

Abstract classes and methods

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, students will 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 presented in a clean, organized format.
- Code is appropriately encapsulated.
- Inheritance is used appropriately to avoid code duplication.
- Abstraction is used appropriately to prevent object creation and enforce needed method creation on inheritance.
- The code meets the specifications defined below.

Notes for all classes

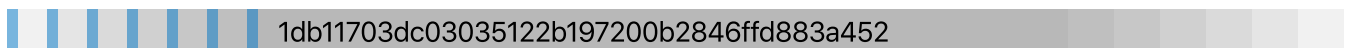
- Readonly attributes don't require a Setter.

<<<<<< HEAD

Paint Calculator

=====

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.

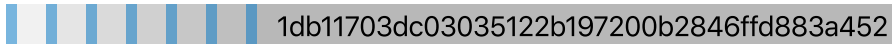
Each wall has a name, a color, and dimensions. The dimensions needed are based on the shape of the wall. The application supports rectangular walls, square walls, and triangular walls.

To allow polymorphism and reuse common code, a **Wall** class is implemented that all other walls inherit from. But it doesn't include any dimensions of its own and, therefore, can't be used in the application as a wall.

<<<<<< HEAD If you open up **PaintCalculator** you will notice that some of the code has been commented out and this is by design. As you complete a step you will be asked to uncomment the code. With the step completed the code should now compile.

Step One: `Wall`

===== If you open `PaintCalculator`, you'll see that some of the code has been commented out. After you complete a step, you'll be asked to uncomment the code. After that, the code compiles.

1db11703dc03035122b197200b2846ffd883a452

Step One: Implement the `Wall` class

The `Wall` class can't be 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.
- Remove any code that's commented out below `// Step One:`.
- Run the application and perform any steps available.

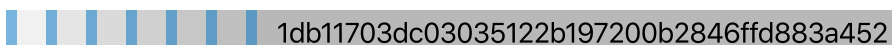
<<<<<< HEAD To verify you have completed Step One:

- All tests in `/src/test/java/com/techelevator/WallTest` should pass
- Remove any code that is commented out below `// Step One:`
- Run the application and perform any steps available

Step Two: `RectangleWall`

=====

Step Two: Implement the `RectangleWall` class

1db11703dc03035122b197200b2846ffd883a452

`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
```

<<<<<< HEAD To verify you have completed Step Two:

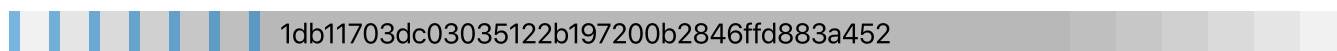
- All tests in `/src/test/java/com/techelevator/RectangleWallTest` should pass
- Remove any code that is commented out below // Step Two:
- Run the application and perform any steps available

Step Three: SquareWall

===== To complete this step, you must:

- Verify that all tests in `/src/test/java/com/techelevator/RectangleWallTest` pass.
- Remove any code that's commented out below `// 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
```

<<<<<< HEAD To verify you have completed Step Three:

- All tests in `/src/test/java/com/techelevator/SquareWallTest` should pass
- Remove any code that is commented out below `// Step Three:`
- Run the application and perform any steps available

Step Three: TriangleWall

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

To complete this step, you must:

- Verify that all tests in `/src/test/java/com/techelevator/SquareWallTest` pass.
- Remove any code that's commented out below `// 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:

```
1db11703dc03035122b197200b2846ffd883a452
```

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

<<<<<< HEAD `getArea()` returns the `base` multiplied by the `height` and then divided by 2. The answer should be rounded down to the nearest whole number.

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

Note: The answer is rounded down to the nearest whole number because you're dividing an `int` by another `int`.

```
1db11703dc03035122b197200b2846ffd883a452
```

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

```
name (basexheight) triangle
```

<<<<<< HEAD To verify you have completed Step Four:

- All tests in `/src/test/java/com/techelevator/TriangleWallTest` should pass
- Remove any code that is commented out below `// Step Four`:
- Run the application and perform any steps available

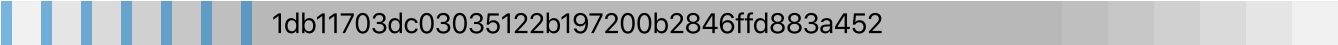
Tips & Tricks

- You've been given 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.
- If a class can't be instantiated, that means that it is an abstract class.
- If a method must be implemented by a non-abstract class, that means that the method must be abstract. ===== To complete this step, you must:
- Verify that all tests in `/src/test/java/com/techelevator/TriangleWallTest` pass.
- Remove any code that's commented out below `// Step Four`:
- Run the application and perform any steps available.

Tips and tricks

- You've been given 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.

- If a class can't be instantiated, that means that it's an abstract class.
- If a method must be implemented by a non-abstract class, that means that the method must be abstract.



1db11703dc03035122b197200b2846ffd883a452