DSED C# Classes Assessment 2015

Create a Program that runs a race and allows betting on the outcome – (Not a Dog race).

This assessment is due ------

Instructions:

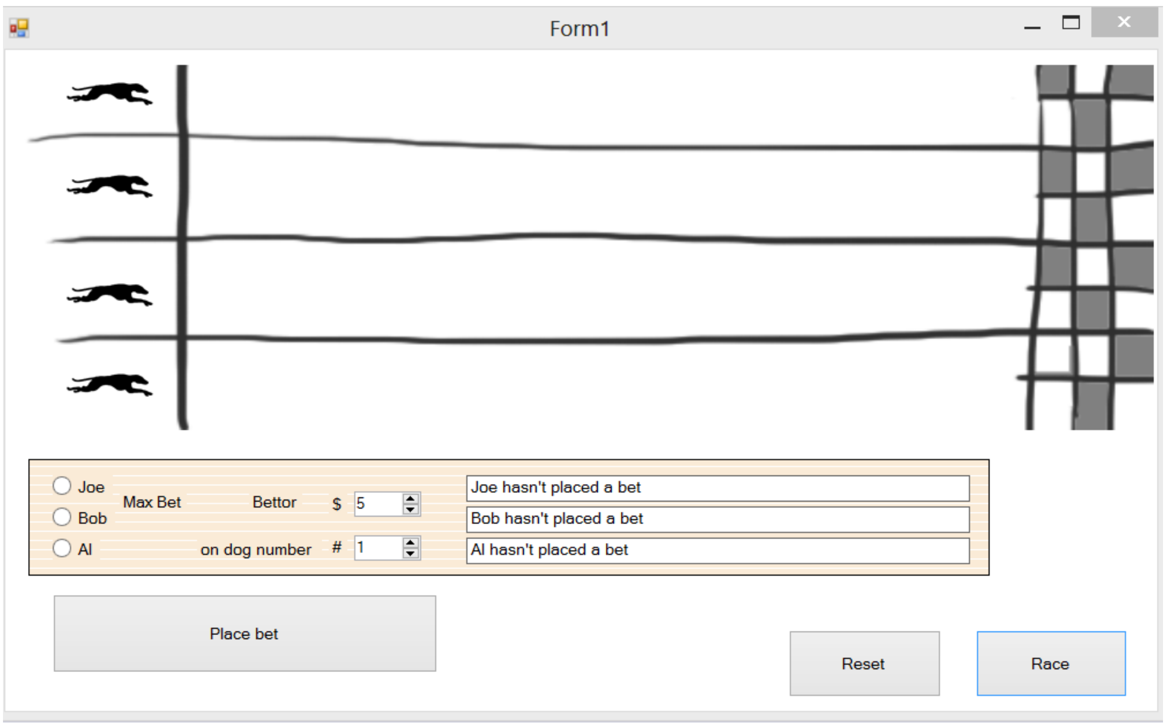
- The assessment is an open book exercise – students may consult with others, but finally must present their own work
- Your code files and your documentation files will be put in your named folder on the tutor computer

1) Complete the following specifications:

   a) **Context:** There are two parts to this program, the first is to create a 4 race using random numbers and picture boxes for each racer. The second part of the program is to create a betting system that is based on the outcome of the race.

   b) Each bettor is given $50 to bet with, the program has to add and subtract from that original bet until the money has gone. The Max Bet label says how much money the bettor has to spend, and the Up/Down control only goes to that limit, so you can't spend more than you have.

   c) The game ends when everyone has lost their money, or there is only one bettor left.



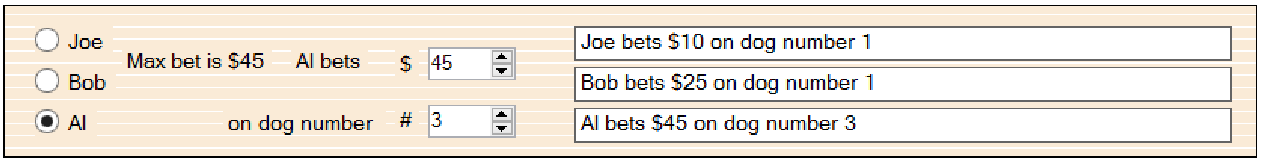
Joe hasn't placed a bet

Bob hasn't placed a bet

Al hasn't placed a bet

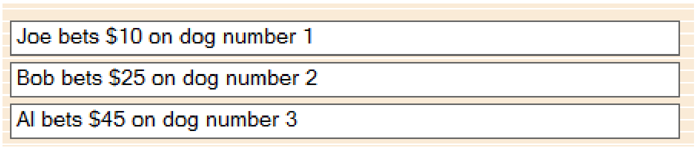
d) **The bettors must show**

- The maximum amount that can be bet for each bettor in a label
- The Up/down box can only go to that maximum number for each bettor. (i.e.: Al's max bet is $45)

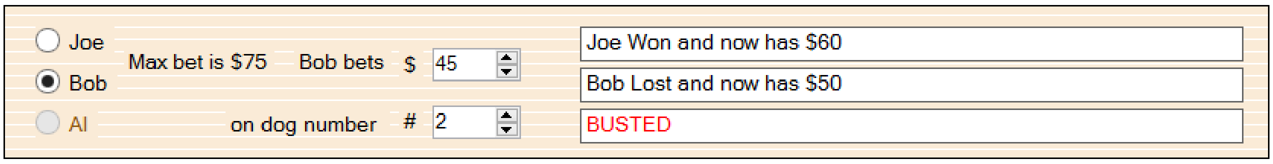

( ) Joe
( ) Bob
(●) Al
Max bet is $45 Al bets $ 45
on dog number # 3

Joe bets $10 on dog number 1
Bob bets $25 on dog number 1
Al bets $45 on dog number 3

- When the Bet is laid the Name, Amount, and Dog appear on the right

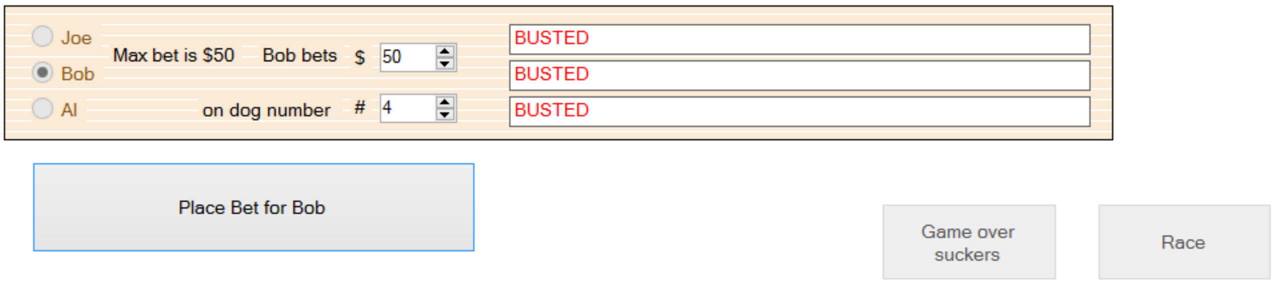

Joe bets $10 on dog number 1
Bob bets $25 on dog number 2
Al bets $45 on dog number 3

- When a person is out of money, **they cannot bet again** (in this case the radio button is inactive), and Busted appears


( ) Joe
(●) Bob
( ) Al
Max bet is $75 Bob bets $ 45
on dog number # 2

Joe Won and now has $60
Bob Lost and now has $50
BUSTED

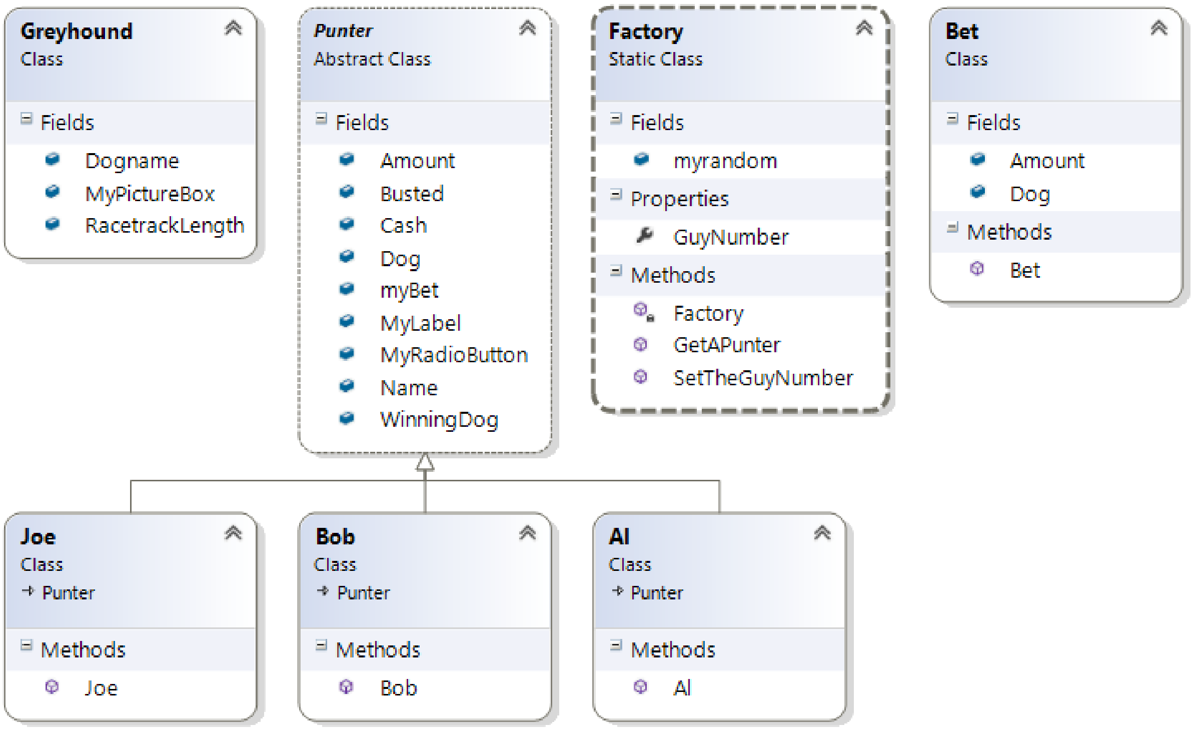
- When all the bettors lose (which they eventually will) the game is over


( ) Joe
(●) Bob
( ) Al
Max bet is $50 Bob bets $ 50
on dog number # 4

BUSTED
BUSTED
BUSTED

You must use the **Factory Design Pattern**, and an **Abstract Class** for the Punters. The 3 punters are to inherit from the Punter class.

On the main form instantiate as an array `Punter[] myguy = new Punter[3];`



1) Create at least **1 Unit Test** of your project using Instantiation and linking back to the class.

Write a log of your work

Comment your code thoroughly explaining what each major portion does

Form features

- Data entry text boxes or listboxes, and labels
- Buttons or Radio buttons or any other clickable event to manipulate data
- Adequate signage and titles to make it easy to understand.

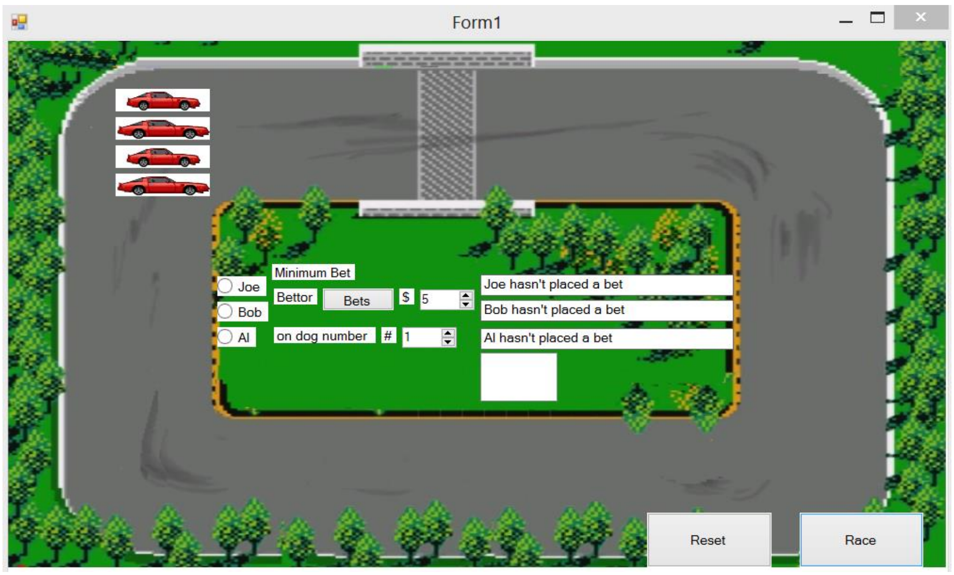
Hints.

The exercise uses Classes and Arrays they are the heart of the program and make the code quite short.

Look at the book **C# Head First** for help to make your program.

Look at the exe in the Goodies folder and see how it works.

DO NOT make your program look like a bunch of Greyhounds as these images show above. Use your imagination and Coding Ninja skills to change it into something far more entertaining and surprise us.



Joe hasn't placed a bet

Bob hasn't placed a bet

Al hasn't placed a bet

This is my adaption, the cars go around the track.

Beat that …..

Marking Schedule

Form features

1.1 The maximum amount that can be bet for each bettor in a label
1.2 The Up/down box can only go to that maximum number for each bettor
1.3 When the Bet is laid the Name, Amount, and Dog appear on the right
1.4 When a person is out of money, **they cannot bet again**
1.5 When all the bettors lose (which they eventually will) the game is over

Class Operations

2.1 Two classes Greyhound and Bet (Bet is optional)
2.2 An Abstract Punter class
2.3 Three inherited bettor classes

Unit test

3.1 A unit test using Instantiation

% of Grade	Excellent 100%	Adequate 80%	Poor 60%	Not Met 0%
Program Specifications / Correctness				
50%	No errors, program always works correctly and meets the specification.	Minor details of the program specification are violated, program functions incorrectly for some inputs.	Significant details of the specification are violated, program often exhibits incorrect behavior.	Program only functions correctly in very limited cases or not at all.
Mark	50	40	30	0
Readability				
20%	No errors, code is clean, understandable, and well-organized.	Minor issues with layout, variable naming, or general organization.	At least one major issue with layout, variable names, or organization.	Major problems with at three or four of the readability subcategories.
Mark	20	16	12	0
Documentation				
20%	No errors, code is well-commented.	One or two places that could benefit from comments are missing them **or** the code is *overly* commented.	Complicated lines or sections of code uncommented or lacking meaningful comments.	No comments present.
Mark	20	16	12	0
Code Efficiency				
5%	No errors, code uses the best approach in every case.	*N/A*	Code uses poorly-chosen approaches in at least one place.	Many things in the code could have been accomplished in an easier, faster, or otherwise better fashion.
Mark	5	4	3	0
Assignment Specifications				
5%	No errors	*N/A*	Minor details of the assignment specification are violated, such as files named incorrectly or extra instructions slightly misunderstood.	Significant details of the specification are violated, such as extra instructions ignored or entirely misunderstood.
Mark	5	4	3	0