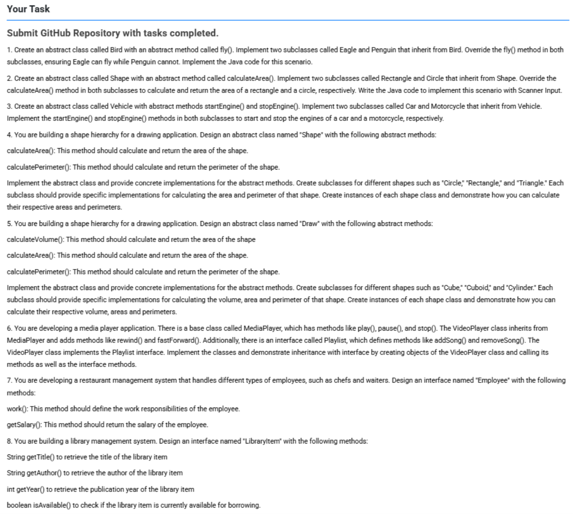
Java OOP Task Breakdown



Your Task

Submit GitHub Repository with tasks completed.

1. Create an abstract class called Bird with an abstract method called fly(). Implement two subclasses called Eagle and Penguin that inherit from Bird. Override the fly() method in both subclasses, ensuring Eagle can fly while Penguin cannot. Implement the Java code for this scenario.

2. Create an abstract class called Shape with an abstract method called calculateArea(). Implement two subclasses called Rectangle and Circle that inherit from Shape. Override the calculateArea() method in both subclasses to calculate and return the area of a rectangle and a circle, respectively. Write the Java code to implement this scenario with Scanner Input.

3. Create an abstract class called Vehicle with abstract methods startEngine() and stopEngine(). Implement two subclasses called Car and Motorcycle that inherit from Vehicle. Implement the startEngine() and stopEngine() methods in both subclasses to start and stop the engines of a car and a motorcycle, respectively.

4. You are building a shape hierarchy for a drawing application. Design an abstract class named "Shape" with the following abstract methods:

calculateArea(): This method should calculate and return the area of the shape.

calculatePerimeter(): This method should calculate and return the perimeter of the shape.

Implement the abstract class and provide concrete implementations for the abstract methods. Create subclasses for different shapes such as "Circle," "Rectangle," and "Triangle." Each subclass should provide specific implementations for calculating the area and perimeter of that shape. Create instances of each shape class and demonstrate how you can calculate their respective areas and perimeters.

5. You are building a shape hierarchy for a drawing application. Design an abstract class named "Draw" with the following abstract methods:

calculateVolume(): This method should calculate and return the area of the shape

calculateArea(): This method should calculate and return the area of the shape.

calculatePerimeter(): This method should calculate and return the perimeter of the shape.

Implement the abstract class and provide concrete implementations for the abstract methods. Create subclasses for different shapes such as "Cube," "Cuboid," and "Cylinder." Each subclass should provide specific implementations for calculating the volume, area and perimeter of that shape. Create instances of each shape class and demonstrate how you can calculate their respective volume, areas and perimeters.

6. You are developing a media player application. There is a base class called MediaPlayer, which has methods like play(), pause(), and stop(). The VideoPlayer class inherits from MediaPlayer and adds methods like rewind() and fastForward(). Additionally, there is an interface called Playlist, which defines methods like addSong() and removeSong(). The VideoPlayer class implements the Playlist interface. Implement the classes and demonstrate inheritance with interface by creating objects of the VideoPlayer class and calling its methods as well as the interface methods.

7. You are developing a restaurant management system that handles different types of employees, such as chefs and waiters. Design an interface named "Employee" with the following methods:

work(): This method should define the work responsibilities of the employee.

getSalary(): This method should return the salary of the employee.

8. You are building a library management system. Design an interface named "LibraryItem" with the following methods:

String getTitle() to retrieve the title of the library item

String getAuthor() to retrieve the author of the library item

int getYear() to retrieve the publication year of the library item

boolean isAvailable() to check if the library item is currently available for borrowing.

Thanks for uploading the task list image. Here's a breakdown of the 8 tasks you're supposed to implement in Java, along with what each one involves:

✅ Task Breakdown and Requirements

---

1. Bird Class with Eagle and Penguin Subclasses

- **Abstract Class:** `Bird`
- **Method to Implement:** `fly()` (abstract in Bird, overridden in subclasses)
- **Subclasses:**
  - `Eagle`: should override `fly()` to show flying behavior.
  - `Penguin`: override `fly()` to indicate penguins don’t fly.

---

2. Shape Class with Rectangle and Circle

- **Abstract Class:** `Shape`
- **Abstract Method:** `calculateArea()`

- **Subclasses**:
  - `Rectangle` (takes length and breadth)
  - `Circle` (takes radius)
- **Scanner Input**: Use to read dimensions and print area.

---

3. Vehicle Class with Car and Motorcycle

- **Abstract Class**: `Vehicle`
- **Abstract Methods**: `startEngine()`, `stopEngine()`
- **Subclasses**:
  - `Car`
  - `Motorcycle`
- Implement and print messages for engine start/stop.

---

4. Shape Hierarchy for Drawing (2D Shapes)

- **Abstract Class**: `Shape`
- **Abstract Methods**:
  - `calculateArea()`
  - `calculatePerimeter()`
- **Subclasses**:
  - `Circle`, `Rectangle`, `Triangle`
- Create objects of each and display area and perimeter.

---

5. Draw Hierarchy for 3D Shapes

- **Abstract Class**: `Draw`
- **Abstract Methods**:

    - `calculateVolume()`
    - `calculateArea()`
    - `calculatePerimeter()`
- **Subclasses**:
    - `Cube`, `Cuboid`, `Cylinder`
- Create and demonstrate objects showing all three calculations.

---

6. Media Player with Inheritance and Interface

- **Base Class**: `MediaPlayer`
    - Methods: `play()`, `pause()`, `stop()`, `rewind()`, `fastForward()`
- **Class**: `VideoPlayer` extends `MediaPlayer`, implements `Playlist`
    - `Playlist` Interface: `addSong()`, `removeSong()`
- Create objects and demonstrate calling class + interface methods.

---

7. Restaurant Management with Interface

- **Interface**: `Employee`
    - `work()`: responsibilities
    - `getSalary()`: return salary
- **Implementing Classes**: `Chef`, `Waiter`
- Demonstrate with basic outputs.

---

8. Library Management System

- **Interface**: `LibraryItem`
    - `getTitle()`
    - `getAuthor()`

  - `getYear()`
  - `isAvailable()`
- Implement class(es) and show functionality with object calls.

Would you like:

- Java code for each task?
- A GitHub repo structure idea?
- Help with implementing or organizing everything for submission?

Let me know how you'd like to proceed!

give me all code

Abstract Interface Tasks

All the code for your 8 tasks has been compiled in the document **“Abstract Interface Tasks”**. It includes:

- Abstract classes and method implementations.
- Subclasses demonstrating polymorphism and inheritance.
- Interface-based implementations for tasks like the media player, employee system, and library system.
- Sample method overrides and functionality demos.

Let me know if you'd like:

- A `main()` method to demonstrate/test each task.
- Separate files for GitHub submission.
- Help creating a `README.md` for your repository.