**Exercise 1 - Paintings and Painters**

**Overview**

The provided Java code implements a system to manage painters and their paintings. It encapsulates information about individual paintings and a painter’s collection of paintings. The functionality includes:

1. Adding, removing, and selling paintings.
2. Calculating metrics such as total income, expected income, and the most expensive painting sold.
3. Finding the largest painting in the collection.

**Class Design**

The system is structured into three classes:

1. **Tester Class:** Contains the main method for testing the functionality of the Painting and Painter classes.
2. **Painting Class:** Represents individual paintings, encapsulating attributes such as title, dimensions, price, and sold status.
3. **Painter Class:** Manages a collection of paintings and provides methods to interact with and query the collection.

**Explanation of Classes and Methods**

**Painting Class**

This class models a painting with the following attributes and methods:

**Attributes:**

* title (String): The name of the painting.
* length (int): The length of the painting in arbitrary units.
* width (int): The width of the painting in arbitrary units.
* price (double): The price of the painting.
* sold (boolean): Indicates whether the painting has been sold.

**Constructor:**

* Initializes the painting with the provided values.

**Methods:**

* Getter methods for all attributes.
* Setter methods for price and sold status.
* toString(): Returns a string representation of the painting's details for display purposes.

**Painter Class**

This class models a painter and their collection of paintings.

**Attributes:**

* name (String): Painter's first name.
* surname (String): Painter's last name.
* birthYear (int): Painter's birth year.
* paintingsList (ArrayList): A collection of the painter’s paintings.

**Constructor:**

* Initializes the painter with the provided details and creates an empty list for paintings.

**Methods:**

* **Add and Remove Paintings:**
  + addPainting(Painting p): Adds a painting to the collection.
  + removePainting(String title): Removes a painting by its title.
* **Sell Painting:**
  + sellPainting(String title): Marks a painting as sold if it exists and isn’t already sold.
* **Metrics:**
  + numberOfPaintings(): Returns the total number of paintings.
  + numberOfSoldPaintings(): Returns the count of sold paintings.
  + totalIncome(): Calculates the total income from all paintings.
  + expectedIncome(): Calculates the potential income from unsold paintings.
  + mostExpensiveSoldPainting(): Finds the title of the most expensive painting that has been sold.
  + largestPainting(): Finds the title of the painting with the largest area (length \* width).

**Thought Process Behind the Code**

1. **Encapsulation:**
   * The use of private attributes and public getter/setter methods in Painting and Painter ensures controlled access to class data.
   * Encapsulation improves code maintainability and security.
2. **Separation of Concerns:**
   * The Painting class handles the properties of individual paintings.
   * The Painter class manages a collection of paintings and provides high-level operations on the collection.
3. **Dynamic Collection Management:**
   * The use of an ArrayList in the Painter class allows dynamic resizing and easy management of paintings.
4. **Flexibility:**
   * The sellPainting and removePainting methods use loops and conditional checks to handle various scenarios, such as selling an already sold painting or trying to remove a non-existent painting.
5. **User Feedback:**
   * The methods provide feedback to the user, such as printing messages when a painting is sold or when an operation fails (e.g., removing a non-existent painting).
6. **Iterative Calculations:**
   * Methods like totalIncome, expectedIncome, and largestPainting iterate over the collection to compute results, ensuring the metrics are always up-to-date.