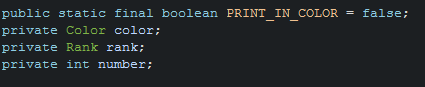
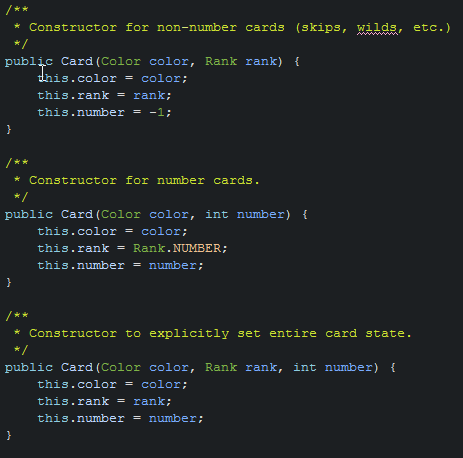
Uno Project by Class and line.

**Card.java**

****

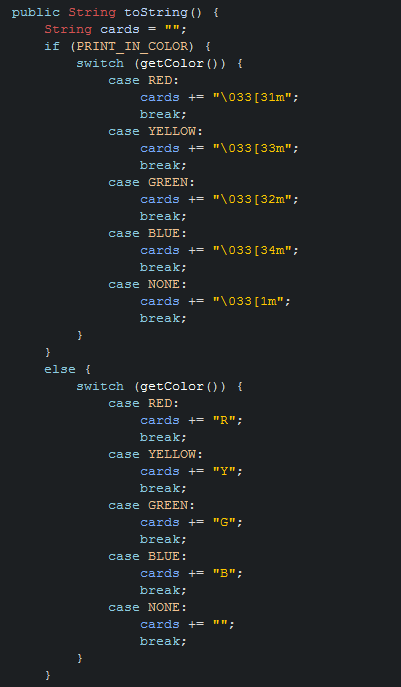
The beginning of the class file, has 4 Variables the first one is a class variable. The second and third create a Color Object and Rank object and finally the last is a int called number.

* PRINT\_IN\_COLOR = is a class variable because it is public available but cannot be changed by any code (static final) stops code being changed. This decides whether to use the color or to just use the the letters R Y G B “” to indicate Red, Yellow, Green, Blue or None.
* Private Color color , creates a Color object and this will be used to indicate what color is used the Color Object is based upon the Color.java which contains an ENUM with the colors that can be used.
* Rank rank, creates a Rank object called rank and this indicates what rank of the card is. This is based on the ENUM Rank in the rank.java file.



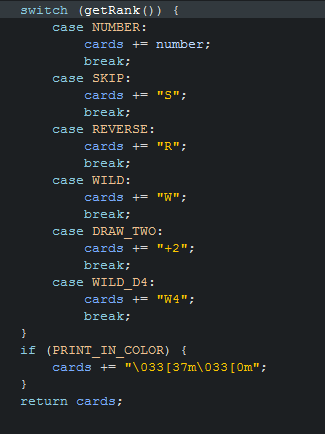
The next are constructors these create an instance of a Card and each handle different types of cards.

* Constructor Card (Color color, Rank rank) takes two input variables one is the color and the other is the rank for example RED , SKIP. And can be called like this Card (RED, SKIP) and we create a Card instance with the following. RED, SKIP, -1. This constructor is for non-numbered cards (SKIP, REVERSE, DRAW\_TWO, WILD, WILD\_D4) and that is why the number is set to -1 as we know the number has no value.
* Constructor Card (Color color, int number) takes two input variables one is the color and the other is the number for example RED, 4 and can be called like this Card(RED,4) and we will create a Card instance with the following (RED, NUMBER, 4). The Rank will be set to NUMBER as this constructor is for number cards only.
* Constructor Card (Color color, Rank rank, int number) takes all three variables and can be called to create a Card instance for example (RED, REVERSE, -1) which will create a red reverse card. This is used to create any card type you like.

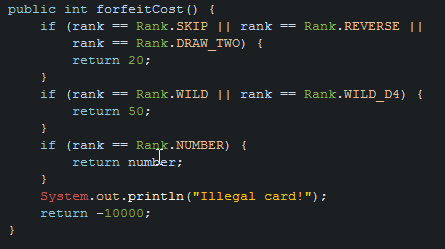


The toString() Method creates a string of the cards so we know what cards are available. We are using a switch statement as its provides much better readability and its quicker and uses less memory than executing if else statements. We do use a If statement at the beginning to figure out whether to use ANSI colors or Text Colors.

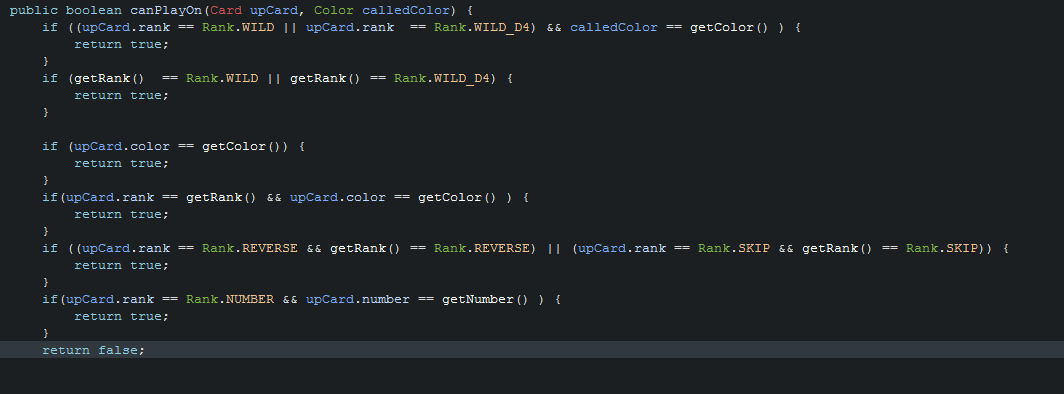
* Firstly we create a new method variable called cards which is an empty string. Then we check to see if PRINT\_IN\_COLOR is set to true. If its set to true then we check what color the current card is by using the switch statement:
  + The first part of a switch statement switch (getColor()) . The getColor() is called which returns the color of the card. And the word switch is expression to check against. For example switch(RED).
  + Then each case is checked against that expression above. So for example if it was switch(RED) it would get to case RED and do a comparison (RED equals RED) is true so set the string to the ANSI Color and then you are done. If it was switch(BLUE) it would check each case until it got to BLUE break there instead.
* If PRINT\_IN\_COLOR is set to false, then we go into the switch case which is the same as above except of returning ANSI Color. It returns R , Y , G, B or “”.



* Next we check the rank of the card in the same way as the color, we getRank() and then check that rank the current card is against the List. And add the Rank to the Card. Finally if the PRINT\_IN\_COLOR is true we add at the end the ANSI Escape. Then we return the string and the example would be like this “YR” would be Yellow Reverse or “G5” Green five. To read more about ANSI COLOR <http://jafrog.com/2013/11/23/colors-in-terminal.html> but basically it just tells the terminal how to represent the color.

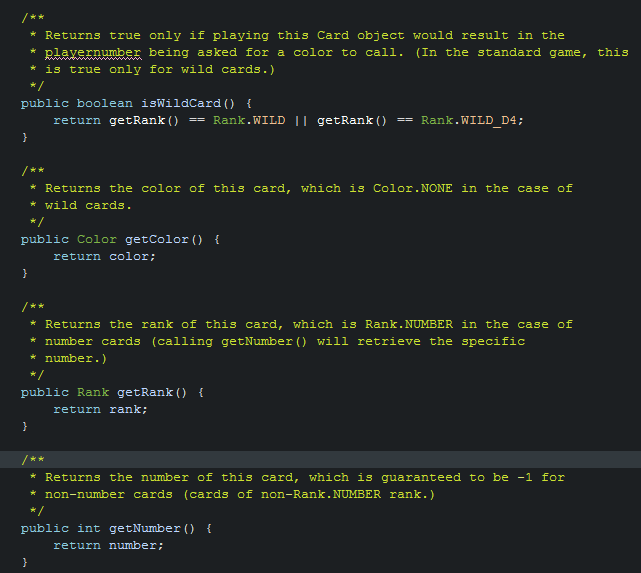


The next method called forfeitCost() is based upon the amount of points that will count against the player who holds it in his / her hand when another player goes out.

* The first if checks to see if the rank equals SKIP, REVERSE or DRAW\_TWO if that is true then we return 20/
* The second rank checks to see if the rank equals WILD or WILD\_D4 and if true returns 50
* The last one checks the RANK equals Number and if it does it returns the number value.
* Finally if its not any of them then its an illegal card and we return -10000

The next method is a Boolean that checks if a card can actually be played. It takes to variables the first one is the current card upCard which An "up card" upon which the current object might (or might \* not) be a legal play. And the calledColor that is only valid if it’s a wildcard that the player can call a color on.

* + The first if does the following it checks to see if the upcard is either WILD or WILD\_D4 and the calledColor equals getColor (which if you remember returns the color of this card) and if this is true. They can play a card.
  + The second if the getRank() is either a WILD or WILD\_D4 if so that is also okay to play. (remember getRank gets the rank of this card not the upcard)
  + The next checks to see if the upcard and this card have the same color if they do we can play it.
  + The next one checks to see if the upcard rank and color both match the current cards rank and color and if it does then we can play that card.
  + The next if check to see if the Upcard is reverse and this card is reverse OR upcard rank is SKIP and this card is also SKIP.
  + Finally the last one checks to see if the upcard is a number and the upcard matches the number of this card.
  + If none of those are true. Then we can’t play the card.



The final part of the Card.java file are getters that return the rank number or color. We have a special one for wildcard to check if it’s a wild.

* The isWildCard() checks if getRank() returns either WILD OR WILD\_D4 and returns true if they are.
* The getColor() returns the color of current card.
* The getRank() returns the rank.
* Finally getNumber() last returns the number of the card.

**Color.java**

The Color.java is a Enum that contains all the aloud colors

public enum Color { RED, YELLOW, GREEN, BLUE, NONE }

Its only purpose is to handle the color of a card.

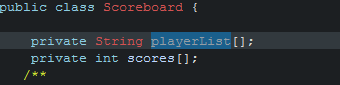
**Rank.java**

The **Rank**.java is a Enum that contains all the aloud colors

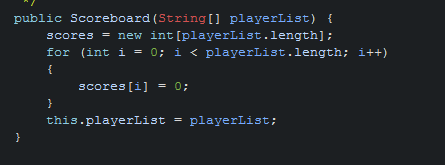
public enum Rank { NUMBER, SKIP, REVERSE, DRAW\_TWO, WILD, WILD\_D4 }

Its only purpose is to handle the Rank of a card.

**Scoreboard.java**

****

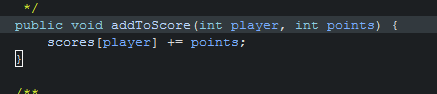
The beginning of Scoreboard.java does two things. It creates to private arrays one that handles the list of players and the other handles the scores.



This Creates the scoreboard so firstly we get parsed the player list for example (Joe ,John , Steve, Peter) and then we set scores to a new array object to the size equal to the length for example

**scores = new int[4]** we then loop through each player and set his score to 0 (because we haven’t played the game yet.

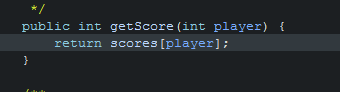
Then we set the this.playerlist that we created to the playerList of the class.



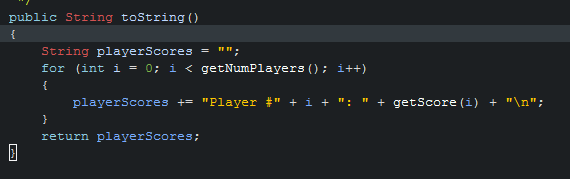
AddToScore takes the player (e.g player 1 or player 2) and the amount of points they get(500 or 50 ) and adds it to that players score. For example if the player is player 1 and the points was 50 it would look like.

**Scores[1] += 50.**

The += means add this number to the current number which is different to if it was just scores[1] = 50 which would just override it.



This returns the score of the player in that position for example if its player 1 again. It would return scores[1] which would return a value like 50 or 5000 depending on the score.



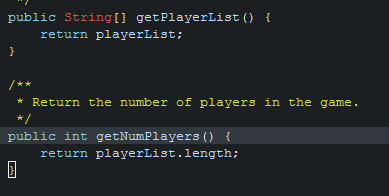
The ToString() method creates a string with every player and their score.

* Firstly we set an empty string
* Then we loop through each player and add their name, score and add a new line
* When we are done we return playerScores.

This would look like this:

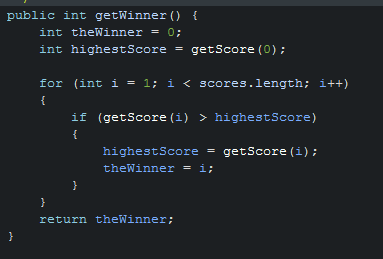
“Player # John: 50

Player # Steve: 600”



The GetPlayerList() returns the list of players for example "John", "Susan", "Michael", "Delenn", "Na'toth"

The second returns the amount of players for example "John", "Susan", "Michael", "Delenn", "Na'toth" would return 5.

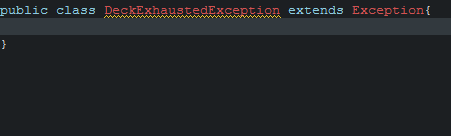


This method figure’s who has the highestScore.

So first we set the winner to 0 and we set the highestScore to getScore(0) which is the first in the array (arrays start at 0 index).

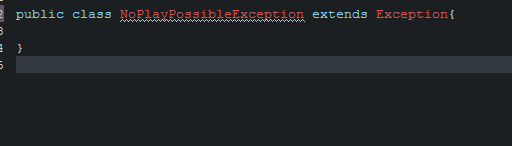
We then loop through all of the scores starting at 1 as we already have position 0 score. WE check to see if the getScore of each player is higher then the current highestScore. If the getScore(i) is we set the highest score equal to that score and set the winner to that position of I. Finally when we found the highest score we return that winner.

**DeckExhaustedException**



This class is created to handle if there were no more cards left, the professor asked for this but we never actually used it.

**NoPlayPossibleException**



This class is created to handle if there were no more moves left, the professor asked for this but we never actually used it.

**The Test classes**

I won’t go through the test classes because this code isn’t implemented by you but the reason for these classes is so we can make sure your code does everything your professor wanted. Usually you write tests when you are a developer to make sure what you think your code does in fact do that. The point of these tests in your example is just a code checker and the only way to run these. The jar files are the Junit testers that people use from command line or terminal (if on MAC) but when you do it in say eclipse or NetBeans you can just say this is a Junit test and it will import the files. If one of these tests doesn’t pass it will tell you exactly which test failed so you can compare and look at the code and find your bug.