Ship of Fools

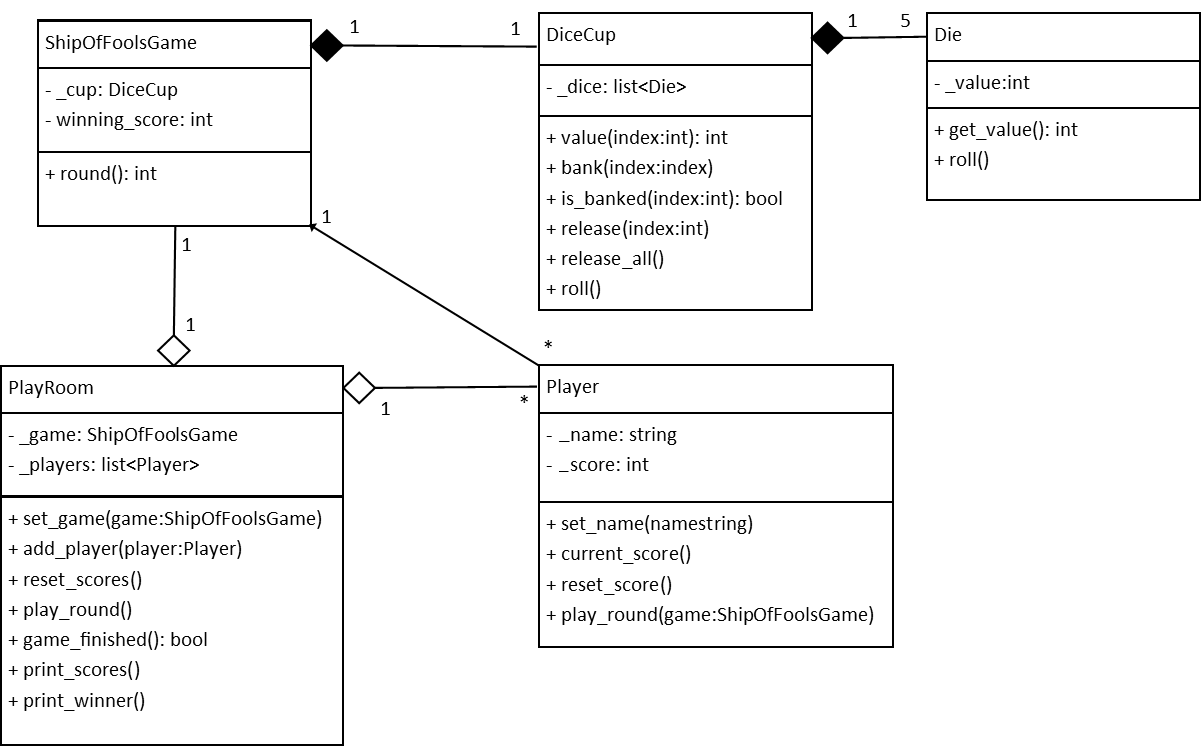
Description

Ship of Fools is a simple classic dice game. It is played with five standard 6-faced dice by two or more players. The game goal is to gather a 6, a 5 end a 4 (ship, captain and crew) in the mentioned order. The sum of the two remaining dice (cargo) is preferred as high as possible. The player with the highest cargo score wins the round.

**Example:** Each player gets to roll dices three times. If a player in the first round turn roll 6 4 3 3 1 the player can bank the 6 (ship), but the rest needs to be re-rolled since there is no 5. I the second round turn, if the player rolls 6 5 4 4 (four dice, since the 6 from last turn is banked) the player can bank the 5 (captain) and the 4 (crew). Now, the player has three choices of what to do with the remaining 6 and 4. The player can bank both and score 10 points or re-roll one or two of the dice and hope for a higher score. If the player in the second round turn instead  rolled 4 4 3 1 all dice needs to be re-rolled since there is no 5.

Model

Below is a suggestion how the game can be modeled by a set of classes:



The division of responsibility between the different classes is as follows.

**Die:** Responsible for handling randomly generated integer values between 1 and 6.

**DiceCup**: Handles five objects (dice) of class Die. Has the ability to bank and release dice individually.  Can also roll dice that are not banked.

**ShipOfFoolsGame**: Responsible for the game logic and has the ability to play a round of the game resulting in a score. Also has a property that tells what accumulated score results in a winning state, for example 21.

**Player**: Responsible for the score of the individual player. Has the ability, given a game logic, play a round of a game. The gained score is accumulated in the attribute score.

**PlayRoom**: Responsible for handling a number of players and a game. Every round the room lets each player play, and afterwards check if any player have reached the winning score.

Implementation

The algorithm for the game logic can be quite simple (pseudo code):

has\_ship = False  
has\_captain = False  
has\_crew = False  
  
# This will be the sum of the remaining dice, i.e., the score.  
crew = 0  
  
# Repeat three times  
for round in range(3):  
 roll unbanked dice  
 if not has\_ship and a dice is 6:  
 bank it  
 has\_ship = True  
 if has\_ship and not has\_captain and a dice is 5:  
 # A ship but not a captain is banked  
 bank it  
 has\_captain = True  
 if has\_captain and not has\_crew and a dice is 4:  
 # A ship and captain but not a crew is banked  
 bank it  
 has\_crew = True  
   
 if has\_ship and has\_captain and has\_crew:  
 # Now we got all needed dice, and can bank the ones we like to save.  
 bank all unbanked dice > 3  
  
# If we have a ship, captain and crew (sum 15),   
# calculate the sum of the two remaining.  
if has\_ship and has\_captain and has\_crew:  
 crew = sum of dice - 15

Main script

This can be the main script of the program.

**if** \_\_name\_\_ == "\_\_main\_\_":  
    room = PlayRoom()  
    room.set\_game(ShipOfFoolsGame())  
    room.add\_player(Player(”Ling”))  
    room.add\_player(Player(”Chang”))  
    room.reset\_scores()  
    **while not** room.game\_finished():  
 room.play\_round()  
 room.print\_scores()  
 room.print\_winner()

Hints

Many methods involves to iterate over a list that is an attribute of the class.  There are some general algorithms that we often want to implement when we program. Note that you can reach elements in two different ways. Either by getting a reference to one object after another, or by index. We start with two examples where we use the reference and then one where we use the index (which is needed in this case).

**Search (Pseudo code)**

# To set a value given through a parameter or a fixed value  
value\_to\_find = <something>  
# Set value to search for  
found = False  
# For each value of the attribute (presumed a list)  
for obj in self.attribute\_name:  
 # Asks the object for the value and compare  
   if obj.get\_value() == value\_to\_find  
 # Update the value  
        found = True

**Accumulate** (for example adding, or concatenating e.g. a string)

# Set initial value  
sum\_of\_values = 0  
# For each value of the attribute (presumed a list)  
for obj in self.attribute\_name:  
   # Asks the object for the value and add.  
   sum\_of\_values += obj.get\_value()

**Example**

If we want to check if a not banked die has the value 4, and if so change the value of a helper variable.

# Set initial value  
has\_crew = False  
# For each value of the attribute (presumed a list)  
for i in range(len(self.\_dice)):  
 if dice[i].get\_value() == 4:  
 dice[i].bank()  
 has\_crew = True

Your code shall fulfill the following:

* All classes in the given [UML diagram for Ship of Fools](https://bth.instructure.com/courses/3687/files/546620/download) should be implemented according to the model.   
  You may add functionality (attribute(s) and/or method(s)) to the classes if needed.
* Use the coding style discussed in the course regarding naming conventions.
* You should implement a main script, and the program should be able to play a full game until one player reaches the winning score.   
  A suggested main script can be found in [Ship of Fools](https://bth.instructure.com/courses/3687/pages/ship-of-fools).
* Do not zip the code files. Submit them as it is.