Name : W.S Mendis Index No : 140392

CS2012 : Principles of Object Oriented Programming – Semester Project – Final Report

Uses of fundamental OOP Principles :

1. Inheritance :

Inheritance allows the subclasses inherit all the attributes and methods of their superclass. As clearly displayed in the UML diagram, inheritance was used in several occasions in this project. All the human related classes such as Swimmer, Judge, Spectator, \Inheritance allows the subclasses inherit all the attributes and methods of their superclass. As clearly displayed in the UML diagram, inheritance was used in several occasions in this project. All the human related classes such as Swimmer, Judge, Spectator, SupportStaff etc. inherits from Person super class. And MaleSwimmer and FemaleSwimmer classes inherits from abstract Swimmer class. So, this concept is useful in code redundancy and extending the reusability.

2. Abstraction :

Abstraction concept is referred to ignoring irrelevant aspects and focusing of reflecting aspects of an object. This concept is useful in handling the complexity of the project. Couple of abstract classes such as Person and Swimmer were used in this project. Even though these classes cannot be instantiated, they minimizes the complexity by holding attributes and methods of their subclasses.

The abstract methods such as doBackStroke(), doBreastStroke() etc in Swimmer class are useful in implementing (overriding) each method slightly differently in MaleSwimmer and FemaleSwimmer classes. **(Line – 115,116,117,118 in Swimmer Class)**

3. Polymorphism :

The ability of an object to take many forms is called as polymorphism. Polymorphism was mainly used Swimmer type ArrayLists (ArrayList<Swimmer>) which can store objects of both MaleSwimmer and FemaleSwimmer types. **(Line – 13,14 in PeopleTrack Class)** This reduces the complexity of the project by giving a more generic type to store different types of objects. The same

concept was used in method load in SaveLoadData class where a ArrayList of type Person (ArrayList<Person>) was declared to store different types of objects (Swimmers, Judges etc.)

(Line – 26 in SaveLoadData Class) In ThreadMovement class the ArrayList results of type Swimmer is used to store both male and female swimmers. (**Line - 28 in ThreadMovement Class**)

4. Encapsulation :

Encapsulation is referred to hiding of implemented attribute in order to maintain the security of the code. This concept was used in this project to hide the most important attributes such as ArrayLists which keeps track on people in PeopleTrack class. **(Line – 13,14,15,16 in PeopleTrack Class)** These ArrayLists were declared private, therefore they cannot be accessed or edited from a outside class. These lists were maintained by using setter and getter methods. Other than this, in almost all the classes, the encapsulation is maintained by declaring attributes private and implementing setters and getters.

Concurrency Control :

The movement of the each swimmer is configured by a thread which is implemented in ThreadMovement Class. This class is a subclass of Thread class in Java, therefore the run() method was overridden to initiate the movement of each swimmer. In the run() method, the movement of the swimmer is configured by generating a random value according to the swimmer's rating (which is given by the user) and summing them inside a loop. **(Line – 74,76,83 in ThreadMovement Class)** This will make sure that the movement of each swimmer differ from one another. These threads are initiated (Thread.start()) in the doBackStroke(), doBreastStroke() etc methods in the MaleSwimmer and FemaleSwimmer classes.

So, whenever a new simulation begins multiple threads will start and creates a multi threaded environment. However all the threads in this projects are mainly used configure the movement of the swimmer, therefore even there are interruptions between the threads, it doesn't really break the progress of the simulation. The run method was declared as a synchronized method to make sure it is only executed by a one of the threads at once. And the sleep time of the thread was minimized to around 40 milliseconds to maintain a smooth flow in GUI movement, otherwise sliders tend to show a lagging movement. **(Line – 79 in ThreadMovement Class)**

Other than setting the movement of swimmers, these threads in ThreadMovement Class updates the status of the Judges and Spectators, displays and records the results, and updates the GUI in different

stages of their running time. **(Line – 68,69,70 in ThreadMovent Class)** Because run method is declared as a synchronized method multiple threads cannot perform above functions simultaneously. This useful in maintaining the flow and continuity of the GUI.

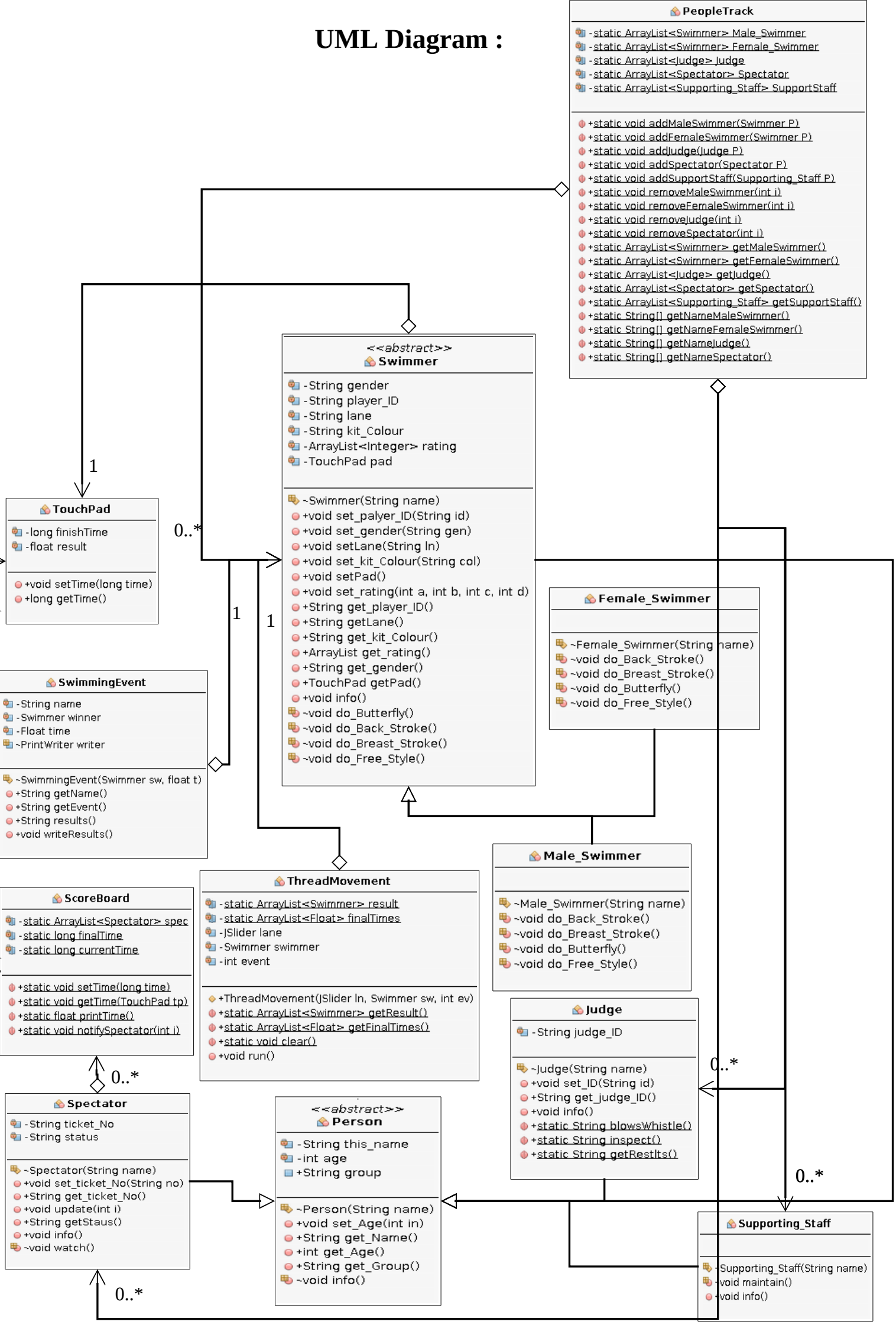
Observer Observable Concept :

Observer Observable concept was used in this project to build the relation between Spectator and the Score Board. In this project this relation was built manually without using in built the Observer and Observable classes in Java.

In this scenario Observer is the Spectator and the Observable is the Score Board which notifies the Observer when states changed. First a ArrayList of type Spectator (ArrayList<Spectator>) was implemented inside the Class ScoreBoard. This is one of the fundamentals of this concept where Observable should consist of a list of Observers. Usually there should be attach() and detach() methods inside the Observable Class in order to maintain list of Observers, but in this case these methods haven't been implemented because PeopleTrack Class already keeps track on every Spectator created in the simulation. **(Line – 16 in PeopleTrack class)** Therefore that list of Spectators is implemented inside the ScoreBoard class.

Then the notifySpectator(int i) method was implemented to update all the Spectators in the list **(Line – 45 in ScoreBoard Class)** This integer parameter (i) is given by the running thread, and inside the method this value is passed to the update(int i) method in Spectator Class. According to this value Spectator changes its status **(Line – 33 in Spectator Class)** In order to Spectator objects to be updated according the ScoreBoard states, notifySpectator(int i) method should constantly feed update(int i) method with values given by the Thread. It is done in the ThreadMovement Class. **(Line – 71,90 in ThreadMovement Class)**

UML Diagram :


PeopleTrack
-static ArrayList<Swimmer> Male_Swimmer
-static ArrayList<Swimmer> Female_Swimmer
-static ArrayList<Judge> Judge
-static ArrayList<Spectator> Spectator
-static ArrayList<Supporting_Staff> SupportStaff
+static void addMaleSwimmer(Swimmer P)
+static void addFemaleSwimmer(Swimmer P)
+static void addJudge(Judge P)
+static void addSpectator(Spectator P)
+static void addSupportStaff(Supporting_Staff P)
+static void removeMaleSwimmer(int i)
+static void removeFemaleSwimmer(int i)
+static void removeJudge(int i)
+static void removeSpectator(int i)
+static ArrayList<Swimmer> getMaleSwimmer()
+static ArrayList<Swimmer> getFemaleSwimmer()
+static ArrayList<Judge> getJudge()
+static ArrayList<Spectator> getSpectator()
+static ArrayList<Supporting_Staff> getSupportStaff()
+static String[] getNameMaleSwimmer()
+static String[] getNameFemaleSwimmer()
+static String[] getNameJudge()
+static String[] getNameSpectator()
<<abstract>>
Swimmer
-String gender
-String player_ID
-String lane
-String kit_Colour
-ArrayList<Integer> rating
-TouchPad pad
~Swimmer(String name)
+void set_palyer_ID(String id)
+void set_gender(String gen)
+void setLane(String ln)
+void set_kit_Colour(String col)
+void setPad()
+void set_rating(int a, int b, int c, int d)
+String get_player_ID()
+String getLane()
+String get_kit_Colour()
+ArrayList get_rating()
+String get_gender()
+TouchPad getPad()
+void info()
~void do_Butterfly()
~void do_Back_Stroke()
~void do_Breast_Stroke()
~void do_Free_Style()
TouchPad
-long finishTime
-float result
+void setTime(long time)
+long getTime()
SwimmingEvent
-String name
-Swimmer winner
-Float time
~PrintWriter writer
~SwimmingEvent(Swimmer sw, float t)
+String getName()
+String getEvent()
+String results()
+void writeResults()
Female_Swimmer
~Female_Swimmer(String name)
~void do_Back_Stroke()
~void do_Breast_Stroke()
~void do_Butterfly()
~void do_Free_Style()
Male_Swimmer
~Male_Swimmer(String name)
~void do_Back_Stroke()
~void do_Breast_Stroke()
~void do_Butterfly()
~void do_Free_Style()
ThreadMovement
-static ArrayList<Swimmer> result
-static ArrayList<Float> finalTimes
-JSlider lane
-Swimmer swimmer
-int event
+ThreadMovement(JSlider ln, Swimmer sw, int ev)
+static ArrayList<Swimmer> getResult()
+static ArrayList<Float> getFinalTimes()
+static void clear()
+void run()
ScoreBoard
-static ArrayList<Spectator> spec
-static long finalTime
-static long currentTime
+static void setTime(long time)
+static void getTime(TouchPad tp)
+static float printTime()
+static void notifySpectator(int i)
Judge
-String judge_ID
~Judge(String name)
+void set_ID(String id)
+String get_judge_ID()
+void info()
+static String blowsWhistle()
+static String inspect()
+static String getRestIts()
Spectator
-String ticket_No
-String status
~Spectator(String name)
+void set_ticket_No(String no)
+String get_ticket_No()
+void update(int i)
+String getStaus()
+void info()
~void watch()
<<abstract>>
Person
-String this_name
-int age
+String group
~Person(String name)
+void set_Age(int in)
+String get_Name()
+int get_Age()
+String get_Group()
~void info()
Supporting_Staff
~Supporting_Staff(String name)
~void maintain()
+void info()
1
0..*
1
1
1
1
0..*
0..*
0..*
0..*
0..*