player_manger fields and sub class of
player_manager before creating deck. Shuffling is called shuffle cards.

--------------------------------
       this.card_stack_used = deck.card_stack_used;
       dealer_card_ph.add(new card_placeholder());
       dealer_card_ph.add( new
card_placeholder(card_stack_used.get(1),false,player));
       dealer_card_ph.add( new card_placeholder(card_stack_used.get(2),true,player));
       this.dealer_hit_number=2;
--------------------------------
All cards are placed in respective placeholders. With false indicating the first card
by dealer is faced down. The dealer has dealt cards 2 times himself technically
calling hit so dealer_hit_number=2 .

--------------------------------
       card_ph.add(new card_placeholder());
       card_ph.add(new card_placeholder(card_stack_used.get(3),true,player));
       card_ph.add(new card_placeholder(card_stack_used.get(4),true,player));
       player.hit_number=2;
       current_card_stack_number =4;
      player.score= player_score(player);

--------------------------------
```

Black Jack

code by Neville Ekka

The dealer has dealt cards 2 times faced upto players place holder. current_card_stack_number indicating all cards drawn equals 4. Also calculates player score.

```
public void hit(players player) => Hits card to specific player i.e on player
                                   specific placeholder.
```

```
      if(!player.call_status.equals("stand") ){
      player.call_status="hit";
      player.hit_number++;

      this.current_card_stack_number++;
```

Stores a "hit" status to respective player only if players don't have "stand" status. Increments number of hits and number of cards drawn.

```
    card_ph.add(new
card_placeholder(card_stack_used.get(this.current_card_stack_number),true));

    player.score=player_score(player);


    if (player.score>21){player.winner=2;dealer_card_ph.get(1).set_show_status(true);
player.on_loser_declaration();}
    if
(player.score==21){player.winner=1;dealer_card_ph.get(1).set_show_status(true);player
.on_winner_declaration();}}
```

Adds new card placeholder ,faced up, for player and stores information in the place holder object. Also calculates player score and triggers on_winner_declaration() or on_loser_declaration() if player has score of 21 or over.

```
public void stand(players player) => Sets stand status for players and increments
                                     player_stand. Also it checks if all players have
                                     called "stand" by checking if player_stand=
                                     players_id . If they have all_stand() is called.


public void all_stand() => The end phase of the game where dealer flips up his first
                           card and checks if his score is less than 17 by calling
                           dealer_score(); Dealer keeps calling "hit" until the score
                           is more than or equal to 17. Winner is declared after
                           that.
```

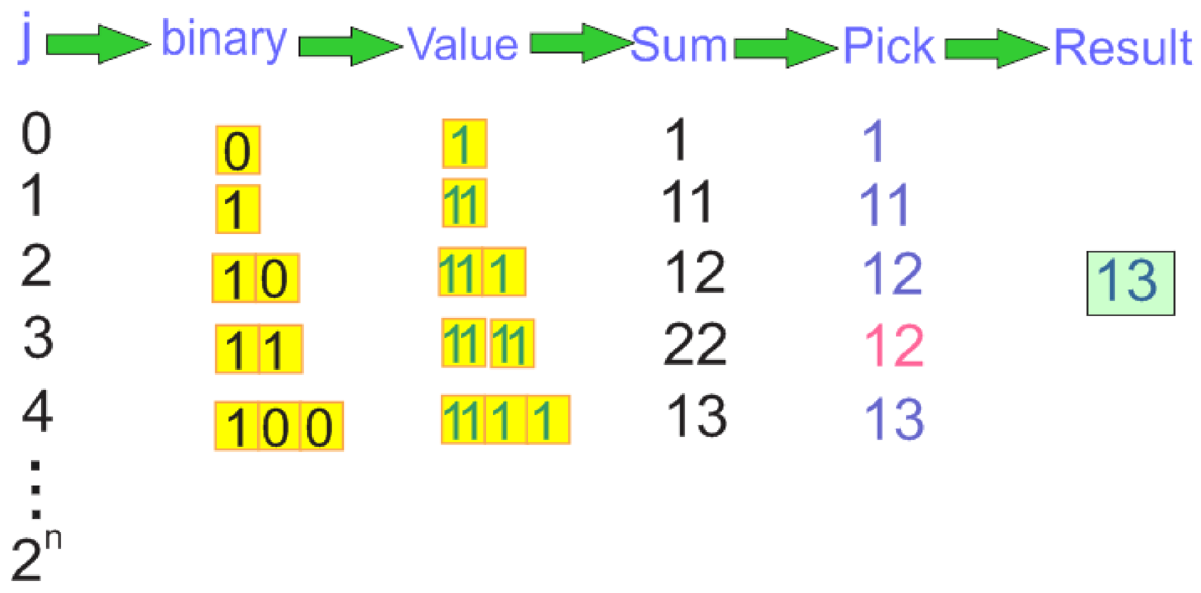
Black Jack

code by Neville Ekka

```
public void declare_winner(players player) => Method that has conditions to check if
                                              the players have won or lost.



public void create_new_deck() => IDs and creates new deck.

public int player_score(players player) => Same as dealer_score but triggers for
                                           player only. Explanation given on
                                           dealer_score().
```

```
public void dealer_score() => This method calculates dealer score by checking all the
dealer's place holder and adding them together. It also uses a clever but inefficient
method to calculate soft hands i.e hand involving aces, to determine best score. It
first adds all the non-aces placeholders if any exists then checks for number of aces
on the placeholders. Once atleast one aces is found, an array of 2^n, where n =number
of aces, is created. Numbers upto 2^n is generated by for loop and converted to
binary number. Bit 0 of the binary number equals 1 of the ace value whereas 1 equals
11. For each number all ace values are added. All possible aces values are stored in
an array where the best possible value, either 21 or closest to 21, is selected.
```

Below diagram depicts dealer/player_score operation.


j
binary
Value
Sum
Pick
Result
0
1
2
3
4
2^n
0
1
10
11
100
1
11
11 1
11 11
11 1 1
1
11
12
22
13
1
11
12
12
13
13