**7 PDA: Software Development**

**Level 8 - Student Evidence Checklist**

|  |  |  |
| --- | --- | --- |
| **Full name** | **Peter McCready** | **Key:** A & D - Analysis and Design Unit  I & T - Implementation and Testing Unit  P - Project Unit |
| **Cohort** | **G3** |

The evidence required can be taken from your assignments, homework that you have completed on your own or by creating a specific example for the PDA.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Week 2** | **Unit** | **Ref.** | **Evidence** | **Done** |
| I & T | I.T 5 | Demonstrate the use of an array in a program. Take screenshots of:   * An array in a program   As part of Test Driven Development (TDD) I used an array within the assert\_equal function as the expected result of running the test\_array\_of\_capitals method.     * A function that uses the array   The array\_of\_capitals function used the capitals array, used in conjunction with country hashes in the TDD above. The empty capitals array is populated with the capitals of each country when the function is run.     * The result of the function running   When the function is run the value of ‘actual’ is set to the value held against the captial key in the country hash, for each country. |  |
| I & T | I.T 6 | Demonstrate the use of a hash in a program. Take screenshots of:   * A hash in a program   The @customers array holds each customer as a hash containing name, pets and cash keys with their associated values.     * A function that uses the hash   The function below sells a pet to the first customer (Craig) held in the customers array above. A find method is used to locate a specific pet and the sell\_pet\_to\_customer function is used to move that pet into the customer’s pets array which is held in the customer’s hash. The test below then checks to ensure the length of the pets array in the customer’s hash has increased to 1, checks that the pets\_sold by the pet\_shop is increased to 1 and checks the total cash of the pet\_shop has increased by the value of the sold pet (from 1000 to 1900, an increase of 900, the price of Arthur the Husky).     * The result of the function running   When the function is run the pet object is added to the customer hash’s pets array. |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Week 3** | **Unit** | **Ref.** | **Evidence** | **Done** |
| I & T | I.T 3 | Demonstrate searching data in a program. Take screenshots of:   * Function that searches data   Combining the SqlRunner and self.find() method of the Artist class it was possible to find a specific instance of the Artist class. This was done by calling the method against the class in Pry-Byebug.       * The result of the function running   When the function was run, passing in the id of the Artist I was trying to find (27) the artist with that id is returned.    This can be cross-referenced with the artists table of the music database to confirm the correct data is returned: |  |
| I & T | I.T 4 | Demonstrate sorting data in a program. Take screenshots of:   * Function that sorts data   Combining the SqlRunner and self.all() method of the Album class it was possible to find all instances of that class. This was done by calling the method against the class in Pry-Byebug. ORDER BY was used within the SQL to sort the returned dataset.       * The result of the function running     To confirm that sorting has taken place this can be confirmed with checking against the entries held in the album table of the music database: |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Week 5** | **Unit** | **Ref.** | **Evidence** | **Done** |
| A & D | A.D 1 | A Use Case Diagram |  |
| A & D | A.D 2 | A Class diagram. |  |
| A & D | A.D 3 | An Object diagram. |  |
| A & D | A.D 4 | An Activity Diagram |  |
| A & D | A.D 6 | Produce an Implementations Constraints plan detailing the following factors:   * Hardware and software platforms * Performance requirements * Persistent storage and transactions * Usability * Budgets * Time |  |
| P | P 5 | Create a user sitemap. |  |
| P | P 6 | Produce two wireframe designs. |  |
| P | P 10 | Take a screenshot of an example of pseudocode for a function. |  |
| P | P 13 | Show user input being processed according to design requirements. Take a screenshot of:   * The user inputting something into your program        * The user input being saved or used in some way |  |
| P | P 14 | Show an interaction with data persistence. Take a screenshot of:   * Data being inputted into your program   Changing the pricing details for Alter Bridge The Last Hero album:  Original values on edit album:    Changed ‘Buy Price’, ‘Discount’ and ‘Initial Sell Price’ before hitting ‘Amend Details’:    Note: ‘Final Sell Price’ and ‘Profit Margin’ are calculated after the ‘Amend Details’ button is clicked so is changed on the next page, see next screenshot below.   * Confirmation of the data being saved   Changed data saved and the ‘Final Sell Price’ and ‘Profit Margin’ calculated and displayed by the system: |  |
| P | P 15 | Show the correct output of results and feedback to user. Take a screenshot of:   * The user requesting information or an action to be performed   Album shown in list of Artists/Albums (All Stock) and request to ‘Delete Album’ (Dead Kennedys, Give me death):       * The user request being processed correctly and demonstrated in the program   Deleted Album removed from list of Artists/Albums (All Stock): |  |
| I & T |  | Coding exercise 1: Static and Dynamic testing task A  Uploaded to GitHub in Static and Dynamic Testing Folder. |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Week 6** | **Unit** | **Ref.** | **Evidence** | **Done** |
| I & T | I.T 7 | Demonstrate the use of Polymorphism in a program.    The Playable interface is implemented by the Player class. The dealer class extends Player. This allows both the Player and Dealer to be passed into the Game as playables. The game stores an ArrayList of Playables (Dealer and Player) in this case.      The Playable interface allows the Dealer to be both a Dealer and a blackjack player in this program, displaying polymorphism. |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Week 9** | **Unit** | **Ref.** | **Evidence** | **Done** |
| A & D | A.D 5 | An Inheritance Diagram    Dealer inherits from Player. This allows the Dealer class to make use of the Player functions.  Both Player and Dealer implement the Playable interface which enforces both Classes to implement the interface functions. |  |
| I & T | I.T 1 | Take a screenshot of an example of encapsulation in a program.    In this case the private instance variables are encapsulated as they are only accessible within the class. If trying to access these ‘hidden’ variables outside of the Class definition the appropriate getter must be used. E.g. Card.getName( ); |  |
| I & T | I.T 2 | Take a screenshot of the use of Inheritance in a program. Take screenshots of:   * A Class      * A Class that inherits from the previous class      * An Object in the inherited class     The Dealer class is instantiated with a name (String) which is then passed to the Player superclass using Super for construction. As the Dealer class inherits from the Player class it has access to it’s own numWins variable. The following passing test (hasName) shows that the Dealer has a name when instantiated despite not explicitly possessing it as a property (as it inherits this from Player).     * A Method that uses the information inherited from another class.     The cpuNameText and cpuWinsText are set to the dealer.getName() and dealer.getNumWins() respectively, which are functions that access instance variables on the Player superclass. Dealer is therefore able to access these as it inherits from Player. |  |
| P | P 11 | Take a screenshot of one of your projects where you have worked alone and attach the Github link.      <https://github.com/PMGH/rick_and_morty_top_trumps> |  |
| P | P 12 | Take screenshots or photos of your planning and the different stages of development to show changes.    INITIAL    FINAL  **App Map** – this initial map changed as a database was no longer deemed a requirement. The home page became a Splash page and the Rules page became the home page. A Play Again activity was used to redirect the player to another game while maintaining data. The Database was deemed unnecessary for the MVP so the option to open old game sessions was not implemented but was left as and option for enhancement.    INITIAL    FINAL  **Class Diagram** – the class diagram was largely adhered to for most classes but additional functions were needed for the Playable interface. The Game logic was much more complex than initially anticipated so contained many more functions than initially planned. Additional functions are highlighted green. Functions highlighted red were removed.      **Wireframes** – these were used in the early stages of development to give a blueprint to work to. Wireframes were also sketched in a notebook at various steps as the product evolved. The final wireframe also included a Play Again activity which was used to notify the user that the game had finished and gave the option to play again. Data persistence through a Singleton game class was used to increment number of game wins and update the number of cards in the user and CPU hands. On completion, the interface looked slightly different from the wireframes with some additional details but the general shape of the pages were maintained. |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Week 11** | **Unit** | **Ref.** | **Evidence** | **Done** |
| P | P 18 | Demonstrate testing in your program. Take screenshots of:   * Example of test code     The Java code above runs tests to check that items can be added/removed from a Shopping Basket object.    The java code above is the Shopping Basket class with empty add and remove functions, causing the failing tests.   * The test code failing to pass      * Example of the test code once errors have been corrected     The java code above shows the add and remove functions complete and capable of passing the tests.   * The test code passing |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Week 12** | **Unit** | **Ref.** | **Evidence** | **Done** |
| P | P 16 | Show an API being used within your program. Take a screenshot of:   * The code that uses or implements the API     The JavaScript code above takes the data from the RestCountries API (<https://restcountries.eu/rest/v2/all>) and creates charts from it using the Highcharts API (<http://code.highcharts.com/highcharts.js>).   * The API being used by the program whilst running |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Week 14** | **Unit** | **Ref.** | **Evidence** | **Done** |
| P | P 1 | Take a screenshot of the contributor’s page on Github from your group project to show the team you worked with. |  |
| P | P 2 | Take a screenshot of the project brief from your group project.      The project that my group and I decided to do was based around the two project briefs shown above. Our application allows the selection of start (user geolocation) and finish locations (venue) for direction routes using the GoogleMaps Directions API. Alternative to user geolocation the user can search for a location using a GoogleMaps search bar which utilises Google’s places library to offer predictions as the user types. The GoogleMaps Distance Matrix API is also used to give route distance and duration details which are then displayed on the map within an associated route infoWindow. The GoogleMap within the application also plots markers with an associated infoWindow for each bar and distillery held within the NoSQL database collections. |  |
| P | P 3 | Provide a screenshot of the planning you completed during your group project, e.g. Trello MOSCOW board. |  |
| P | P 4 | Write an acceptance criteria and test plan. |  |
| P | P 7 | Produce two system interaction diagrams (sequence and/or collaboration diagrams). |  |
| P | P 8 | Produce two object diagrams. |  |
| P | P 9 | Select two algorithms you have written (NOT the group project). Take a screenshot of each and write a short statement on why you have chosen to use those algorithms.  1)  The screenshot above shows functionality within my Snow Resorts app.js file which creates a marker on a Google Map for each ski area returned by the Ski Map API. The reason I’ve chosen this is that the Ski Map data was incomplete in some cases as certain resorts were not assigned to a region (i.e undefined) so caused a problem when creating new map markers. I used a ternary operator to check if the current iteration of the forEach loop (Ski Area) had an undefined region. If so, I assigned the name key of the area.region object to “No region data” to prevent the subsequent code of creating a new skiArea object from failing. The new skiArea object was then passed as an argument to the mainMap.addMarker() function in order to create a new marker on the map for the Ski Area.  2)  The screenshot above shows functionality within the Game class of my Top Trumps Game project. The reason I have chosen this is that it was necessary in my project to choose the best category of the CPU player’s card in order to make each round of the game competitive. It gets the values of each category (key) of the CPU (Dealer) card and adds them to a hashmap (categoryValues). The function then loops through the categoryValue values and determines the highest value, adding it to the cpuBestCategory array. If values are equal then they are both added to the cpuBestCategory array. The function returns the value in the cpuBestCategory array. If there is more than one value in the cpuBestCategory array then it returns one of the values randomly. |  |
| P | P 17 | Produce a bug tracking report |  |
| I & T |  | Coding Exercise 2: Unit and Integration testing task B |  |