**Familiarity Review**

**Name:** Jeremy James Alkire

**Date:** February 6th, 2019

**Week:** 5

**Coding Topic:** MVC

**Description of Understanding:** The MVC is a three-part architectural design pattern used for developing user interfaces. The three components are the model, view, and controller. The model manages the data and logic of the application. The view displays a visual representation of the data. It handles both input and output and is what the user interacts with. The model and view components do not communicate with each other directly, they are both managed by the controller, which acts as a medium between the two. The controller is essentially the engine of the program. It calls methods from the view and the model and passes data between them as necessary.

**Teaching Video:**  None.

**Starting at:** N/A

**Also Integrated with:** Use Case Diagrams, Use Case Documents, Sequence Diagrams

|  |  |  |  |
| --- | --- | --- | --- |
| **File** | **Git Link** | **What should I be looking for?** | **Sandbox or Your code?** |
| RequestData.java | <https://github.com/alkire-jeremy/CIT360/blob/master/Familiarity%20Requests/Request%201%20-%20Week%204/Code/fr1/view/RequestData.java> | This class operates as the **view** within the MVC architecture. The method *requestAthleteData()* outputs a prompt to the user that requests that they input da ta into the console. That data is handled and stored in method variables and returned to the method caller. It functions as a text-based UI. By itself, it does nothing. The controller calls its functions. | My code. |
| Athlete.java | <https://github.com/alkire-jeremy/CIT360/blob/master/Familiarity%20Requests/Request%201%20-%20Week%204/Code/fr1/model/Athlete.java> | This class operates as the **model** within the MVC architecture. It specifies the attributes of the Athlete object. It has private non-static variables, public getters & setters, and a universal configuration method *configureAthlete()* that allows you to set the value of all variables with a single function call (and three arguments.) | My code. |
| Engine.java | <https://github.com/alkire-jeremy/CIT360/tree/master/Familiarity%20Requests/Request%201%20-%20Week%204/Code/fr1/controller> | This class operates as the **controller** within the MVC architecture. It creates an object instance of the Athlete (model) class, calls *requestAthleteData()* from the RequestData (view) class and stores its output in a string variable. It then splits the string into a list of strings by comma (,) characters. From there, it calls *configureAthlete()* from the model, and passes each element in the list to model’s method via its parameters. | My code. |

**Coding Topic:** Java Collections

**Description of Understanding:** A collection, sometimes called a container, is simply an object that groups multiple elements into a single unit. Collections are used to store, retrieve, manipulate, and communicate aggregate data. These typically represent items that form a natural group, such as telephone directories. A collections framework is a unified architecture for manipulating collections. Collection frameworks contain interfaces, implementations, and algorithms.  
  
Interfaces are abstract data types that represent collections. These allow collections to be manipulated independently of the details of their representation, and generally form a hierarchy. Nearly all interfaces inherit methods from the Collection superclass and pass these methods down to their implementations. Core interfaces in Java include: Collection, Set, List, Queue, Deque, Map, Sorted Set, and Sorted Map.

Implementations are concrete implementations of the collection interfaces. They are reusable data structures, and children of the interfaces, whom are their parents, and whom they inherit methods from. General-purpose implementations in Java include: HashSet, HashMap, ArrayList, ArrayDeque, TreeSet, TreeMap, LinkedList, LinkedHashSet, and LinkedHashMap.  
  
Algorithms are methods that perform useful computations, such as searching, sorting, inserting, and deleting elements inside of a collection. They’re described as polymorphic, meaning they’re able to be used in many different implementations. These are typically implemented via methods, such as add() and put().

**Teaching Video:**  <https://youtu.be/S-1a0m8Fi9Q>

**Starting at:** 0:00

**Also Integrated with:** ArrayList is used briefly in my controller class (Engine.java) in my MVC coding examples.

|  |  |  |  |
| --- | --- | --- | --- |
| **File** | **Git Link** | **What should I be looking for?** | **Sandbox or Your code?** |
| ArrayListExample.java | <https://github.com/alkire-jeremy/CIT360/blob/master/Familiarity%20Requests/Request%201%20-%20Week%204/Code/collections/ArrayListExample.java> | This class utilizes the ArrayList implementation of the List interface. | My code. |
| CollectionExample.java | <https://github.com/alkire-jeremy/CIT360/blob/master/Familiarity%20Requests/Request%201%20-%20Week%204/Code/collections/CollectionExample.java> | This class also utilizes the ArrayList implementation of the List interface. | My code. |
| HashMapExample.java | <https://github.com/alkire-jeremy/CIT360/blob/master/Familiarity%20Requests/Request%201%20-%20Week%204/Code/collections/HashMapExample.java> | This class utilizes the HashMap implementation of the Map interface. | My code. |
| LinkedListExample.java | <https://github.com/alkire-jeremy/CIT360/blob/master/Familiarity%20Requests/Request%201%20-%20Week%204/Code/collections/LinkedListExample.java> | This class utilizes the LinkedList implementation of the List and Queue interfaces. | My code. |
| TreeMapExample.java | <https://github.com/alkire-jeremy/CIT360/blob/master/Familiarity%20Requests/Request%201%20-%20Week%204/Code/collections/TreeMapExample.java> | This class utilizes the TreeMap implementation of the Map interface. | My code. |
| HashSetExample.java | <https://github.com/alkire-jeremy/CIT360/blob/master/Familiarity%20Requests/Request%201%20-%20Week%204/Code/collections/HashSetExample.java> | This class utilizes the HashSet implementation of the Set interface. | My code. |

**Diagram**: Sequence Diagram

**Description of Understanding:** A sequence diagram demonstrates the interaction between messages and objects within a system. It displays these relationships in a sequential manner, so that people can clearly see how messages flow throughout the system. Components of a sequence diagram include objects, messages, and sequence numbers, which indicate how methods are called one after another.

**Teaching Video:** None

|  |  |  |
| --- | --- | --- |
| **File** | **What should I be looking for?** | **Example**  **Or Your code?** |
| <https://github.com/alkire-jeremy/CIT360/blob/master/Familiarity%20Requests/Request%201%20-%20Week%204/Diagrams/Sequence%20Diagrams/sequence_diagram_athlete.jpg> | The link on the left links directly to a .jpg image of my sequence diagram. | My code. The same code used to demonstrate the MVC architectural pattern. |

**Diagram**: Use Case Diagram

**Description of Understanding:** Use case diagrams are used to display the requirements of a system. It is a very high-level representation of the system that doesn’t portray many specific details at all. The components of a use case diagram are actors, associations, system boundaries, and use cases. Actors are individuals who interact with a use case. They are named by nouns, and they trigger the use cases. They often provide input and expect output. Use cases are system functions, either automated or manual, and are named by verbs. Actors are linked to use cases, though not all use cases are linked to actors. Communication / associative links connect actors to use cases, to represent their relationship. System boundaries may outline the entire system, or simple modules. Actors lie outside of the system boundary, and use cases lie within it.

**Teaching Video:** None

|  |  |  |
| --- | --- | --- |
| **File** | **What should I be looking for?** | **Example**  **Or Your code?** |
| <https://github.com/alkire-jeremy/CIT360/blob/master/Familiarity%20Requests/Request%201%20-%20Week%204/Diagrams/Use%20Case%20Diagrams/use_case_diagram_athlete.jpg> | The link on the left links directly to a .jpg image of my use case diagram. | My code. The same code used to demonstrate the MVC architectural pattern. |

**Diagram**: Use Case Document

**Description of Understanding:** A use case document offers a story of how an actor utilizes a system (via use cases) to achieve its goals. It provides a detailed step by step description of how the actor will use the system to achieve its desired outcome. Unlike a use case diagram, it provides a written representation of this information, rather than illustrated. It is typically in table format and is composed of a general description of the use case, its name, detail about the author and date in which this information was documented, a list of actors, preconditions, postconditions, an explanation of flow, alternative flows, exceptions, and requirements.

**Teaching Video:** None

|  |  |  |
| --- | --- | --- |
| **File** | **What should I be looking for?** | **Example**  **Or Your code?** |
| <https://github.com/alkire-jeremy/CIT360/blob/master/Familiarity%20Requests/Request%201%20-%20Week%204/Documentation/Engine%20(Use%20Case%20Document).doc> | The link on the left links directly to a .doc file, which is my use case document for the program I used in my MVC example.  The program is VERY simple, so there wasn’t much to document about this program, either in the use case document or the use case diagram. | My code. The same code used to demonstrate the MVC architectural pattern. |