# Diploma in Software Engineering and Design

# Assignment Cover Sheet

|  |  |
| --- | --- |
| Course name: Diploma in Software Engineering and Design | **Student’s name:**  Kyle Harvey |
| **Module Name or Number :** Object Oriented Programming (15 credits) | |
| **Assignment title and/or number**: Object Oriented Assessment | |
| **Passing Criteria** | Need to score 50 or more marks to pass the assessment.  **Total Marks : 100** |
| **Assessment weighting** | *12.5% of the overall programme.* |
| **Due date**: **2nd June 2017** | **Date submitted**:  (late submissions incur 10% penalty, after 7 days late, the assessment will not be marked) |
| **Assessment conditions:** | This is a resource-based assessment. This means that you may have access to any relevant resources to assist you. This could include, for example, your learning materials, information on the Internet, and so on. However, all work must be your own with no assistance from any other person. |
| **Submission requirements:** | You’re required to submit the following into your assignment submission directory:   * This document, completed where appropriate * Visual Studio project files * Github link to your project   GitHub link below:  github.com/SumOfAllN00bs/TheBettingGame |
| **Learning Outcomes** | * Demonstrate understanding of Object Oriented Programming concepts including  1. Classes and objects 2. Constructors 3. Encapsulation 4. Inheritance 5. Inheritance based polymorphism 6. Class Diagrams 7. Static polymorphism 8. Dynamic Polymorphism |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Assignment Checklist:** | |  |  | | --- | --- | | **Requirement** | **Completed** | | Database | ✓🗶 | | User interface | ✓🗶 | | Functionality | ✓🗶 | | Coding | ✓🗶 | | Testing | ✓🗶 | | Assessment Sheet | ✓🗶 | |

# Disclaimer of Plagiarism and Collusion

I declare that, to the best of my knowledge, this assessment is my own work, and has not been copied from any other student's work or from any other source.

Enter your name here to indicate you agree to the above statement.

|  |
| --- |
| Kyle Harvey |

# DSED C# Classes Assessment

## Create a Program that runs a race and allows betting on the outcome –

## Instructions:

1. Complete the following specifications:
   1. **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.
   2. 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.
   3. Every bet is double-or-nothing—either the winner doubles his money, or he loses what he bet. There’s a minimum bet of 5 bucks, and each guy can bet up to 15 bucks on a single dog. If the dog wins, the bettor ends up with twice the amount that he bets (after the race is complete). If he loses, that amount disappears from his pile.
   4. The game ends when everyone has lost their money, or there is only one bettor left.
   5. **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)
* When the Bet is laid the Name, Amount, and Dog appear on the right
* When a person is out of money, **they cannot bet again** (in this case the radio button is inactive), and Busted appears
* When all the bettors lose (which they eventually will) the game is over

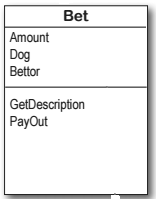
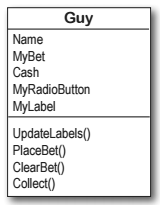
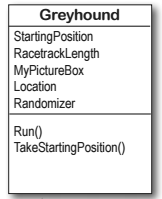


## Coding Requirements

You need to use an OOP approach to solve this example, you will **not pass the assessment if you do not satisfy this requirement i.e. your game logic should not be on the UI form or UI class.**

At the minimum you need to have three classes for the Dog, the bettors and the actual bet that is placed. However, you are free to use other OOP concepts like inheritance, polymorphism etc.

For example, have a look at the sample classes below



## Requirements

|  |  |
| --- | --- |
| Functionality | |
| 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 |
| 1.6 | Generate your class diagram and paste it in this document |
| 1.7 | Create a test log of all the tests performed and submit it with your application |
|  | |
|  |  |
|  |  |
|  |  |
|  | |
|  |  |



**Marking Schedule**

|  |  |
| --- | --- |
| **Parameters** | **Marks** |
| **User Interface** | **20** |
| **Functionality** | **30** |
| **Coding** | **40** |
| **Testing** | **10** |

Detailed Marking Schedule

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **% of Grade** | **Excellent 80 - 100%** | **Adequate 60 - 80%** | **Poor 50 - 60%** | **Not Met 0 - 50%** |
| User Interface  20% | Simple easy to use intuitive UI, no errors spelling mistake and good color schemes used | Minor errors with the UI, minor layout issues | Major UI errors making it hard to understand. High possibility of human errors. | Significant UI errors with no logical sense and frequent UI issues |
| Functionality  30% | 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 |
| Coding  40% | No errors, code uses the best approach in every case and follows the coding standards | Minor errors or repetition of code, coding and naming standards not followed in some occasions | Code uses poorly-chosen approaches in some places. Naming standards not followed in some places. | Many things in the code could have been accomplished in an easier, faster, or otherwise better fashion. Poor naming and coding standards |
| Testing  10% | Program is well tested to identify and fix bugs and errors. No major bugs or defects in the program. Testing results matches the actual program. | Program is well tested to identify most of the bugs, but some bugs still exist.  Some test cases marked pass which are false | Program has a lot of major bugs and is not tested. Testing sheet incorrect or incompletely filled out. | Program is not at all tested and testing sheet is not filled. |