

ΟΟΠ

+ SOLID, KISS, DRY

14.01.2025

Организационная инфа

Парочка главных вопросов:

- *Как зовут преподавателя?* – Резуник Людмила Александровна
- *Какая формула?* – автомат: $(0.15 * \text{СЕМ} + 0.15 * \text{миниДЗ} + 0.6 * \text{КДЗ}) / 0.9$

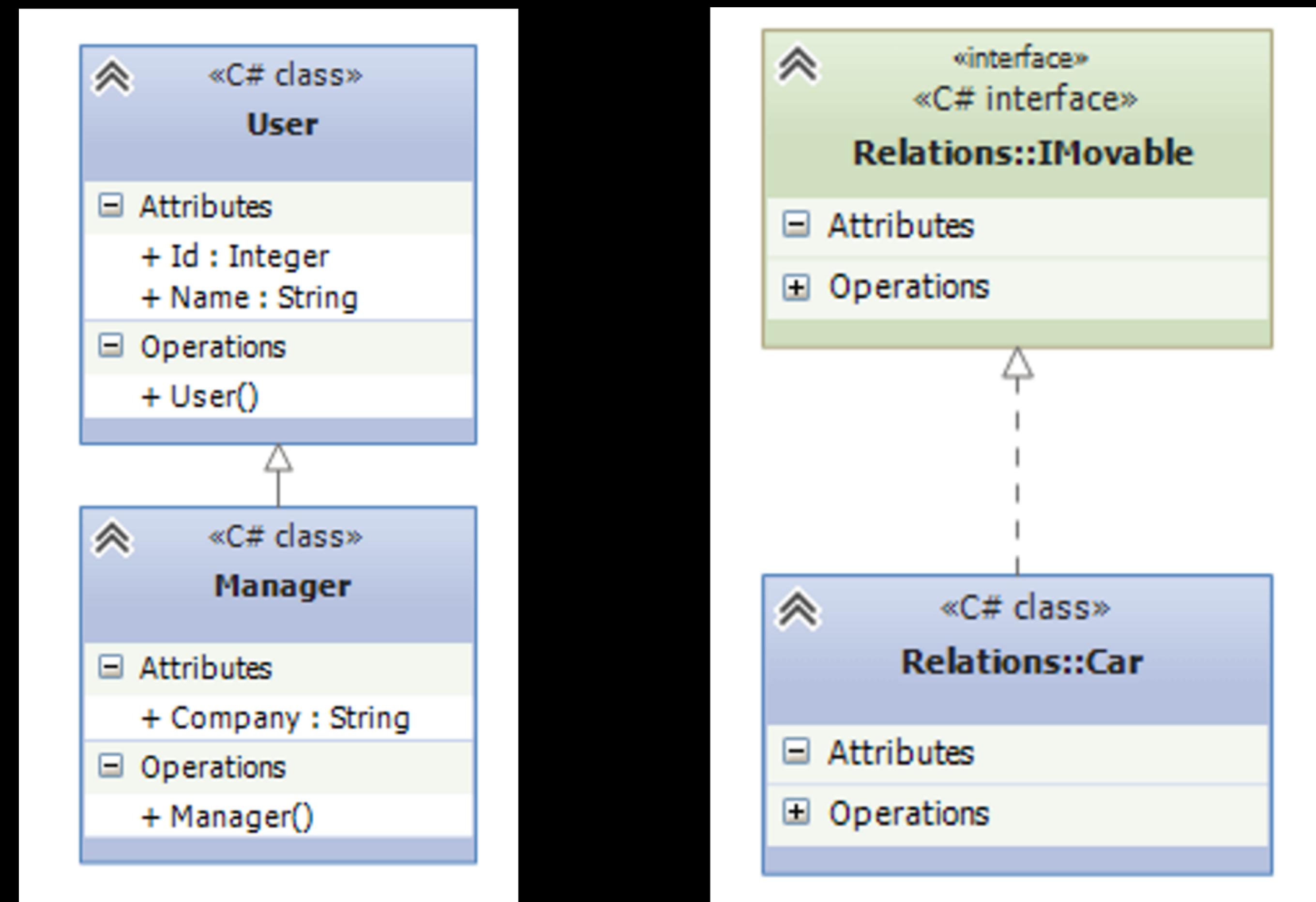
Вне семинара вопросы можно задавать в чатике или личке (@lucy_r)

Вспоминаем...

Отношения между классами – агрегация, композиция, ассоциация

Реализация vs Наследование

Перед этим стоит вспомнить про различия абстрактных классов и интерфейсов



```
1 public abstract class Vehicle
2 {
3     public abstract void Move();
4 }
5
6 public class Car : Vehicle
7 {
8     public override void Move()
9     {
10         Console.WriteLine("Машина едет");
11     }
12 }
13
14 public class Bus : Vehicle
15 {
16     public override void Move()
17     {
18         Console.WriteLine("Автобус едет");
19     }
20 }
21
22 public class Tram : Vehicle
23 {
24     public override void Move()
25     {
26         Console.WriteLine("Трамвай едет");
27     }
28 }
```

```
1 public interface IMovable
2 {
3     void Move();
4 }
5
6 public abstract class Vehicle : IMovable
7 {
8     public abstract void Move();
9 }
10
11 public class Car : Vehicle
12 {
13     public override void Move() => Console.WriteLine("Машина едет");
14 }
15
16 public class Bus : Vehicle
17 {
18     public override void Move() => Console.WriteLine("Автобус едет");
19 }
20
21 public class Horse : IMovable
22 {
23     public void Move() => Console.WriteLine("Лошадь скачет");
24 }
25
26 public class Aircraft : IMovable
27 {
28     public void Move() => Console.WriteLine("Самолет летит");
29 }
```

Пример

Студенты ВШЭ разработали инновационный автомобиль, сердцем которого стал прорывной педальный двигатель, работающий на силе духа и ног автовладельца. В связи с большим спросом на данное изобретение, студенты обратились к вам с просьбой автоматизировать процесс учета их продукции.

Информационная система должна **вести учет продукции – собранные студентами педальные автомобили и контролировать очередь из желающих приобрести данное ноу-хау**. В разрабатываемой информационной системе важно учесть, что инновационный автомобиль **не предполагается продавать без революционного педального двигателя**, а также любой произведенный автомобиль **изначально становится собственностью НИУ ВШЭ**.

Люди, которые желают приобрести данное чудо техники, как-то жили и без этого изобретения, а следовательно **они должны существовать вне зависимости от того, дождутся они своей очереди или нет**.

Технические требования к информационной системе учета педальных автомобилей:

- класс **Engine** содержит такую характеристику как «размер педалей» и имеет ассоциацию, реализованную в виде композиции с классом **Car**.
- класс **Car**, помимо включенного в него класса **Engine**, имеет порядковый номер произведенного автомобиля;
- класс **Customer** содержит ссылку на класс **Car** (между классами ассоциация в виде агрегации), а также ФИО;
- класс **FactoryAF** содержит коллекцию объектов **Customer** (между классами ассоциация в виде агрегации) и коллекцию объектов **Car** (связь в виде композиции). В данном классе реализовать метод **SaleCar()**, который должен «пробежаться» по всем клиентам в очереди на покупку и по возможности (при наличии) вручить каждому педальный автомобиль. При этом «врученный» автомобиль удаляется из коллекции автомобилей предприятия автотранспортного факультета.
В случае, если всем желающим выдали автомобиль, но они все равно остались на складе, то происходит их ликвидация (очистка коллекции автомобилей)

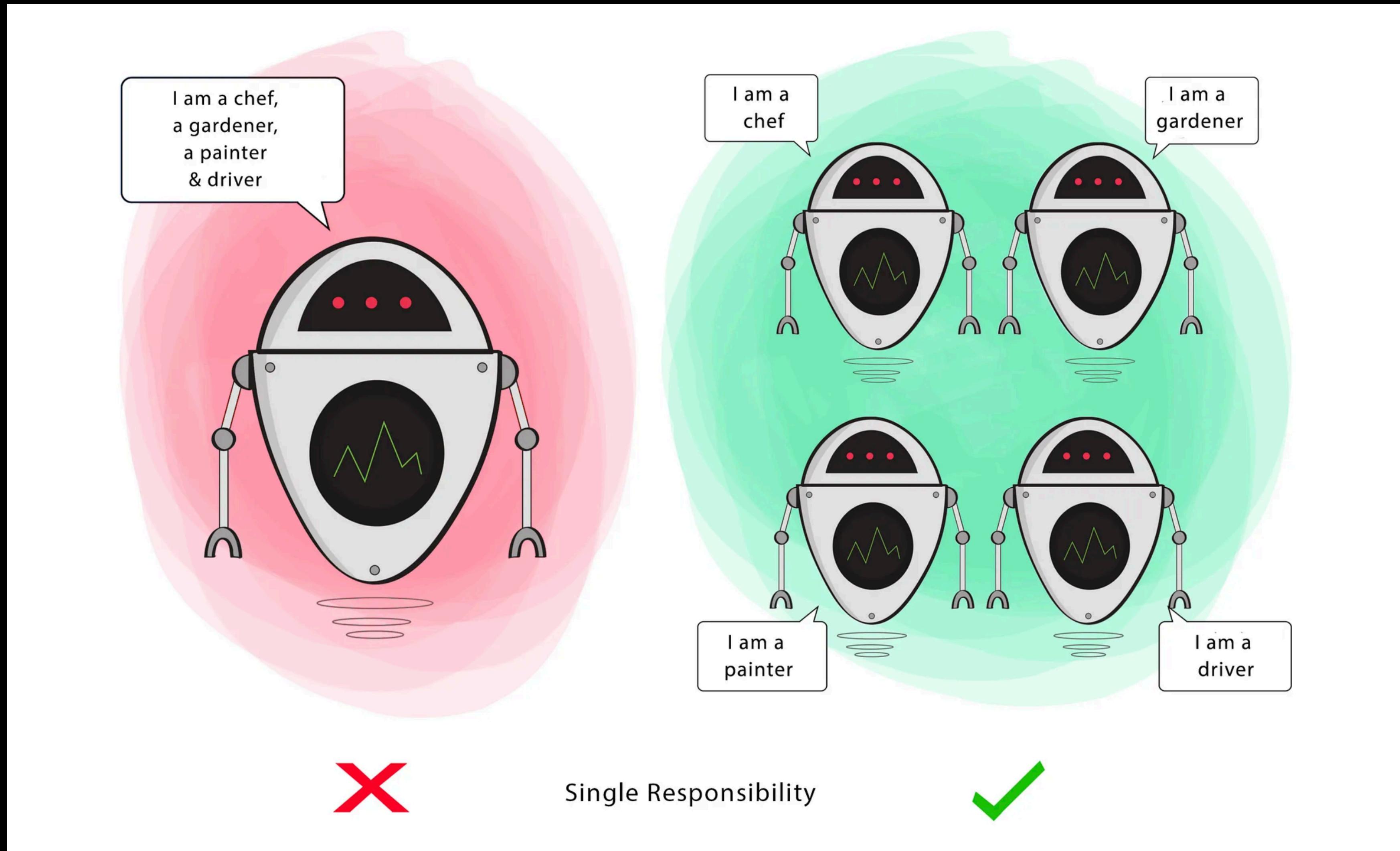
SOLID

- Описывает 5 принципов проектирования и программирования
- Каждая буква – отдельный принцип

SOLID

- S – Single responsibility principle (принцип единственной ответственности)

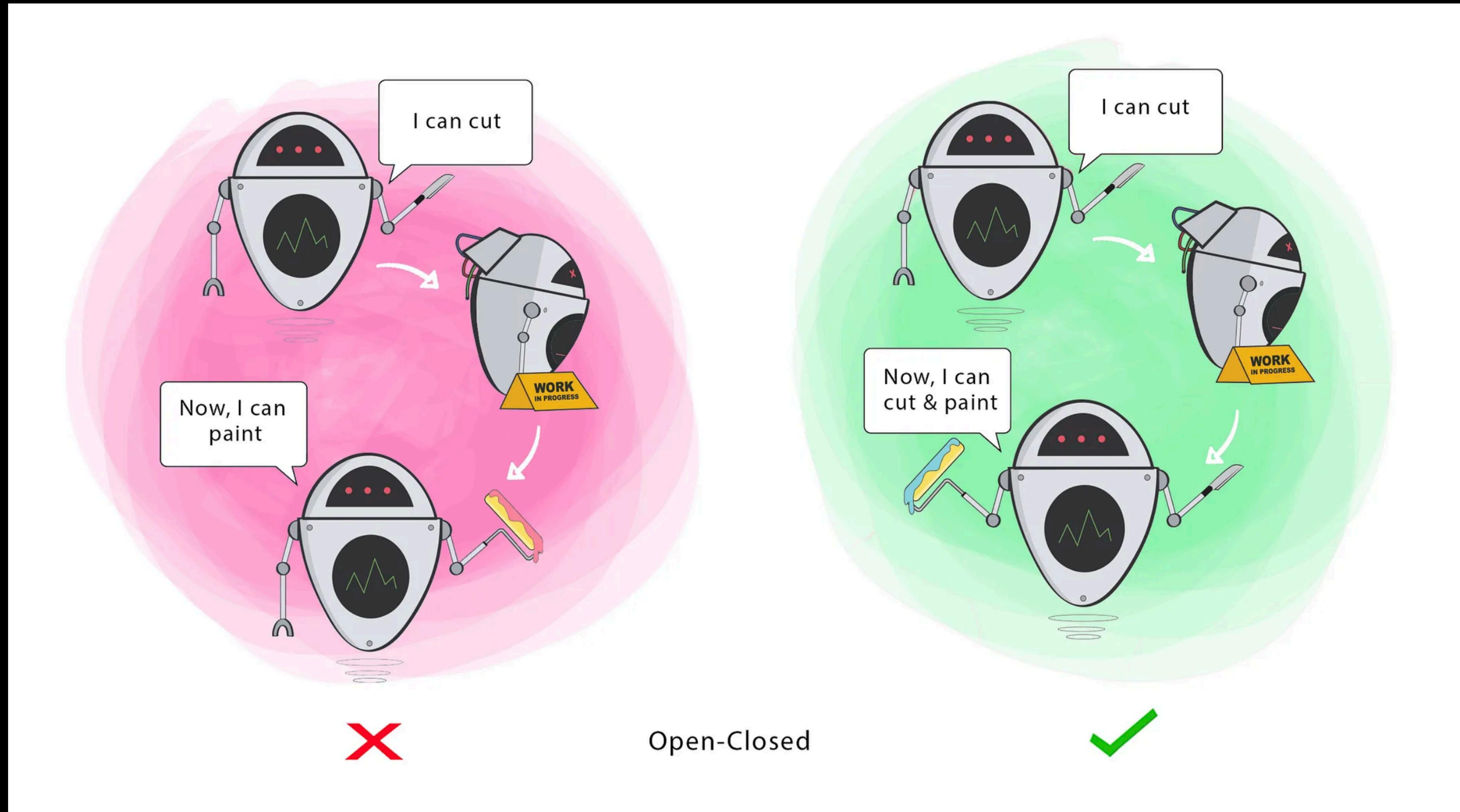
Если у класса много ответственостей, это увеличивает возможность возникновения багов (так как изменения в одном, могут повлиять на другое).



SOLID

- O – Open-closed principle (принцип открытого-закрытого)

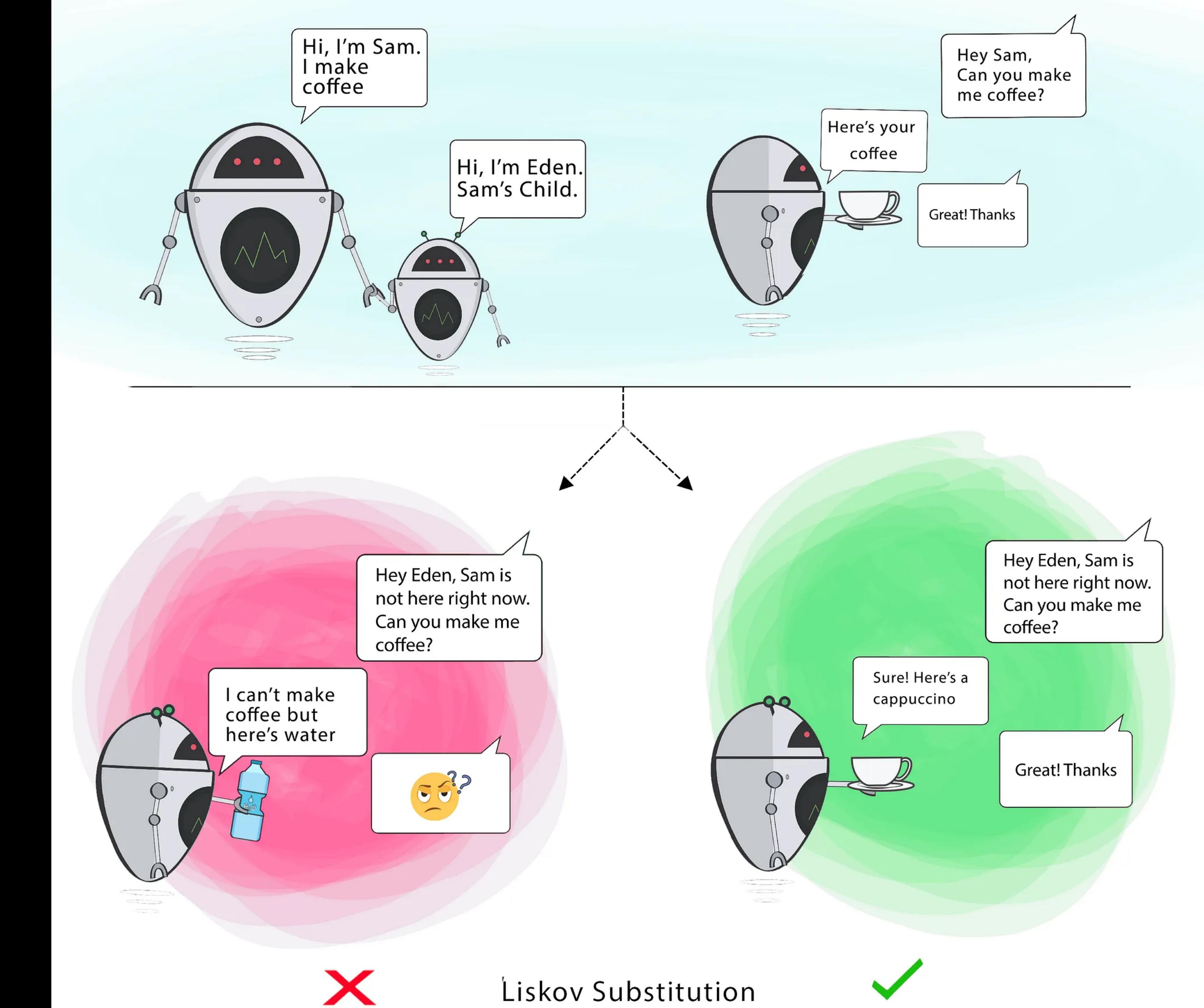
Классы должны быть открыты для расширения, но закрыты для изменения. Изменения в поведении класса воздействуют на все системы, что использовали этот класс. Стремимся добавлять функции, а не изменять их.



SOLID

- L – Liskov substitution principle (принцип подстановки Лисков)

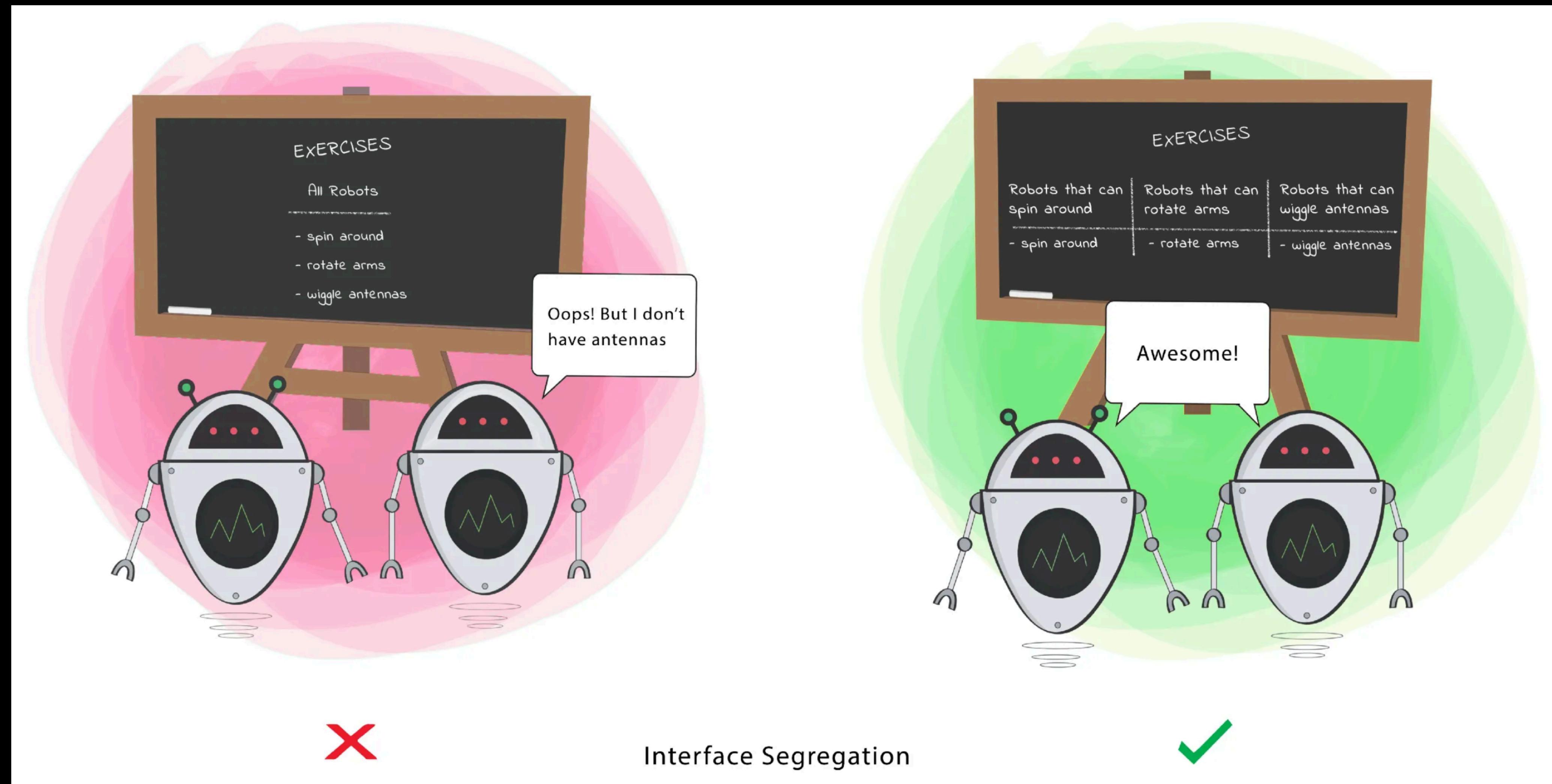
Если S – подтип T, то объекты типа T можно заменить на объекты типа S безболезненно и без каких-либо других изменений в программе.



SOLID

- I – Interface Segregation principle (принцип разделения интерфейса)

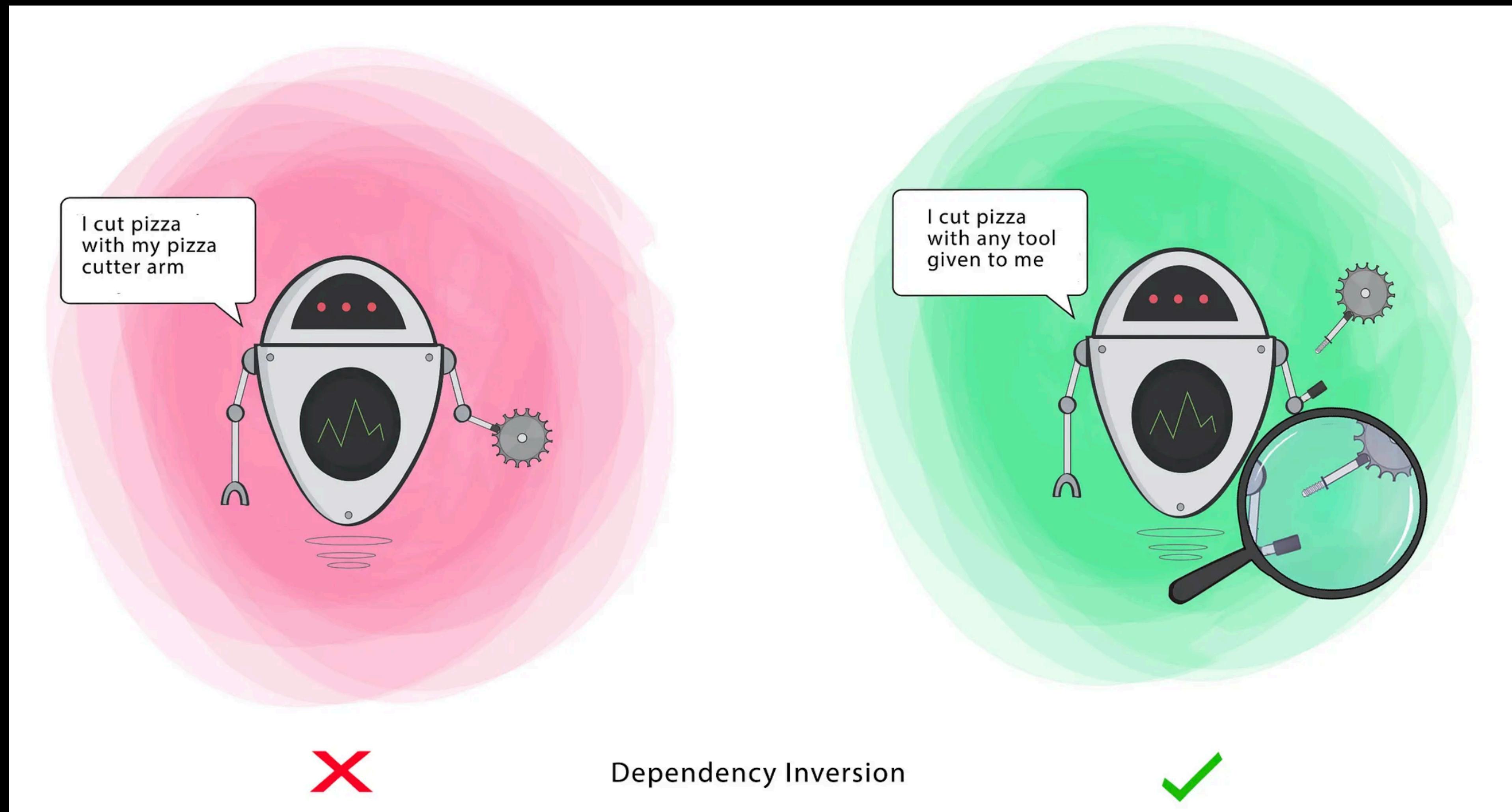
Клиентам не нужно навязывать зависимость от методов, которые они не используют. Класс должен выполнить только те действия, которые необходимы, чтобы выполнить роль. Выделяем более мелкие классы и интерфейсы.



SOLID

- D – Dependency Inversion principle (принцип инверсии зависимостей)

Модули верхних уровней не должны зависеть от модулей нижнего уровня. Абстракции не должны зависеть от деталей. Класс и используемый им инструмент лучше не объединять, а использовать интерфейс, который можно использовать с разными инструментами.



KISS

Keep it simple, stupid. («Делай проще, тупица») и другие аналоги

По самой фразе уже можно сделать вывод, что имеется ввиду:)

Там, где можно сделать проще, лучше сделать проще, чем выдумывать велосипед.

DRY

Don't repeat yourself. («Не повторяйся»)

Этот принцип настолько важен, что не требует повторения!

Разделяя систему на компоненты, можно добиться состояния, когда каждая часть системы отвечает только за определенные действия. Конкретно эти части должны иметь одно представление в коде.

