**[Cơ bản 1] Giải thích tính toán đóng gói (Encapsulation) trong Java**

**1. Mục tiêu:**

* Hiểu và nắm được cách sử dụng tính chất của OOP là tính đóng gói (Encapsulation) trong Java.
* Tạo ra một đoạn mã đơn giản để thể hiện tính đóng gói

**2. Mô tả:**

Giải thích khái niệm **Tính đóng gói (Encapsulation)** trong Java. Hãy trả lời các câu hỏi sau:

1. Tính đóng gói là gì? Tại sao tính đóng gói lại quan trọng trong lập trình đối tượng?

2. Làm thế nào để áp dụng tính đóng gói trong Java? Giải thích cách sử dụng các phương thức getter và setter.

3. Cho ví dụ về cách sử dụng tính đóng gói trong một lớp có thuộc tính name, age (thuộc tính private), và phương thức getter và setter để truy xuất và thay đổi giá trị của chúng.

4. Lợi ích của việc sử dụng tính đóng gói trong việc bảo vệ dữ liệu và kiểm soát quyền truy cập là gì?

***Yêu cầu:***

* Trả lời các câu hỏi lý thuyết.
* Tạo một đoạn mã đơn giản mô tả tính đóng gói trong Java.

**[Cơ bản 2] Giải thích kế thừa (Inheritance) trong Java**

**1. Mục tiêu:**

* Hiểu và nắm được cách sử dụng tính chất của OOP là tính kế thừa (Inheritance) trong Java.
* Hiểu rõ lý thuyết nền tảng để ứng dụng vào thực tiễn. Đồng thời phân biệt được tính kế thừa và tính đóng gói

**2. Mô tả:**

Giải thích khái niệm **Kế thừa (Inheritance)** trong Java. Hãy trả lời các câu hỏi sau:

1. Kế thừa là gì? Nó giúp ích gì trong lập trình hướng đối tượng?

2. Sự khác biệt giữa kế thừa và bao đóng (encapsulation) trong Java là gì?

3. Viết một ví dụ ngắn giải thích cách sử dụng từ khóa extends để tạo lớp con kế thừa các thuộc tính và phương thức từ lớp cha.

4. Hãy chỉ ra các lợi ích và hạn chế khi sử dụng kế thừa trong Java?

***Yêu cầu:***

* Trả lời các câu hỏi lý thuyết.
* Viết một ví dụ minh họa sử dụng từ khóa extends và giải thích rõ ràng cách các lớp con kế thừa các thuộc tính và phương thức từ lớp cha.

**[Cơ bản 3] Xây dựng lớp Car và tính bao đóng**

**1. Mục tiêu:**

* Học cách xây dựng một lớp Java với các thuộc tính có tính bao đóng (encapsulation)
* Hiểu và áp dụng nguyên lý **bao đóng** trong lập trình hướng đối tượng bằng cách sử dụng các phương thức **getter** và **setter**
* Cải thiện khả năng kiểm soát truy cập các thuộc tính của lớp thông qua việc ẩn các thuộc tính này và chỉ cung cấp các phương thức truy cập thích hợp
* Nâng cao kỹ năng kiểm tra tính hợp lệ của dữ liệu thông qua các phương thức setter

**2. Mô tả:**

Trong bài tập này, sinh viên sẽ xây dựng một lớp Car đơn giản đại diện cho một chiếc ô tô với các thuộc tính cơ bản sau:

* **make**: Thương hiệu của xe (Ví dụ: "Toyota", "Honda").
* **model**: Mẫu xe (Ví dụ: "Corolla", "Civic").
* **year**: Năm sản xuất của xe.

Các thuộc tính này cần phải được khai báo là **private** để đảm bảo tính bao đóng. Sinh viên sẽ cần tạo ra các phương thức **getter** và **setter** để truy cập và thay đổi giá trị của các thuộc tính này từ bên ngoài lớp.

***Yêu cầu:***

* Tạo lớp Car với các thuộc tính make, model, và year.
* Đảm bảo các thuộc tính này là **private** để không thể truy cập trực tiếp từ bên ngoài lớp.
* Cung cấp các phương thức **getter** và **setter** cho từng thuộc tính.
  + Phương thức setter của thuộc tính year phải kiểm tra xem năm sản xuất có hợp lệ hay không (năm sản xuất không được lớn hơn năm hiện tại).
* Tạo một lớp **TestCar** để thực hiện kiểm thử:
  + Khởi tạo đối tượng Car và truy cập các thuộc tính qua các phương thức **getter** và **setter**.
  + Thử thay đổi giá trị của thuộc tính year thông qua phương thức setter và kiểm tra xem giá trị có hợp lệ không.

* Dán link github lên phần nộp bài tập trên hệ thống.

**[Cơ bản 4] Xây dựng lớp Book và tính bao đóng**

**1. Mục tiêu:**

* Học cách xây dựng một lớp Java với các thuộc tính có tính bao đóng (encapsulation)
* Áp dụng nguyên lý **bao đóng** trong lập trình hướng đối tượng bằng cách sử dụng các phương thức **getter** và **setter**
* Cải thiện khả năng kiểm soát truy cập các thuộc tính của lớp thông qua việc ẩn các thuộc tính này và chỉ cung cấp các phương thức truy cập thích hợp
* Nâng cao kỹ năng kiểm tra tính hợp lệ của dữ liệu đầu vào thông qua các phương thức setter

**2. Mô tả:**

Trong bài tập này, sinh viên sẽ xây dựng một lớp Book đại diện cho một quyển sách với các thuộc tính cơ bản sau:

* **title**: Tên sách.
* **author**: Tác giả của sách.
* **price**: Giá của sách.

Các thuộc tính này cần phải được khai báo là **private** để đảm bảo tính bao đóng. Sinh viên sẽ cần tạo ra các phương thức **getter** và **setter** để truy cập và thay đổi giá trị của các thuộc tính này từ bên ngoài lớp.

***Yêu cầu:***

* Tạo lớp Book với các thuộc tính title, author, và price là **private**.
* Cung cấp các phương thức **getter** và **setter** cho từng thuộc tính.
  + Phương thức setter của thuộc tính price phải kiểm tra xem giá trị có hợp lệ không (giá không được âm).
* Tạo một lớp TestBook để thực hiện kiểm thử:
  + Khởi tạo đối tượng Book và truy cập các thuộc tính qua các phương thức **getter** và **setter**.
  + Thử thay đổi giá trị của thuộc tính price thông qua phương thức setter và kiểm tra tính chính xác.

**[Khá 5] Xây dựng lớp Circle và lớp Cylinder**

**1. Mục tiêu:**

* Học cách xây dựng lớp trong Java và áp dụng các khái niệm về **kế thừa** và **bao đóng**
* Làm quen với việc sử dụng **constructor** và **method overriding** trong Java
* Áp dụng **tính kế thừa** để tạo ra lớp Cylinder từ lớp Circle và hiểu rõ mối quan hệ giữa các lớp
* Cải thiện khả năng xây dựng các đối tượng, tính toán các đặc tính hình học, và kiểm tra tính hợp lệ của các dữ liệu đầu vào

**2. Mô tả:**

Trong bài tập này, sinh viên sẽ xây dựng hai lớp:

* **Lớp Circle**: Đại diện cho một hình tròn, có thuộc tính bán kính và phương thức tính diện tích của hình tròn.
* **Lớp Cylinder**: Kế thừa từ lớp Circle, đại diện cho một hình trụ, có thêm thuộc tính chiều cao và phương thức tính thể tích của hình trụ.

Lớp **Cylinder** sẽ sử dụng kế thừa từ lớp **Circle**, và thêm các tính toán riêng cho thể tích của hình trụ. Cả hai lớp sẽ áp dụng tính bao đóng để đảm bảo các thuộc tính như bán kính và chiều cao được bảo vệ khỏi sự thay đổi trực tiếp từ bên ngoài lớp.

***Yêu cầu:***

**Lớp Circle**:

* Thuộc tính **radius**: Bán kính của hình tròn (dạng double).
* Phương thức **getArea()**: Tính diện tích của hình tròn (PI \* radius^2).
* Cung cấp các phương thức **getter** và **setter** cho thuộc tính radius.

**Lớp Cylinder** kế thừa từ lớp Circle:

* Thuộc tính **height**: Chiều cao của hình trụ (dạng double).
* Phương thức **getVolume()**: Tính thể tích của hình trụ (PI \* radius^2 \* height).
* Cung cấp các phương thức **getter** và **setter** cho thuộc tính height.

Cả lớp **Circle** và **Cylinder** đều phải sử dụng **tính bao đóng** (**encapsulation**) để bảo vệ các thuộc tính của lớp khỏi việc truy cập trực tiếp từ bên ngoài.

**Lớp TestShape**:

* Tạo đối tượng của lớp **Circle** và **Cylinder**.
* In ra diện tích của hình tròn và thể tích của hình trụ.
* Thử thay đổi giá trị bán kính và chiều cao thông qua các phương thức setter và kiểm tra kết quả.

**[Khá 6] Xây dựng lớp Point và MoveablePoint**

**1. Mục tiêu:**

* Học cách sử dụng kế thừa trong Java để xây dựng các lớp có mối quan hệ giữa các đối tượng
* Cải thiện khả năng làm việc với các phương thức kế thừa và ghi đè (overriding methods)
* Áp dụng nguyên lý **bao đóng** (encapsulation) để bảo vệ các thuộc tính của lớp khỏi việc thay đổi trực tiếp từ bên ngoài
* Tìm hiểu cách sử dụng **constructor** và phương thức **super()** trong quá trình kế thừa
* Làm quen với việc xử lý các thao tác di chuyển đối tượng trong không gian tọa độ (x, y)

**2. Mô tả:**

Bài tập này yêu cầu sinh viên xây dựng hai lớp:

* **Lớp Point**: Đại diện cho một điểm trong không gian 2D với các thuộc tính x và y là các tọa độ.
* **Lớp MoveablePoint**: Lớp kế thừa từ lớp Point, đại diện cho một điểm có thể di chuyển trong không gian 2D. Lớp này sẽ thêm các thuộc tính xSpeed và ySpeed để mô phỏng tốc độ di chuyển theo phương x và y.

Lớp **MoveablePoint** sẽ kế thừa tất cả các phương thức và thuộc tính từ lớp **Point** và sẽ bổ sung thêm các phương thức để thay đổi tọa độ của điểm theo tốc độ di chuyển (xSpeed, ySpeed).

***Yêu cầu:***

* **Lớp Point**:
  + Thuộc tính x và y: Các tọa độ của điểm trong không gian 2D (dạng double).
  + Các phương thức getter và setter cho các thuộc tính x và y.
  + Phương thức toString(): Trả về chuỗi mô tả điểm theo định dạng (x, y).
* **Lớp MoveablePoint** kế thừa từ lớp Point:
  + Thuộc tính xSpeed và ySpeed: Tốc độ di chuyển của điểm theo trục x và y.
  + Các phương thức getter và setter cho các thuộc tính xSpeed và ySpeed.
  + Phương thức move(): Cập nhật tọa độ của điểm mới dựa trên tốc độ di chuyển (xSpeed, ySpeed).
    - Tọa độ x mới = tọa độ x hiện tại + xSpeed.
    - Tọa độ y mới = tọa độ y hiện tại + ySpeed.
  + Ghi đè phương thức toString() để trả về thông tin của điểm di chuyển theo định dạng (x, y) with speed (xSpeed, ySpeed).
* **Lớp TestMoveablePoint**:
  + Tạo đối tượng của lớp Point và MoveablePoint.
  + Hiển thị thông tin của điểm trước và sau khi di chuyển.
  + Kiểm tra việc thay đổi tọa độ và tốc độ di chuyển qua các phương thức setter.

**[Giỏi 7] Xây dựng lớp Student và GraduateStudent**

**1. Mục tiêu:**

* Hiểu và áp dụng các nguyên tắc cơ bản của tính kế thừa và tính bao đóng (Encapsulation) trong Java
* Xây dựng hệ thống quản lý thông tin sinh viên thông qua việc thiết kế các lớp có quan hệ cha-con
* Làm quen với cách sử dụng các thuộc tính và phương thức được bao đóng (private/protected) trong một lớp kế thừa

**2. Mô tả:**

Bài toán chúng ta quản lý các đối tượng **Student** và **GraduateStudent** gồm các thông tin sau:

**1. Lớp Student** (lớp cha)

* Thuộc tính:
  + name (tên sinh viên) – kiểu String.
  + id (mã sinh viên) – kiểu String.
  + gpa (điểm trung bình tích lũy) – kiểu double.
* Contructor: Khởi tạo constructor đầy đủ tham số
* Phương thức:
  + getDetails(): Trả về thông tin chi tiết của sinh viên (tên, mã sinh viên, điểm trung bình).

**2. Lớp GraduateStudent** (kế thừa từ Student)

* Thuộc tính:
  + researchTopic (đề tài nghiên cứu) – kiểu String.
  + supervisorName (tên giảng viên hướng dẫn) – kiểu String.
* Constructor: Khởi tạo constructor đầy đủ tham số. Sử dụng super để kế thừa thuộc tính lớp cha
* Phương thức:
  + Ghi đè phương thức **getDetails()** để trả về thông tin chi tiết bao gồm đề tài nghiên cứu và tên giảng viên hướng dẫn.

**3. Chức năng hiển thị thông tin sinh viên:**

* Tạo danh sách gồm cả sinh viên thường và sinh viên tốt nghiệp(Dùng mảng)
* Sử dụng vòng lặp để hiển thị thông tin của tất cả sinh viên bằng cách gọi phương thức **getDetails().**

***Yêu cầu:***

* Thiết kế lớp **Student** và **GraduateStudent**:
  + Các thuộc tính phải được bao đóng bằng từ khóa private hoặc protected.
  + Các phương thức truy cập (**getter**/**setter**) phải được sử dụng để thay đổi và truy xuất giá trị thuộc tính.
* Ghi đè phương thức (**getDetails**):
  + Sử dụng từ khóa **@Override** trong lớp **GraduateStudent**.
  + Sử dụng từ khóa **super** để gọi phương thức **getDetails** của lớp **Student**.
* Quản lý danh sách sinh viên:
  + Tạo một danh sách chứa cả các đối tượng **Student** và **GraduateStudent**.
  + Hiển thị thông tin của từng sinh viên bằng cách gọi phương thức **getDetails().**
* Kiểm tra tính hợp lệ:
  + Nếu gpa hoặc các chuỗi đầu vào (tên, đề tài nghiên cứu) trống hoặc không hợp lệ, hiển thị thông báo lỗi.

**[Giỏi 8] Xây dựng lớp Animals và các lớp kế thừa Dog và Cat**

**1. Mục tiêu:**

* Hiểu và áp dụng tính kế thừa (Inheritance) trong việc thiết kế các lớp trong Java
* Áp dụng tính bao đóng (Encapsulation) để bảo vệ dữ liệu và cung cấp các phương thức truy cập hợp lệ
* Sử dụng từ khóa super để gọi các phương thức và constructor của lớp cha

**2. Mô tả:**

Trong bài tập này chúng ta sẽ quản lý các đối tượng như Animals và Dog và Cat gồm các thông tin như sau:

* **Lớp Animals** (lớp cha): Mô tả các đặc điểm chung của động vật.
  + Thuộc tính:
    - name (tên động vật) – kiểu String.
    - age (tuổi động vật) – kiểu int.
  + Phương thức:
    - Constructor: Khởi tạo các thuộc tính cơ bản.
    - displayInfo(): Hiển thị thông tin chung về động vật.
    - makeSound(): Trả về “Some generic sound”
* **Lớp Dog** (kế thừa từ Animals):
  + Có thêm thuộc tính:
    - breed (giống chó) – kiểu String.
  + Ghi đè phương thức makeSound() để trả về âm thanh của chó (ví dụ: "Woof Woof").
  + Ghi đè phương thức displayInfo() để hiển thị thêm giống chó.
* **Lớp Cat** (kế thừa từ Animals):
  + Có thêm thuộc tính:
    - furColor (màu lông) – kiểu String.
  + Ghi đè phương thức makeSound() để trả về âm thanh của mèo (ví dụ: "Meow Meow").
  + Ghi đè phương thức displayInfo() để hiển thị thêm màu lông.
* **Chức năng hiển thị danh sách**:
  + Tạo một danh sách các đối tượng Animals chứa cả Dog và Cat.
  + Sử dụng vòng lặp để hiển thị thông tin của từng đối tượng và gọi phương thức makeSound() tương ứng.

***Yêu cầu:***

* Thiết kế lớp cha **Animals**:
  + Đảm bảo các thuộc tính được bảo vệ bằng từ khóa **private** hoặc **protected**.
  + Cung cấp các phương thức truy cập (**getter**/**setter**) để làm việc với thuộc tính.
* Kế thừa và ghi đè:
  + Các lớp con phải ghi đè phương thức **makeSound()** và **displayInfo().**
  + Sử dụng từ khóa **@Override** để xác định rõ ràng các phương thức được ghi đè.
* Quản lý danh sách động vật:
  + Tạo danh sách hoặc mảng chứa nhiều loại động vật khác nhau.
  + Hiển thị thông tin và âm thanh của mỗi đối tượng.
* Kiểm tra tính hợp lệ:
  + Xử lý các trường hợp không hợp lệ, ví dụ: tuổi âm hoặc tên trống.

**[Xuất sắc 9] Xây dựng hệ thống quản lý nhân viên với các lớp Employee, Manager và Developer trong Java**

**1. Mục tiêu:**

* Sinh viên sẽ áp dụng **bao đóng** và **kế thừa** trong lập trình Java để xây dựng một hệ thống quản lý nhân viên đơn giản
* Bài tập giúp sinh viên hiểu cách sử dụng các lớp con kế thừa từ lớp cha, ghi đè phương thức và sử dụng các phương thức trong việc tính toán lương và thông tin nhân viên
* Sinh viên sẽ thực hành cách sử dụng các lớp, phương thức, và đối tượng trong Java để mô phỏng một hệ thống thực tế

**2. Mô tả:**

Trong bài tập này chúng ta sẽ quản lý 3 đối tượng như sau:

* **Lớp Employee**:
  + Thuộc tính:
    - name: Tên nhân viên.
    - id: Mã số nhân viên.
    - salary: Lương cơ bản của nhân viên.
  + Constructors: Không tham số và đầy đủ tham số
  + Phương thức:
    - getSalary(): Trả về lương cơ bản của nhân viên.
    - increaseSalary(double amount): Tăng lương cho nhân viên.
    - toString(): Hiển thị thông tin nhân viên dưới dạng chuỗi (bao gồm tên, mã số và lương).
* **Lớp Manager** (Kế thừa từ lớp Employee):
  + Thuộc tính:
    - bonus: Tiền thưởng của quản lý.
  + Constructors: Không tham số và đầy đủ tham số
  + Phương thức:
    - getSalary(): Ghi đè phương thức từ lớp Employee để trả về lương của quản lí
    - toString(): Ghi đè phương thức từ lớp Employee để hiển thị thông tin quản lý, bao gồm tên, mã số, lương cơ bản và tiền thưởng.
* **Lớp Developer** (Kế thừa từ lớp Employee):
  + Thuộc tính:
    - programmingLanguage: Ngôn ngữ lập trình mà lập trình viên sử dụng.
  + Constructors: Không tham số và đầy đủ tham số
  + Phương thức:
    - getSalary(): Ghi đè phương thức từ lớp Employee để trả về lương cơ bản của lập trình viên
    - toString(): Ghi đè phương thức từ lớp Employee để hiển thị thông tin lập trình viên, bao gồm tên, mã số, lương cơ bản và ngôn ngữ lập trình.

***Yêu cầu:***

* Xây dựng các lớp **Employee**, **Manager**, và **Developer**:
  + Sinh viên phải tạo lớp **Employee** với các thuộc tính và phương thức như đã mô tả.
  + Sinh viên phải kế thừa lớp **Employee** và xây dựng các lớp **Manager** và Developer, mỗi lớp có phương thức ghi đè **getSalary()** và **toString()** để tính lương và hiển thị thông tin cho phù hợp với đặc thù của từng loại nhân viên.
* Tạo đối tượng và kiểm tra các phương thức:
  + Tạo ít nhất một đối tượng cho mỗi lớp **Employee**, **Manager**, và **Developer**.
  + Sử dụng phương thức **getSalary()** để in ra lương của mỗi nhân viên, quản lý và lập trình viên.
  + Sử dụng phương thức **toString()** để in thông tin chi tiết về nhân viên.
* Tăng lương cho một nhân viên và hiển thị lại thông tin:
  + Sử dụng phương thức **increaseSalary()** để tăng lương cho một nhân viên, sau đó in lại thông tin nhân viên sau khi đã tăng lương.

**[Xuất sắc 10] Xây dựng hệ thống quản lý hình học với các lớp Shape, Circle, Rectangle và Triangle trong Java**

**1. Mục tiêu:**

* Sinh viên sẽ áp dụng **bao đóng** và **kế thừa** trong lập trình Java để xây dựng hệ thống quản lý các hình học cơ bản (hình tròn, hình chữ nhật, tam giác)
* Bài tập này giúp sinh viên hiểu rõ cách sử dụng **kế thừa** để tạo ra các lớp con kế thừa từ lớp cha, sử dụng **ghi đè** **phương thức** để tính toán diện tích và chu vi của các hình
* Sinh viên sẽ thực hành việc xây dựng các lớp, sử dụng các phương thức trong Java để mô phỏng hệ thống hình học với tính kế thừa và bao đóng

**2. Mô tả:**

Trong bài tập này chúng ta sẽ quản lý các hình học với các thông tin như sau:

* **Lớp Shape** (lớp cha):
  + Thuộc tính:
    - name: Tên của hình
  + Constructor: Đầy đủ tham số
  + Phương thức:
    - calculateArea(): Phương thức để tính diện tích của hình. Trả  về 0.0
    - calculatePerimeter(): Phương thức để tính chu vi của hình. Trả về 0.0
    - toString(): Phương thức trả về thông tin chi tiết về hình (bao gồm tên của hình, diện tích và chu vi).
* **Lớp Circle** (kế thừa từ Shape):
  + Thuộc tính:
    - radius: Bán kính của hình tròn.
  + Constructor: Đầy đủ tham số
  + Phương thức:
    - calculateArea(): Tính diện tích hình tròn (diện tích = PI \* radius²).
    - calculatePerimeter(): Tính chu vi hình tròn (chu vi = 2 \* PI \* radius).
    - toString(): Ghi đè phương thức toString() của lớp Shape để hiển thị thông tin về hình tròn, bao gồm bán kính, diện tích và chu vi.
* **Lớp Rectangle** (kế thừa từ Shape):
  + Thuộc tính:
    - width: Chiều rộng của hình chữ nhật.
    - height: Chiều cao của hình chữ nhật.
  + Constructor: Đầy đủ tham số
  + Phương thức:
    - calculateArea(): Tính diện tích hình chữ nhật (diện tích = width \* height).
    - calculatePerimeter(): Tính chu vi hình chữ nhật (chu vi = 2 \* (width + height)).
    - toString(): Ghi đè phương thức toString() của lớp Shape để hiển thị thông tin về hình chữ nhật, bao gồm chiều rộng, chiều cao, diện tích và chu vi.
* **Lớp Triangle** (kế thừa từ Shape):
  + Thuộc tính:
    - a, b, c: Ba cạnh của tam giác.
  + Phương thức:
    - calculateArea(): Tính diện tích tam giác (sử dụng công thức Heron: diện tích = √(p \* (p - a) \* (p - b) \* (p - c)), trong đó p là nửa chu vi).
    - calculatePerimeter(): Tính chu vi tam giác (chu vi = a + b + c).
    - toString(): Ghi đè phương thức toString() của lớp Shape để hiển thị thông tin về tam giác, bao gồm ba cạnh, diện tích và chu vi.

***Yêu cầu:***

* Xây dựng các lớp **Shape**, **Circle**, **Rectangle**, và **Triangle**:
  + Sinh viên cần tạo lớp **Shape** với các phương thức **calculateArea()** và **calculatePerimeter().**
  + Sinh viên cần kế thừa lớp **Shape** để xây dựng các lớp **Circle**, **Rectangle**, và **Triangle**, mỗi lớp có các phương thức ghi đè **calculateArea(),** **calculatePerimeter(),** và **toString().**
* Kiểm tra tính đúng đắn của các phương thức:
  + Tạo ít nhất một đối tượng cho mỗi lớp **Circle**, **Rectangle**, và **Triangle**.
  + Sử dụng phương thức **calculateArea()** và **calculatePerimeter()** để in ra diện tích và chu vi của từng hình.
  + Sử dụng phương thức **toString()** để in ra thông tin chi tiết về các hình.
* Hiển thị và kiểm tra:
  + Tạo một mảng hoặc danh sách chứa các đối tượng **Shape** (bao gồm các đối tượng **Circle**, **Rectangle** và **Triangle**).
  + Duyệt qua danh sách để hiển thị diện tích và chu vi cho mỗi hình trong danh sách.