**SCC.110: Term 2 Java Programming Coursework Specification**

**Assessed Exercise: Java Program**

**Moodle Submission Deadline:**  **Your timetabled lab session in Week 20**

**Assessment Mode: Moodle CodeRunner quiz and code submission**

**Aims**

This assessed exercise is designed to test your understanding of the software development concepts we’ve seen in the lectures and labs, specifically for creating object-oriented code. The assignment is split into *five tasks*: four main tasks and one ‘extension’ task to be carried out in your timetabled lab session in week 20. The tasks ask you to develop classes, instance variables and methods. These classes build on each other and together form a complete Java program.

Your solution will be tested to make sure its functionality meets the specification and object-oriented principles. The number of marks is indicated in this specification against each task, with a total of *54 marks available* (50 marks for tasks 1-4 and an additional 4 marks for the extension task to be conducted in your timetabled lab session in week 20).

**Tasks**

You’ve been asked to create a Java program to represent gaming parties, where players get together, bring snacks, and play a board game. This will require the classes specified in each of the tasks below.

You must follow **Java naming conventions** as well as **name your classes, instance variables and methods exactly the same as stated** in the tasks below. **This includes use of capital or lower-case letters!**

You must ensure all printed error messages / information output requested in the tasks below look exactly the same as that in the specification, so check for spelling and punctuation! **Do not print out any information or debug statements in your code, except for the output requested in this specification.**

**Task 1: Player class**

1. Write a Java class to represent a **Player**. This class should contain the instance variables stated below in the order they appear. Be careful to choose the best types to hold this information, and ensure your class is properly encapsulated.
   * **name**: the name of the player.
   * **age**: the age of the player.

[4 marks]

1. Write a **constructor** for your Player class. This must take two parameters for **name** and **age** in that order.

[2 marks]

1. Write an **accessor (getter) method** for the Player’s **name** called **getName**.

[2 marks]

1. Write an **accessor method** for the Player’s **age** called **getAge**.

[2 marks]

[Total 10 marks for task 1]

**Task 2: Snack class**

1. Write a Java class to represent a **Snack**. This class should contain the instance variables stated below in the order they appear. Be careful to choose the best types to hold this information, and ensure your class is properly encapsulated.
   * **description**: a written description of the snack (e.g.,"Fruit salad").
   * **cost**: the cost of the snack as a decimal value (e.g., 4.5 to represent £4.50).
   * **provider:** the Playerwho is bringing the snack (i.e., using your Player class from task 1).

Your Snack class must also include a **constructor**. This must take three parameters for **description**, **cost**, and **provider** in that order.

[5 marks]

1. Write **accessor methods** for the Snack **description** called **getDescription**, **cost** called **getCost**, and **provider** called **getProvider.**

[3 marks]

[Total 8 marks for task 2]

**Task 3: BoardGame class**

1. Write a Java class to represent a **BoardGame**. This class should contain the instance variables stated below in the order they appear. Be careful to choose the best types to hold this information, and ensure your class is properly encapsulated.
   * **title**: the title of the board game (e.g.,"Pictionary").
   * **owner**:the Playerwho owns the board game (i.e., using your Player class from task 1).
   * **minimumAge**: the minimum age that players must be in order to play the board game (e.g., 18).
   * **duration**: the estimated time to play in whole minutes (e.g., 60).
   * **minimumPlayers**: the minimum number of players required to play the board game (e.g., 1).
   * **maximumPlayers**: the maximum number of players allowed to play the board game (e.g., 10).
   * **rating**: the percentage rating of the board game as a decimal value (e.g., 75.5 to represent 75.5%).

Your **BoardGame** class must also include a **constructor**. This must take seven parameters for **title**, **owner**, **minimumAge**, **duration**, **minimumPlayers**, **maximumPlayers**, and **rating** in that order. If the rating value given to the constructor is not a percentage (i.e., is below 0 or above 100), then the constructor should output an error message "**Rating percentage not valid. Setting to 0.**" and set the rating to 0.

You must also create **accessor methods** for each of the instance variables in the class following the accessor naming convention get[VariableName] (e.g. getTitle, or getMinimumAge).

[4 marks]

1. Write a **mutator (setter) method** for the **rating** of the board game called **setRating**. If the rating value given to the mutator is not a percentage (i.e., is below 0 or above 100), then the method should output an error message "**Rating percentage not valid.**" and not update the rating.

[2 marks]

[Total 6 marks for task 3]

**Task 4: GamingParty class**

1. Write a Java class to represent a **GamingParty**. This class should contain the instance variables stated below in the order they appear. Be careful to choose the best types to hold this information, and ensure your class is properly encapsulated.
   * **theme**: the theme of the gaming party (e.g.,"Friends Gathering").
   * **boardGame**: the board game that players at the party will play (i.e., using your BoardGame class from task 3).
   * **players**: a Player array of players who will attend the gaming party. This must use a standard Java array (i.e., Player[]), not another class (e.g. ArrayList). The array must start as empty.
   * **snacks**: a Snack array of snacks for the gaming party. This must use a standard Java array (i.e. Snack[]), not another class (e.g. ArrayList). The array must start as empty.
   * **winner**: the Player who won the board game at the party.

Your GamingParty class must also include a **constructor**. This must take **two** parameters for **theme** and **boardGame** in that order. The length of the array of **players** should be defined by the maximumPlayers value from the BoardGame instance boardGame. The length of the array of **snacks** should be twice the maximumPlayers value from the BoardGame instance boardGame. The value of **winner** should be set to null.

[5 marks]

1. Write a method called **addPlayer** that adds players (i.e., instances of the Player class) to the **players** array, with the following requirements:
   * If the player does not meet the minimum age of the board game at the gaming party, the player should not be added, and an error message should be printed stating: "**The player does not meet the age requirements for the board game at this party.**"
   * If the player is already at the gaming party (i.e., the player is already in the players array), then the player should not be added again, and an error message should be printed stating: "**The player is already playing the game at this party.**"
   * If the maximum number of players have already been reached for the board game at the gaming party, then the player should not be added, and an error message should be printed stating: "**The maximum number of players has been reached for the game at this party.**"
   * If the player meets the age requirements for the board game, they are not already playing the game, and the maximum number of players has not been reached, then the player should be added, and no output should be printed.

[4 marks]

1. Write a method called **play** that outputs whether the gaming party can begin, with the following requirements:
   * If the minimum number of players for the board game at the gaming party has been met, then a message should be printed stating: "**Play!**"
   * If the minimum number of players for the board game at the gaming party has not been met, then an error message should be printed stating: "**You need** [X] **more player(s)!**" where [X] is replaced by the number of players still required (e.g., "You need 2 more player(s)!")

[2 marks]

1. Write a method called **addSnack** that adds snacks (i.e. instances of the Snack class) to the **snacks** array, with the following requirements:
   * If the maximum number of snacks have already been reached for the gaming party, then the snack should not be added, and an error message should be printed stating: "**There are enough snacks!**"
   * If the maximum number of snacks has not been reached for the gaming party, the snack should be added, and no output should be printed.

[2 marks]

1. Write a method called **getPartyCost** that returns the total cost of the party based on the sum of costs of the party **snacks** as a decimal value (e.g., 20.9). You do not need to, nor should you not, do any additional formatting for this decimal value (e.g., output a pound sign, £, or round values to two decimal places).

[3 marks]

1. Write two methods, an **accessor method** and a **mutator method**, for the **winner** of the board game played at the gaming party called **getWinner** and **setWinner**.

The mutator method must have the following requirements:

* + If the winner has already been set, then the winner should not be updated and an error message should be printed stating: "**The winner has already been decided! It was** [X]**.**" where [X] is replaced by the name of the original winner (e.g., "The winner has already been decided! It was Ioannis.")
  + If the winner has not already been set but the winning player given as a parameter was not at the gaming party, then the winner should not be updated, and an error message should be printed stating: "**The player didn't even play the game so cannot win!**"
  + If the winner has not been set and the winning player given as a parameter was at the gaming party, the winner should be updated and no output should be printed.

[4 marks]

1. Write a method called **outputPartyDetails** which outputs details of the gaming party. For a gaming party with the theme "Friends Gathering" to play "Pictionary", with two players (named Ioannis and Adrian), one snack of "Fruit salad" provided by Adrian, and the winning player being Ioannis, the output should look like the following (including the new lines and punctuation):

**Theme: Friends Gathering**

**Board game: Pictionary**

**Players:**

**Ioannis**

**Adrian**

**Snacks:**

**Fruit salad provided by Adrian**

**The winner is Ioannis!**

Without a winner, snacks or players, the output should look like the following (including the new lines and punctuation):

**Theme: Friends Gathering**

**Board game: Pictionary**

**Players:**

**Snacks:**

**No winner yet!**

[2 marks]

1. Write a method called **calculateRecommendedSnacks** which calculates the recommended number of snacks for the gaming party based on the duration of the board game and the number of players added to the gaming party, with the following requirements:
   * For **each hour** of the board game duration, **one snack** should be recommended **per player** added to the gaming party (*note*: convert the game duration from minutes to hours and round the result to the *nearest whole value*, e.g., 2.5 hours should become 3.0 hours, whereas 5.2 hours should become 5.0 hours). In the case where this is more than the maximum number of snacks allowed at the gaming party (i.e., double the board game’s maximumPlayers value), then the **maximum number of snacks** allowed at the gaming party should be recommended instead.
   * The **minimum number of snacks** recommended should be one for each player added to the gaming party. If no players have been added to the gaming party, no snacks should be recommended.
   * Your method should output a message stating: "**A total of** [X] **snack(s) are recommended for the game.**" where [X] is replaced by the (integer) number of snacks recommended (e.g., "A total of 4 snacks are recommended for the game.")

[4 marks]

[Total 26 marks for task 4]

**[Total 50 marks for tasks 1-4]**

**Submission**

Submission of your work will take place **during your timetabled practical session in Week 20**. You will be asked to submit the four classes (tasks 1-4) specified above as well as a fifth class, which will be an extension task for you to conduct during your timetabled practical session in Week 20, which you **must** attend. There will be **two submission points** for this coursework on Moodle: one submission point will be for your **.java files** (compressed into a single **.zip** file) in your solution. This includes tasks 1-4 as above, as well as the lab session extension task 5. The other submission point will be a **CodeRunner quiz** composed of five questions – each question will be asking for one of your classes (the four classes for tasks 1-4 and the class for the lab extension task 5). **You must submit code – which compiles – to both submission points. Do not submit incomplete code that does not compile, as this will give you a mark of 0**.

More details will be given to you on the day of your timetabled practical session in Week 20.