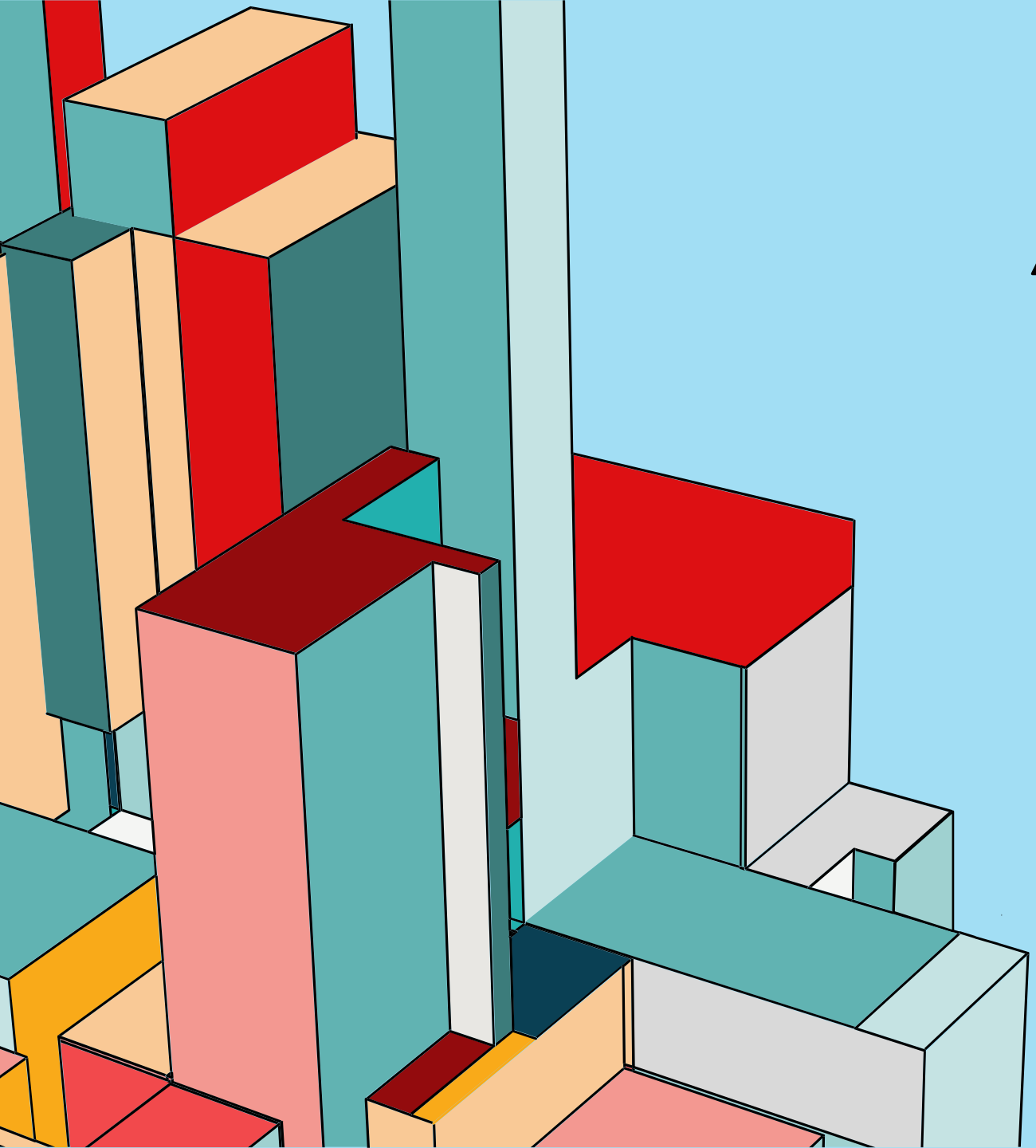
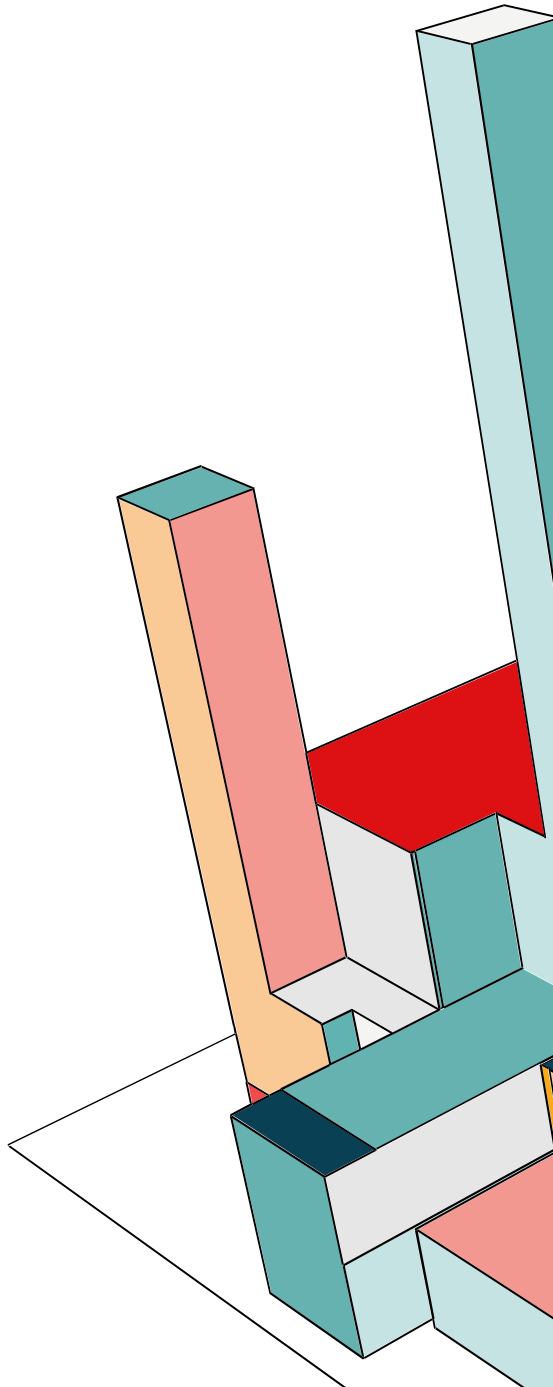


ALCOR PRESENTATION

MODULE 1

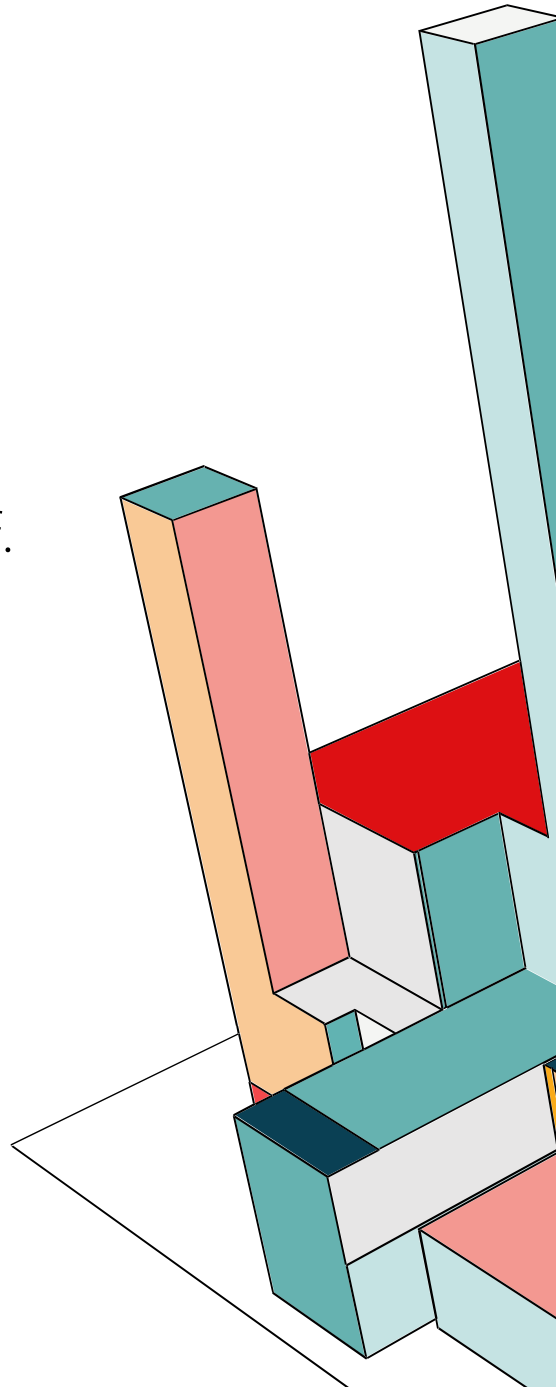
AGENDA

- Do not use getters/setters/properties
    - Purpose
    - Example 1
    - Example 2
- Wrap all primitives and strings
    - Purpose
    - Example



DO NOT USE GETTERS/SETTERS/PROPERTIES

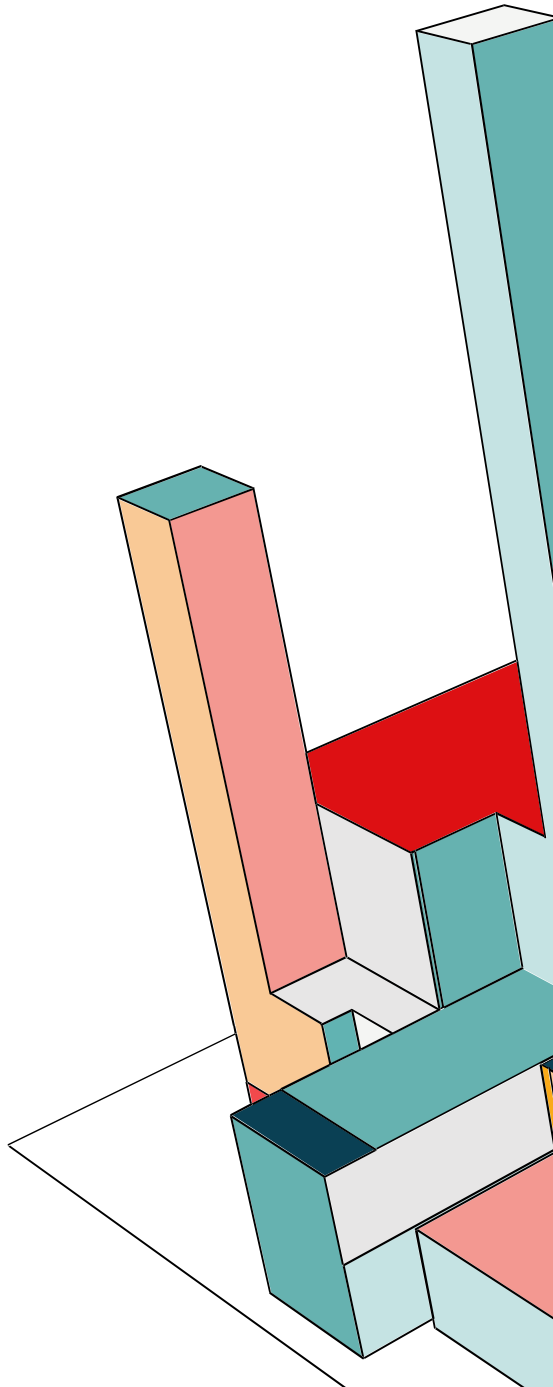
- May destroy encapsulation of your logic.
    - Unnecessary expose variables that are only relevant in the class itself.
    - Unnecessary coupling.
- May go against “Tell, don’t ask” principle.
    - DON’T: Query state from object and take action as a result.
    - DO: Issue an object a command to perform some logic.



EXAMPLE 1 - BAD

```
public class CandyShop{

    public int Money { get; set; } = 0;
}
```

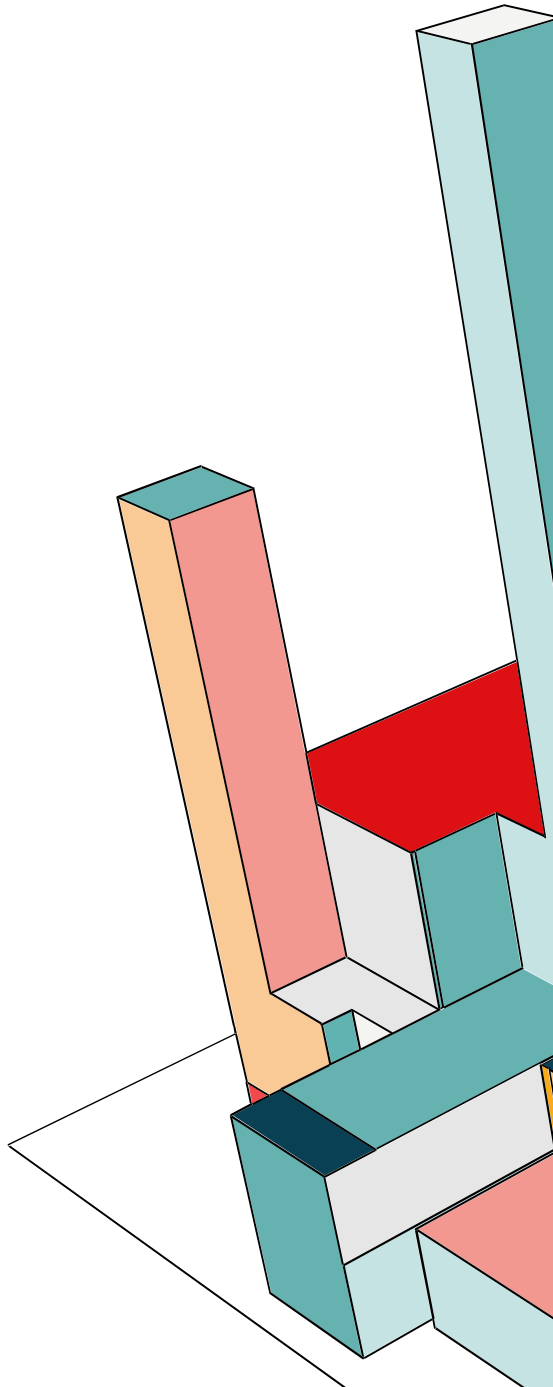
- Everybody with an instance of CandyShop can set money.



EXAMPLE 1 - GOOD

```
public class CandyShop{

    public int Money { get; private set; } = 0;

    public void BuyLollipop(int count)
        => Money -= Constants.LollipopBuyingPrice * count;

    public void SellLollipop(int count)
        => Money += Constants.LollipopSellingPrice * count;
}
```

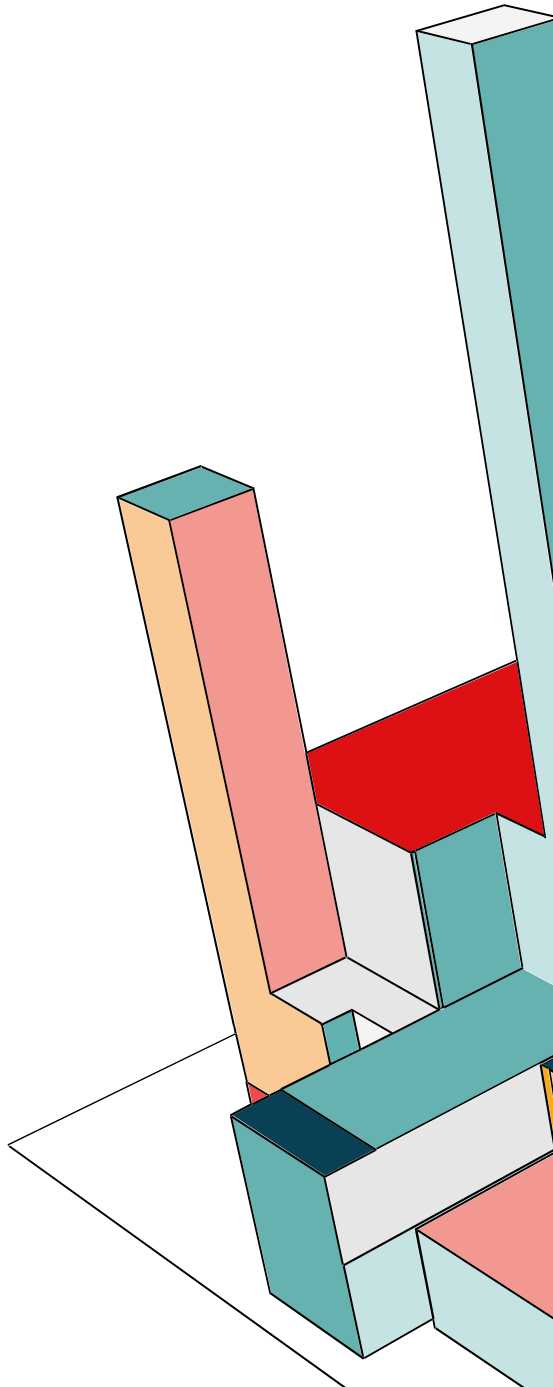
- The class exposes ways to interact with the money but keeps the logic to itself.
- We can add negative number checks.



EXAMPLE 2 - BAD

```
public class Player{

    public boolean alive { get; set; } = true;
}
```

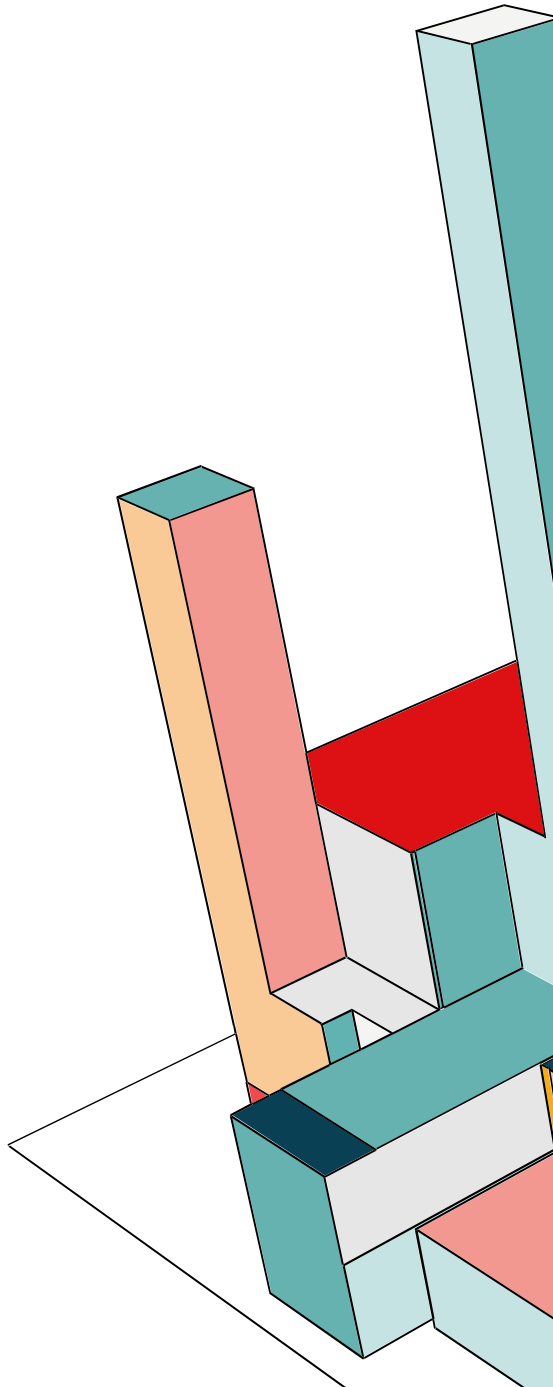
- Classes strongly coupled.
- What if we change to a “health point” system instead of a simple alive Boolean?
- Make changes in class itself and all classes that are using Player.



EXAMPLE 2 - GOOD

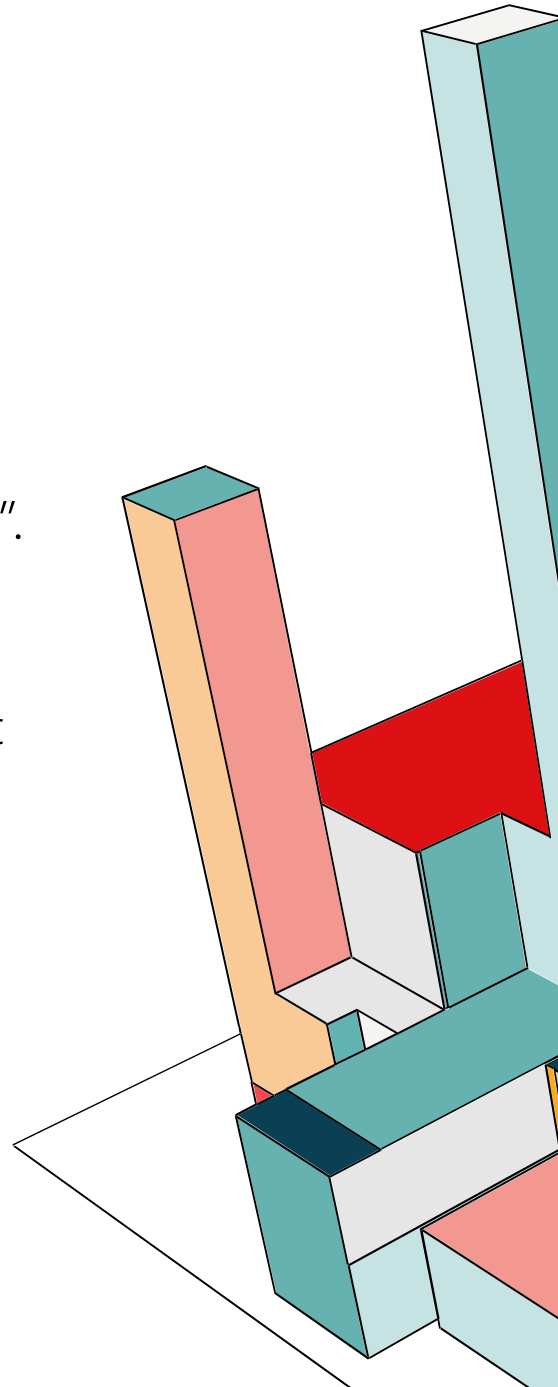
```
public class Player{

    private boolean alive = true;

    public boolean isAlive() { return alive; }
    public void kill() { alive = false; }
}

public class Player{

    private int hp; // Set in constructor.

    public boolean isAlive() { return hp > 0; } // Same method signature.
    public void kill() { hp = 0; } // Same method signature.
    public void damage(int damage) { hp -= damage; }
}
```

- Change implementation without breaking contract of the methods “isAlive” and “kill”.



WRAP ALL PRIMITIVES AND STRINGS

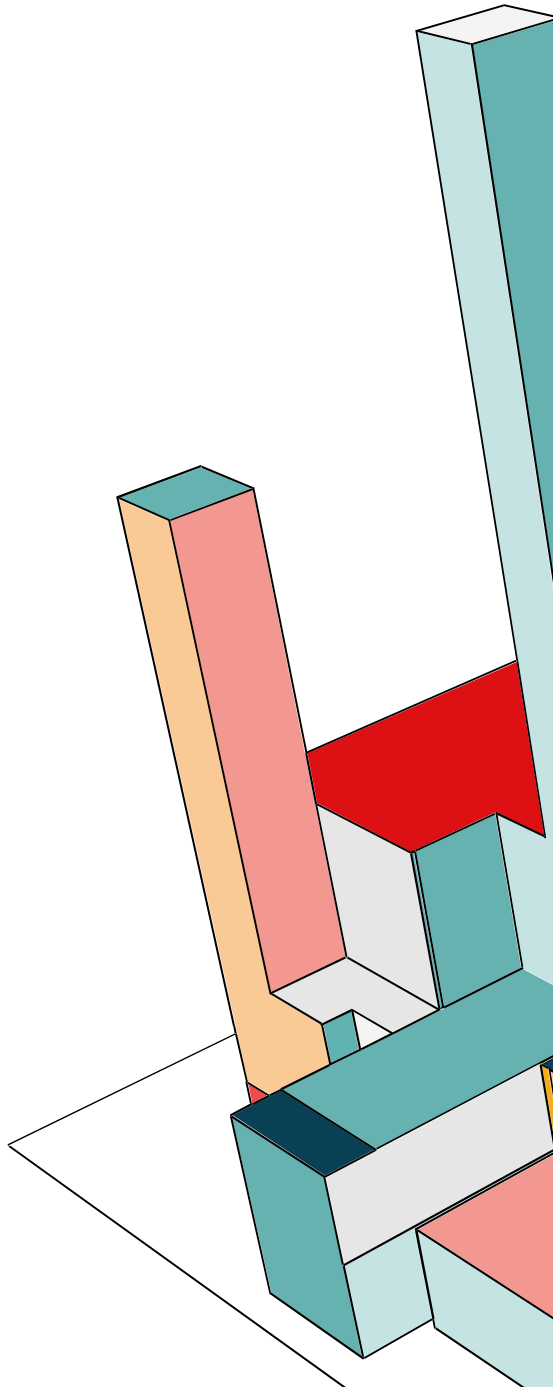
- Represent intention and make it easier to read and use.
- Question to ask: “Is my type holding more information than just its value?”.
- Get rid of code smell “Primitive obsession”.
    - DON’T: Use primitive to represent domain ideas (message, amount of money, …).
    - DO: Use ValueObjects.



EXAMPLE - BAD

```
int distanceInKm;
```

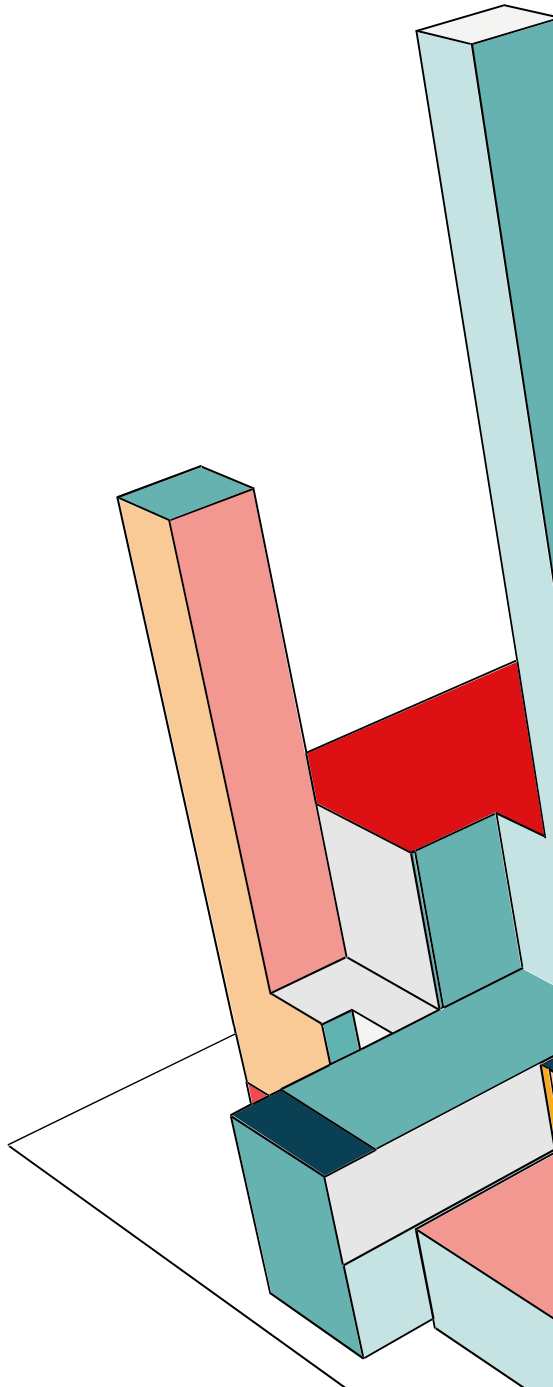
- What if we want to change it to miles?



EXAMPLE - GOOD

```
class Distance{

    private int _distanceInKm;

    public string DescriptionOfDistance() {return ...}
}
```

- Can be easily changed to miles everywhere in the code.



THANK YOU

QUESTIONS?

Dragan Jovanovic

dragan.jovanovic@css.ch