# Appendix: Designing Together

Group Members:

- Cardenas Godoy, Crystal Hatsumi

-Hamson, Brendon

-López, Cristian

-Canteros Barrios, Romina

During week 3 we learned about "Abstraction", the meaning of this is the decomposition of our code into elements. Classes, functions, and such. These can be changed and modified without affecting the entire program.

**Project structure:**  
  The purpose of this program is to build a solid foundation of knowledge about how abstraction is used.Our repository contains a branch where it stores two files:  Class Card and Class Player.  
Card's file complies with the logical part of the program; it returns a value true or false, depending on the user's choice and calculates the points during the game.  
The Player's file directs the game, through the different functions or rules that make the abstraction of the game. That means that each function is focused on one detail of the player's choice. For example, when the game starts and  asks the user if he wants to continue playing or not, using the "play" function if the player continues the game the function asks him what guess is low o high with " get\_inputs". The function “do\_updates" call the card's class and check with the user's guess, "do\_outputs" returns the value of random value and the total points.The game starts and the class player initiates by asking the methods if the user is\_playing, score and card. It will check if he has the option of playing and from the class Card get the random number. Then the game asks for an input, and according to that answer it will update the information, using the do updates method, where it will verify the guess, calculate the score using the get\_point method in class Card and append a new card if the player is still playing. In the do outputs the user will have the opportunity to continue playing until he runs out of points or decides to stop playing.

The following graph shows the main classes and objects to guide us in the project:

**Captain**

* check\_life()
* get\_inputs()
* do\_updates()
* do\_outputs()
* play()
* cards
* score
* is\_playing

**Card**

* card\_number
* points
* check\_guess()
* get\_points()

Having clear the following concept. A class is a template for creating objects. An object is an instance of a class. The class may have an initialization method called a constructor that will be called each time an object of that class is created. The constructor is usually used to initialize the attributes of objects. The two main examples of OBJECTS are the following:

def \_\_init\_\_(self):

self.cards = [Card ()] (we create an object when we initialize the code we have to call the class card to work it, the name would be “cards”)

self.score = 300

self.is\_playing = self.check\_life()

def main():

game=Captain() (Another object example that we have is at the end of the program we call the class “Captain” with the name “game”, this function is used to execute all the code worked before))

game.play() ( “game” call the function “ play to show the card’s number)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**division of team responsibilities**

- Cardenas Godoy, Crystal Hatsumi : In charge of assembling the structure of classes “Card”.

-Hamson, Brendon: In charge of assembling the structure of classes “Player”.

-López, Cristian: In charge of correct code’s errors, complete README parts.

-Canteros Barrios, Romina: In charge of make de README and arm the code parts.