### **Game Management System Assignment**

#### **Requirements**:

### **1. Game Class:**

* Attributes:
  + name: Name of the game.
  + category: Category of the game (Action, Strategy, Adventure, Role-Playing, Simulation, Sports, Racing, Horror, Fighting, Music, Puzzle, Educational etc.).
  + rating: Rating of the game. (0 to 10)
* Constructors, Getters, and Setters:
  + Create a parameterized constructor to initialize the attributes.
    - In the constructor public Game(String name, String category, double rating), you provide three parameters: name, category, and rating. Inside the constructor, you use the this keyword to refer to the instance variables of the class and assign the parameter values to them.
  + Implement appropriate getters and setters for each attribute.
* toString Method:
  + Override the toString method to provide a string representation of the Game object.

### **2. Categories Interface:**

* Interface:
  + Interface Name:
    - Categorizable
  + Abstract Method:
    - String getCategory(): Abstract method to be implemented by classes that can be categorized.
  + Optional Features:
    - Default methods or constants for future expansion (if needed).

### **3. Game Categories:**

* ActionGame Class:
  + Extend the Game class.
  + Implement the Categorizable interface.
  + Provide a specific category for action games.
* StrategyGame Class:
  + Extend the Game class.
  + Implement the Categorizable interface.
  + Provide a specific category for strategy games.
* *Note: Create additional classes for other game categories like* ***Adventure, Role-Playing, Simulation, Sports, Racing, Horror, Fighting, Music, Puzzle, Educational.***

***4. GameManager Class:***

#### *Attributes:*

* + *List<Game>: A list to manage games.*

#### *Methods:*

* + *addGame(Game game):*
    - *Description:*
      * *Adds a new game to the list of managed games.*
    - *Parameters:*
      * *Game game: The game object to be added.*
    - *Implementation:*
      * *Appends the provided game to the list of games.*
  + *categorizeGames():*
    - *Description:*
      * *Categorizes the games using Java 8 streams.*
    - *Implementation:*
      * *Uses Java 8 streams to group the games by their category.*
      * *Displays the categorized games.*
  + *displayGamesByCategory():*
    - *Description:*
      * *Displays the categorized games.*
    - *Implementation:*
      * *Calls the categorizeGames method to categorize the games.*
      * *Prints the categorized games to the console.*
  + *getGameByName(String name):*
    - *Description:*
      * *Searches for a game by its name.*
    - *Parameters:*
      * *String name: The name of the game to search for.*
    - *Returns:*
      * *Returns the Game object if found; otherwise, returns null.*
    - *Implementation:*
      * *Uses Java 8 streams to filter the list of games based on the provided name.*
      * *Returns the first game that matches the name, or null if no match is found.*

#### *Note:*

* *Java 8 Features:*
  + *Utilizes Java 8 features such as streams and lambda expressions.*
  + *The categorizeGames method leverages streams to group games by category.*
  + *The getGameByName method uses streams for filtering based on the game name.*

*This GameManager class efficiently manages a list of games, categorizes them using Java 8 streams, and provides methods for adding games, displaying categorized games, and searching for a game by name. The use of Java 8 features enhances the readability and conciseness of the code.*

### **5. User Class:**

#### Attributes:

### String username**: The username of the user.**

### List<Game>**: A list to manage favorite games.**

#### **Methods:**

### **addFavorite(Game game):**

### **Description:**

### **Adds a game to the list of user's favorite games.**

### **Parameters:**

### Game game**: The game object to be added to the favorites.**

### **Implementation:**

### **Appends the provided game to the list of favorite games.**

#### **Note:**

### **Usage:**

### **Users can create an instance of the** User **class and use the** addFavorite **method to add games to their list of favorite games.**

### **This** User **class provides a simple way to manage a user's information, including their username and a list of favorite games. The** addFavorite **method allows users to add games to their list of favorites.**

### **6. SettingsPage Class:**

#### Attributes:

### User user**: Reference to the user for whom settings are being configured.**

#### **Methods:**

### **changeUsername(String newUsername):**

### **Description:**

### **Changes the username of the user.**

### **Parameters:**

### String newUsername**: The new username to be set.**

### **Implementation:**

### **Sets the username of the associated user to the provided new username.**

### **Additional Methods (Optional):**

### ***Include additional methods for changing preferences as needed.***

### ***For example:***

### changePassword(String newPassword)**: Changes the password of the user.**

### updatePreferences(Map<String, Object> preferences)**: Updates various user preferences based on the provided map.**

#### **Note:**

### **Usage:**

### **Users can create an instance of the** SettingsPage **class with a reference to their user.**

### **They can then use the provided methods to change their username and adjust other settings as needed.**

### **This** SettingsPage **class serves as a mechanism for users to configure their settings. The** changeUsername **method allows users to update their username, and additional methods can be added to support various preferences adjustments.**

### **7. Main Application:**

#### Tasks:

### Create an Instance of GameManager:

### Method Name:

### createGameManager

### Description:

### Instantiates a GameManager **to manage the list of games.**

### **Implementation:**

### **Returns a new instance of the** GameManager **class.**

### **Allow Users to Interact with the System:**

### **a. View Games by Category:**

### **Method Name:**

### viewGamesByCategory

### Description:

### Displays games categorized by their respective categories.

### Implementation:

### Calls the displayGamesByCategory **method of the** GameManager **to show games by category.**

### **b. Search for Games:**

### **Method Name:**

### searchForGame

### Description:

### Allows users to search for games by name.

### Implementation:

### Takes user input for the game name and calls the getGameByName **method of the** GameManager **to search for the game.**

### **c. Add Games to Favorites:**

### **Method Name:**

### addGameToFavorites

### Description:

### Enables users to add games to their list of favorite games.

### Implementation:

### Takes user input for the game name and calls the addFavorite **method of the** User **class associated with the current user.**

### **d. Access the Settings Page:**

### **Method Name:**

### accessSettingsPage

### Description:

### Allows users to access the settings page to modify their preferences.

### Implementation:

### Instantiates a SettingsPage **with a reference to the current user and provides options for changing the username and other preferences.**

#### **Note:**

### **Java 8 Features:**

### **Utilizes Java 8 features such as streams and lambda expressions where applicable.**

### **For example, when displaying games by category, the** viewGamesByCategory **method may use Java 8 streams for efficient processing of the list of games.**

### **Suggestions:**

### **Implement user-friendly console prompts for input.**

### **Provide clear instructions and options for each interaction.**

### **Consider handling edge cases, such as no matching games during search or empty lists.**

### **8. Java 8 Features:**

* Utilize Java 8 Features:
  + Use lambda expressions for concise and expressive code.
  + Apply streams for efficient data processing (e.g., filtering, mapping, grouping).
  + Explore the use of functional interfaces like Predicate for filtering games based on certain criteria.
  + Consider using Optional for methods that may return null.

### **9. Considerations:**

* Focus on Core Functionality:
  + Prioritize implementing core features within the given time frame.
* Organized Class Structure:
  + Create a well-organized class structure with meaningful relationships between classes.
* Error Handling:
  + Implement basic error handling where necessary (e.g., handling null values, avoiding duplicate entries).
* Use Collections:
  + Utilize collections (e.g., ArrayList) to manage lists of games and favorites efficiently.