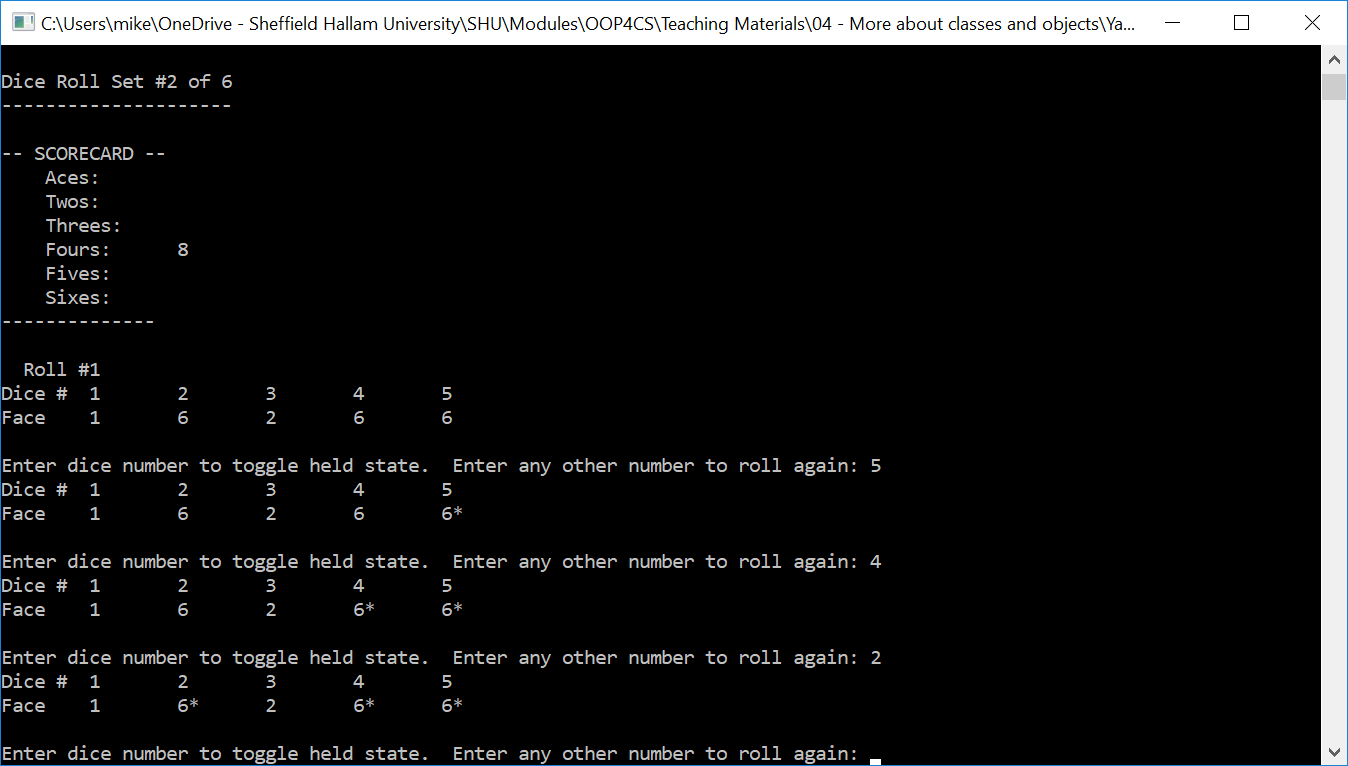
# More about Classes and Objects

This tutorial will give us more practice at creating classes and objects. We’ll consider the simple dice game Yahtzee.

Yahtzee score cardYou have 5 dice that you can roll up to three times per set to get a corresponding score against the scorecard (see right). Each time you roll the dice, you choose which dice you want to throw again (on the first throw, all 5 dice must be rolled, but the second and third rolls could roll fewer). Once you’ve thrown 3 times, you must score your dice against a free row on the scorecard.

You get a total of 13 sets of throws to complete all the rows of the scorecard. Once a row has been completed, you cannot override a score in the cell.

Using an object-oriented approach, we will implement this behaviour, starting with only the top 6 rows (and thus only 6 sets of throws).



If you wish to attempt to create this application yourself, here is an starter class diagram (constructors and some access attributes omitted) and the example application on Blackboard provides a solution to aim for. It is expected that you use best-practice OO techniques (including checking for memory leaks) though.

**Dice**

int getDiceFace()

bool isHeld()

void setHeld(bool)

void roll()

**Yahtzee**

+ void play() // main game function

- void playset()

- void displayOptions()

- void resetDice()

- void rollDice()

- void displayDice()

- void displayScorecard()

**- static** int getRangedInput(int, int)

uses

***However, if you want a little more guidance and semi-guided approach to this task, the following steps will help.***

**1)** Create yourself a new Empty (Console) Application in C++

**2)** Add a new source file that will contain the main method

#include <iostream>

#include <crtdbg.h>

using namespace std;

int main()

{

#ifdef \_DEBUG

// \_CrtSetBreakAlloc(87);

\_onexit(\_CrtDumpMemoryLeaks);

#endif

// start coding here

system("pause");

return 0;

}

**3)** We will need an object for each dice so let’s go ahead and create ourselves a new Dice class (use the Add Class wizard in Visual Studio if that helps).

Each Dice object will need:

int diceface; // what dice face is on display

bool held; // is this dice being held and thus it'll not be rolled

Add these attributes into the appropriate place of the class declaration (header file) – are they private, protected or public?

You will also need to add methods to get and mutate these attributes:

int getDiceFace() const; // gets the current dice face

bool isHeld() const; // returns true is the dice is currently held

void setHeld(const bool held); // sets whether the dice is held from being rolled

void roll(); // rolls a new random dice number (only if not held)

Add these member functions into the appropriate place of the class declaration and provide an implement of each – think about whether it should be inlined or not and thus whether the implementation will go in the header (.h) or source (.cpp) file. Are these methods private, protected or public?

Provide an implementation for the constructor – use initialisation lists to provide sensible values to the Dice object’s variables.

**4)** Test the Dice class. You can use the following code or write your own (don’t forget to include Dice.h within the main cpp file):

// start coding here

srand((unsigned int) time(NULL));

Dice dice;

dice.roll();

cout << "First dice face\n";

cout << dice.getDiceFace() << "\n";

cout << "Held\n";

dice.setHeld(true);

for (int i=0; i<10; i++)

{

dice.roll();

cout << " " << dice.getDiceFace() << "\n";

}

cout << "Unheld\n";

dice.setHeld(false);

for (int i=0; i<10; i++)

{

dice.roll();

cout << " " << dice.getDiceFace() << "\n";

}

Notice how I called srand within the main method – you only need to call it once somewhere at the start of the program, although this perhaps isn’t the best place for it.

**5)** Create yourself a set of Yahtzee class files. The Yahtzee class will contain the game behaviour and make use of the Dice class. The entire game will be launched and executed via a single method call of the Yahtzee object that we will develop over the course of the following steps.

**6)** Declare a private array of 5 Dice objects within the Yahtzee class using stack memory. These will be non-static data members of the class.

**7)** To represent the scorecard, we’ll introduce a few constants. These are static constants because there is no need for each object to have their own copy. Add the following lines to your Yahtzee class definition:

static const int SET\_ACES=0; // constant for aces row (also index into our scores array)

static const int SET\_TWOS=1; // constant for twos row (also index into our scores array)

static const int SET\_THREES=2;// constant for threes row (also index into our scores array)

static const int SET\_FOURS=3;// constant for fours row (also index into our scores array)

static const int SET\_FIVES=4;// constant for fives row (also index into our scores array)

static const int SET\_SIXES=5;// constant for sixes row (also index into our scores array)

static const int NOOFSETS=6; // the number of possible scorecard rows

static const int SCORE\_UNSET=-1; // this value is used to indicate a free scorecard row

static const char\* SCORE\_LABELS[NOOFSETS];

Should they be public, private or protected? (remember, we are focusing on just the upper-half of the score card for the moment)

**8)** The static SCORE\_LABELS array declared above is used during the display methods to help wrap everything up into for-loops. The text for this needs to be defined in the source file (.cpp). Add the following line to the Yahtzee.cpp file:

const char\* Yahtzee::SCORE\_LABELS[NOOFSETS]={"Aces", "Twos", "Threes", "Fours", "Fives", "Sixes"};

There is no need to declare it static again in the source file, although it still needs to be hocked onto the Yahtzee class. This is how you declare and create static data members.

**9)** We’ll use an array to store the scores for each scorecard row. Add another array data member to the Yahtzee class, which will be used to store the scores (what type should it be?). Use the NOOFSETS constant to define the array size (defined above as part of step7).

**10)** Implement the Yahtzee constructor. The scores array should be initialised with the SCORE\_UNSET constant for each cell. You’ll not be able to do this using an initialisation list. You should also perhaps think about calling srand here too.

**11)** We will need to get ranged input from the user to decide which dice to hold and which scorecard row to score against. Declare a static Yahtzee member function in the header file that all objects can make use – the following implementation can be used in the source file, but you still need to prototype it in the class declaration – make sure it is static.

int Yahtzee::getRangedInput(int min, int max)

{

int value;

cin >> value;

while (value<=min || value>=max)

{

cout << "Valid range is " << min << " to " << max << " (inclusive). Try again: ";

cin >> value;

}

return value;

}

**12)** Over to you now… I have provided an executable of the program which you can run and observe the behaviour. Reproduce this (use inline functions where appropriate) and create the basic Yahtzee game. In addition to those methods already considered, I have the following Yahtzee member functions:

public:

void play();

private:

void playSet();

void displayOptions();

void resetDice();

void rollDice();

void displayDice();

void displayScorecard();

Here is the implementation of my play method:

void Yahtzee::play()

{

for (int i=0; i<NOOFSETS; i++)

{

cout << "Dice Roll Set #" << (i+1) << " of " << NOOFSETS << "\n";

cout << "---------------------\n\n";

displayScorecard();

playSet();

}

displayScorecard();

int total=0;

for (int i=0; i<NOOFSETS; i++)

total+=scores[i];

cout << "Total: " << total << "\n\n";

}

I have copied my method implementations at the end of the document, but try to code the rest yourself, only taking a sneaky peek at my program for guidance (remember, there isn’t a single unique solution).

**13)** Update the game play such that the entire scorecard can be used – upper and lower half as well as the upper-half bonus.

void Yahtzee::playSet()

{

resetDice();

for (int i=0; i<3; i++)

{

cout << " Roll #" << (i+1) << "\n";

rollDice();

int value;

do

{

displayDice();

if (i==2)

{

displayOptions();

cout << "Score against: ";

value=getRangedInput(1, NOOFSETS);

while (scores[value-1]!=SCORE\_UNSET)

{

cout << "Score row already taken. Please choose another: ";

value=getRangedInput(1, NOOFSETS);

}

int score=0;

for (int d=0; d<5; d++)

// value represents the dice face being used to score this set

if (dice[d].getDiceFace()==value)

score+=value;

scores[value-1]=score;

cout << "\n\n";

break;

}

else

{

cout << "\nEnter dice number to toggle held state. Any other number to roll again:";

// dice numbers are presented to the user as 1..6, but we want indices 0..5 so minus 1.

value=getRangedInput(1, 6)-1;

// use the XOR operator to toggle between true and false (1^1=0 / 0^1 = 1)

dice[value].setHeld(dice[value].isHeld()^true);

}

} while (value>=0 && value<=5);

}

}

int Yahtzee::getRangedInput(int min, int max)

{

int value;

cin >> value;

while (value<=min && value>=max)

{

cout << "Valid range is " << min << " to " << max << " (inclusive). Try again: ";

cin >> value;

}

return value;

}

void Yahtzee::displayOptions()

{

// only those those scorecard rows that have yet to be used

cout << "\nPlease choose how to score your dice\n";

for (int i=0; i<NOOFSETS; i++)

if (scores[i]==SCORE\_UNSET)

cout << (i+1) << ":\t" << SCORE\_LABELS[i] << "\n";

}

void Yahtzee::displayScorecard()

{

cout << "-- SCORECARD -- \n";

for (int i=0; i<NOOFSETS; i++)

{

cout << " " << SCORE\_LABELS[i] << ":\t";

if (scores[i]!=SCORE\_UNSET)

cout << scores[i];

cout << "\n";

}

cout << "--------------\n\n";

}

void Yahtzee::displayDice()

{

cout << "Dice #\t";

for (int i=0; i<5; i++)

cout << (i+1) << "\t";

cout << "\nFace\t";

for (int i=0; i<5; i++)

{

cout << dice[i].getDiceFace();

if (dice[i].isHeld())

cout << "\*"; // use a \* character to indicate the dice is held

cout << "\t";

}

cout << "\n";

}

void Yahtzee::resetDice()

{

for (int i=0; i<5; i++)

dice[i].setHeld(false);

}

void Yahtzee::rollDice()

{

for (int i=0; i<5; i++)

dice[i].roll();

}