Managing Inheritance exercise

The purpose of this exercise is to practice building class structures using abstract classes and methods and using encapsulation, inheritance, and polymorphism.

Learning objectives

After completing the exercise, you'll be able to:

- Define and use abstract classes and methods
- Reinforce the concepts of Encapsulation, Inheritance, and Polymorphism

Evaluation criteria and functional requirements

- The project must not have any build errors.
- Code is clean and presented in an organized format.
- Code is appropriately encapsulated.
- Inheritance is appropriately used to avoid code duplication.
- Abstraction is appropriately used to prevent object creation and enforce needed method creation on inheritance.
- The code meets the defined specifications.

Notes for all classes

- All fields must be readonly fields. This means each field has a getter, but not a setter.

Paint calculator

Your goal is to create classes for a paint calculator. The calculator works by having a customer input dimensions of walls. Then, the calculator lets them know how many gallons of paint they need to purchase. Much of the UI is already complete. You need to implement the underlying `Wall` classes that make the calculations possible.

Each wall has a name, a color, and dimensions. The dimensions needed depend upon the shape of the wall. The application supports rectangular walls, square walls, and triangular walls.

To allow polymorphism and reuse common code, all the other walls inherit from the base `Wall` class. However, the `Wall` class doesn't include any dimensions of its own, so you can't use it in the application as a wall.

If you open `PaintCalculator`, you'll see there are several blocks of commented out code. After you complete a step, you'll need to uncomment the code. After that, the code compiles.

Step One: Implement the `Wall` class

The `Wall` class can't be directly instantiated. It has two instance variables, `name` and `color`, that are readonly. Add a constructor that looks like this:

```
public Wall(String name, String color)
```

It also has one public method that subclasses must implement. `getArea()` takes no parameters and returns an integer representing the total area of the wall.

To complete this step, you must:

- Verify that all tests in `/src/test/java/com/techelevator/WallTest` pass.
- Uncomment any code that's commented out following `// Step One:`.
- Run the application and perform any steps available.

Step Two: Implement the `RectangleWall` class

`RectangleWall` extends `Wall` and adds two new instance variables, `length` and `height`, that are readonly. Add a constructor that looks like this:

```
public RectangleWall(String name, String color, int length, int height)
```

`getArea()` returns the `length` multiplied by the `height`.

Add a `toString()` method that returns a `String` in the following format:

```
name (lengthxheight) rectangle
```

To complete this step, you must:

- Verify that all tests in `/src/test/java/com/techelevator/RectangleWallTest` pass.
- Uncomment any code that's commented out following `// Step Two:`.
- Run the application and perform any steps available.

Step Three: Implement the `SquareWall` class

`SquareWall` extends `RectangleWall`. Add a constructor that looks like this:

```
public SquareWall(String name, String color, int sideLength)
```

Add a `toString()` method that returns a `String` in the following format:

```
name (sideLengthxsideLength) square
```

To complete this step, you must:

- Verify that all tests in `/src/test/java/com/techelevator/SquareWallTest` pass.

- Uncomment any code that's commented out following `// Step Three:`.
- Run the application and perform any steps available.

Step Four: Implement the `TriangleWall` class

`TriangleWall` extends `Wall` and adds two new instance variables, `base` and `height`, that are readonly. Add a constructor that looks like this:

```
public TriangleWall(String name, String color, int base, int height)
```

`getArea()` returns the `base` multiplied by the `height` and then divided by two.

> Note: Dividing an `int` by another `int` rounds down the answer to the nearest whole number.

Add a `toString()` method that returns a `String` in the following format:

```
name (basexheight) triangle
```

To complete this step, you must:

- Verify that all tests in `/src/test/java/com/techelevator/TriangleWallTest` pass.
- Uncomment any code that's commented out following `// Step Four:`.
- Run the application and perform any steps available.

Tips and tricks

- There are less explicit details for this exercise, but you have everything you need to complete it. Do your best, and let the unit tests guide your work.
- Abstract classes can't be directly instantiated.
- If a method must be explicitly implemented by a non-abstract class, that means that the method must be abstract.