* Data types and student written class types

We have demonstrated our knowledge on data types and written class types by writing 6 classes which work together to run our game, and 3 of them are written class types.

* ArrayLists

We demonstrate our knowledge on ArrayLists in our main runner class: SpySchool. Within, we have 2 ArrayLists in – one is an ArrayList of the Spy class to hold all the spies who are playing, and the other is an ArrayList of Integers, which hold all the points that the team wins as a whole per game (index 0 holds the amount of points the team has won for the Riddles Game, index 1 holds the amount of points the team has won for the Math Game, so on and so forth). Furthermore, we have used iterators to facilitate the traversal of these ArrayLists.

* Control structures: if/else statements, while loops, and for loops

We have used numerous if/else statements throughout our program. One instance where we use if/else statements is when we ask the user to choose between playing as a water or land spy. If the user chooses to play as a water spy, we create an object of the WaterSpy class and place it in the “spies” ArrayList. Otherwise, if the user chooses to play as a land spy, we create an object of the LandSpy class and place it in the “spies” ArrayList. Another instance where we use if/else statements is when we check how many times the team has beat the computer team. If the team does not beat the computer team at least once, they automatically fail the Spy School and the game ends there. Otherwise, we give the team hints for them to guess their key word. Additionally, we used if/else statements to check for the answers of all of our mini-games. For example, in our runRiddles method in the Mini-Games class, we only give the player 1 point if their response is equal to the answer. Otherwise, we give them 0 points.

While loops were essential in our program for error trapping. For example, in SpySchool at the Anagram game part, while the user enters nothing in, we ask them to enter their answer, and only stop once they have entered in a value. Furthermore, while loops are also used in the logic for our games, particularly in our runRiddles method of the MiniGames class, where we use a while loop to continue asking the user only while they enter an incorrect answer and have not used up all 3 of their chances. Additionally, while loops are essential in the sortWord method in the MiniGames class, since they were used to sort the letters into alphabetical order.

For loops are also very important to our program. Since this program is a multi-player team game, each mini-game must be repeated as many times as there are players to give everyone a fair shot. Furthermore, in games where the questions are asked several times – for example, the math game—for loops were used to fix the amount of times to ask the players.

* Classes & Methods

We wrote 6 different class, of which 2 of them can be instantiated. Furthermore, we have written several methods which interact with the runner class and themselves. An example of methods interacting with each other is when the method ranking is called by the guessPassword method in the LastPart class. This method is then called in the runner (SpySchool) class.

* Polymorphism through inheritance

LandSpy and WaterSpy inherit from the Spy abstract class, and thus LandSpy is a Spy and WaterSpy is a Spy. Within the Spy class there are several completed methods, as well as incomplete methods which are completed in WaterSpy and LandSpy (getHint, getSpyType, addSpecialPoints, getSpecialPoints).

Polymorphism is achieved several times in the runner class (SpySchool) – we call the getName method several times to get the names of both the WaterSpy and LandSpy players. Furthermore, we call the method getHint on the Spy objects within the “spies” ArrayList, which will return different chars depending on whether the Spy object is a WaterSpy or a LandSpy (WaterSpy will return vowels, LandSpy will return consonants). Additionally, we call the addPoints or addSpecialPoints method after each Spy’s turn in the mini-game to add the player’s points. Another instance where we use polymorphism is

* File input/output

We use file input for the Riddles and Math games. We read from files that contain the riddle and the answers. We output only the riddle to the user, and use the answer to check the user’s response. If their response matches the answer, the user gets a point.

We use file output for teams that have successfully won the game. As a winning team, they are able to sign a “yearbook” which contains their team name and team “score”. Since this is not a fixed sized multiplayer game, the team’s score is the total number of points earned divided by the number of players (much like a “unit score”) so that can easily and fairly be compared with other teams. Signing the yearbook can be a source of motivation for the players – whoever signs it will stay on it forever and will leave a legacy for others to try and beat.

* Sort/search

We use sorting for the Anagrams game – in the sortWord method in the MiniGame class, both the original word and the user’s input are sorted in alphabetical order. If they are the same, we give the user a point, since they have successfully created an anagram with the word they were given.

We use a combination of sort and search to show the winners their ranking on the “yearbook”. The team “scores” (as mentioned in the file input/output section above) of all the players who have played the game are sorted in descending order. Then we search for the current team’s score and find out their position on the leaderboard. For example, if they have the most amount of points of everyone who has ever played our game, they rank 1st, and so on.

* String Tokenizers

We use String Tokenizers frequently with file input/output. For the Riddles and Math games, the questions and answers are separated by a “=” or a “:”. For the Riddles game, we use file input to get a line of String. Then we use String tokenizers to “chop” the line whenever there is a “:”. This way, it separates the question from the answer, which then can be placed into separate index positions (question is placed in index 0, answer is placed in index 1). We use a similar method with the Math game file, except the delimiter is “=” instead of “:”.