Step-by-step:

1. In your code:

   ```
   1 / 2
   ```

   both `1` and `2` are **int** literals.
2. When both operands are `int`, Java uses **integer division** — it drops any fractional part, it doesn't round.

   So:

   ```
   1 / 2  →  0.5  → fractional part removed → 0
   ```

3. Now, when you multiply:

   ```
   0 * l * h  → always 0
   ```

Why Java does this

Java is strongly typed — the type of the result is determined by the types of the operands.

- `int / int → int`
- `double / int → double`
- `int / double → double`

If both are integers, you never get a decimal result — it's truncated toward zero.

Fix:

Make one of them `double`:

```
System.out.println(0.5 * l * h); // double literal System.out.println((1.0 / 2) * l * h);
// force double division System.out.println((l * h) / 2.0); // also works
```

If you like, I can show you **how Java decides the result type** in these expressions step-by-step, from compiler parsing to execution. That'll make the "why zero" crystal clear.