Принципы SOLID

|  |  |  |
| --- | --- | --- |
| **Буква** | **Принцип** | **Описание** |
| **S** | Single responsibility principle | Принцип единственной обязанности. Каждый объект должен иметь одну обязанность и эта обязанность должна быть полностью инкапсулирована в класс.  Обязанность класса можно понимать как ось (фактор) преимущественных изменений. Если есть у класса есть две причины для изменений, то его надо разделить. |
| **O** | Open/closed principle | Программные сущности должны быть открыты для расширения, но закрыты для изменения.  Этот принцип обеспечивается такими подходами:   * Если требуется изменить функционал класса нужно расширить его с помощью наследования, но не менять базовый класс; * Взаимодействие между подсистемами должно осуществляться через фиксированный интерфейс. Например, пускай класс А должен писать лог в текстовый файл и в базу данных. Класс А не должен использовать напрямую объекты того или иного вида логера. Оба логера должны реализовать общий интерфейс. Класс А принимает на вход общий интерфейс и использует универсальный способ логирования.   Этот принцип пересекается с принципом "Protected Variations" из GRASP |
| **L** | Liskov substitution principle | Функции, которые используют ссылки на базовые классы, должны иметь возможность использовать объекты производных классов, не зная об этом.  Это означает что, если S является подтипом T, тогда объекты типа T в программе могут быть замещены объектами типа S без нарушения работоспособности программы.  Подтип не должен разрушать контракты, предоставляемые базовым типом.  Поведение наследуемых классов не должно противоречить поведению, заданному базовым классом. |
| **I** | Interface segregation principle | Клиенты не должны зависеть от методов, которые они не используют.  Для достижения этого принципа нужно разделить интерфейсы на более специфические, если часть методов не нужна реализующему классу.  В итоге, при изменении метода интерфейса не должны меняться клиенты, которые этот метод не используют. |
| **D** | Dependency inversion principle | Модули верхних уровней не должны зависеть от модулей нижних уровней. Оба типа модулей должны зависеть от абстракций.  Этот принцип тесно связан с принципом "Low Coupling" из GRASP. Когда мы выполняем DIP, связность (зависимость между элементами) системы уменьшается.  На практике принцип означает, что использование конкретных объектов в классе лучше заменить ссылками на интерфейсы (абстракции). |

**Single responsibility principle**

Я старался применять этот принцип по всех классах, но соблюден этот принцип не везде. Так в Task16 классы FileManager, Calculator соответствуют принципу, тогда как класс Computer не соответствует.

Класс Computer кроме установки собственных параметров еще устанавливает операционную систему и приложения.

Установку приложений и операционной системы нужно реализовать в виде интерфейсов. Это также позволит соответствовать Dependency inversion principle и Open/closed principle.

**Open/closed principle**

С одной стороны, я везде использовал наследование и новые функции добавлялись путем наследования. С другой стороны, взаимодействие между подсистемами через фиксированный интерфейс не применялось.

Так в Task21 класс Desktop содержит объект VideoCard videoCard и метод setVideoCard(String type, double ramSize, String ramType) . Видеокарты могут быть разными, это потянет за собой изменения в классе Desktop.

Нужно вынести видеокарту в отдельный интерфейс и передать этот интерфейс классу Desktop.

**Liskov substitution principle**

Насколько я могу судить этот принцип не нарушался в существующем коде.

**Interface segregation principle**

Этот принцип соблюден в объеме учебных заданий. Так в Task21 существуют отдельно интерфейсы DesktopAccessories, LaptopAccessories.

**Dependency inversion principle**

Этот принцип нарушался чаще всего. Так в Task21 класс ComputerStore содержит объекты классов нижних уровней.

*/\*\*Collection of the desktop computers\*/***private** DesktopCatalog **desktopComputers**;  
*/\*\*Collection of the laptop computers\*/***private** LaptopCatalog **laptopComputers**;  
*/\*\*Collection of the tablet computers\*/***private** TabletCatalog **tabletComputers**;  
  
*/\*\*Customer cart\*/***private** Cart<Computer> **customerCart**;

В классах, которые собирают в себе много других сущностей лучше заменить использование конкретных объектов в классе ссылками на интерфейсы.

1. <https://ru.wikipedia.org/wiki/SOLID_%28%D0%BE%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29>