ООП — основные принципы

Java Core (/quests/QUEST_JAVA_CORE)
1 уровень (/quests/lectures/?quest=QUEST_JAVA_CORE&level=1), 1 лекция (/quests/lectures/questcore.level01.lecture01)

ОТКРЫТА

— Привет, Амиго! Сегодня я открою для тебя новый и интересный мир. Этот мир называется ООП – объектно-ориентированное программирование. Ты уже познакомился с классами и объектами. Сегодня ты узнаешь про них больше, намного больше.

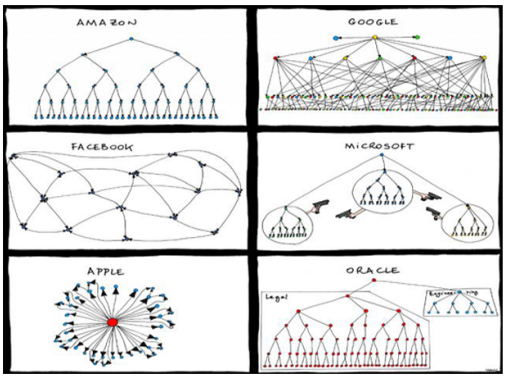
Начнем с четырех принципов ООП. Это: абстракция, инкапсуляция, наследование и полиморфизм. (Раньше их было три, но потом решили добавить абстракцию)

1) Абстракция.

Хорошим примером абстракции в реальной жизни является описание должностей в компании или организации. Название должности – это одно, а обязанности каждой конкретной должности – это уже совсем другое.

Представь, что ты проектируешь структуру своей будущей компании. Ты можешь разделить обязанности секретаря: «раскидать» их по нескольким другим должностям. Можешь разбить должность исполнительного директора на несколько независимых должностей: финансовый директор, технический директор, директор по маркетингу, директор по персоналу. Или, например, объединить должности офис-менеджера и рекрутера в одну.

Ты придумываешь названия должностей в своей фирме, а потом «раскидываешь» обязанности по этим должностям. Это и есть абстракция – разбиение чего-то большого, монолитного на множество маленьких составных частей.



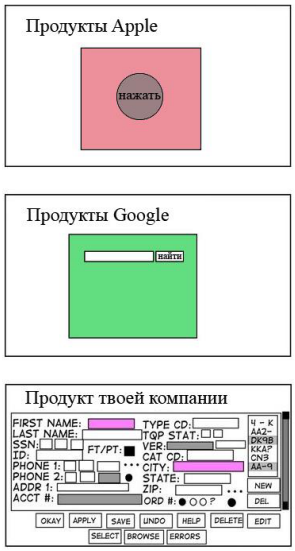
С точки зрения программирования, абстракция — это, скажем так, правильное разделение программы на объекты.

Обычно любую большую программу можно десятками способов представить в виде взаимодействующих объектов. Абстракция позволяет отобрать главные характеристики и опустить второстепенные.

Абстракция – это как стратегия в военном деле. Плохая стратегия – и никакой гениальной тактикой ситуацию уже не исправить.

2) Инкапсуляция.

Цель инкапсуляции – улучшить качество взаимодействия вещей за счет упрощения их.



А лучший способ упростить что-то – это скрыть все сложное от посторонних глаз. Например, если тебя посадят в кабину Боинга, ты не сразу разберешься, как им управлять:

С другой стороны, для пассажиров самолета все выглядит проще: купил билет, сел в самолет, взлетели и приземлились. Ты можешь с легкостью перелететь с континента на континент, обладая только навыками «купить билет» и «сесть на самолет». Все сложности в виде подготовки самолета к полету, взлета, посадки и различных внештатных ситуаций скрыты от нас. Не говоря уже о спутниковой навигации, автопилоте и диспетчерских центрах в аэропортах. И это упрощает нам жизнь.

С точки зрения программирования, инкапсуляция – это «сокрытие реализации». Мне нравится такое определение. Наш класс может содержать сотни методов и реализовывать очень сложное поведение в различных ситуациях. Но мы можем скрыть от посторонних глаз все его методы (пометить модификатором private), а для взаимодействия с другими классами оставить всего пару-тройку методов (пометить их модификатором public). Тогда все остальные классы нашей программы будут видеть в этом классе всего три метода, и будут вызывать именно их. А все сложности будут скрыты внутри класса, как кабина пилотов от счастливых пассажиров.

3) Наследование.

У наследования есть две стороны. Сторона программирования и сторона реальной жизни. С точки зрения программирования, наследование – это специальное отношение между двумя классами. Но гораздо интереснее, что же такое наследование с точки зрения реальной жизни.

Если бы нам понадобилось что-то создать в реальной жизни, то у нас есть два решения:

1) создать нужную нам вещь с нуля, потратив кучу времени и сил.

2) создать нужную нам вещь на основе уже существующей.

Наиболее оптимальная стратегия выглядит так: берем существующее хорошее решение, немного его дорабатываем, подгоняем под свои нужды и используем.

Если мы проследим историю возникновения человека, то окажется, что с момента зарождения жизни на планете прошли миллиарды лет. А если представить, что человек возник из обезьяны (на основе обезьяны), то прошла всего пара миллионов лет. Создание с нуля – дольше. Гораздо дольше.

В программировании тоже есть возможность создавать один класс на основе другого. Новый класс становится потомком (наследником) уже существующего. Это очень выгодно, когда есть класс, который содержит 80%-90% нужных нам данных и методов. Мы просто объявляем подходящий класс родителем нашего нового класса, тогда в новом классе автоматически появляются все данные и методы класса-родителя. Правда, удобно?

4) Полиморфизм.

Полиморфизм – это понятие из области программирования. Оно описывает ситуацию, когда за одним интерфейсом скрываются разные реализации. Если постараться поискать его аналоги в реальной жизни, то одним из таких аналогов будет процесс управления машиной.

Если человек может управлять грузовиком, то его можно посадить и за руль скорой, и за руль спорткара. Человек может управлять машиной вне зависимости от того, что это за машина, потому что все они имеют одинаковый интерфейс управления: руль, педали и рычаг коробки передач. Внутреннее устройство машин разное, но все они имеют одинаковый интерфейс управления.

Если вернуться к программированию, то полиморфизм позволяет единообразно обращаться к объектам различных классов (обычно имеющих общего предка) – вещь, которую трудно переоценить. Ценность его тем выше, чем больше программа.

ООП – это принципы. Внутренние законы. Каждый из них нас в чем-то ограничивает, давая взамен большие преимущества, когда программа вырастает до больших размеров. Четыре принципа ООП – это как четыре ножки стула. Убери хотя бы одну, и вся система станет неустойчивой.

‹ (/quests/lectures/questcore.level01.lecture00)

×11 › (/quests/lectures/questcore.level01.lecture02)

0 0 0 0 0 +62

Комментарии (28) популярные новые старые

Никита

Введите текст комментария

Dmitry Ivanov 15 уровень, Москва 19 февраля, 18:04

Хорошее объяснение, только:
1. по сути самое начало немного доработать (всё-таки, это больше про декомпозицию) и
2. по форме самый конец - три ножки вполне устойчиво будет :)

Ответить 0

Сергей 13 уровень 16 февраля, 13:51

Поправьте пожалуйста определение абстракции, оно ведь совершенно неверное и вводит в заблуждение.

То что в лекции описывается как абстракция это декомпозиция, и отношения к ООП не имеет никакого.

Ответить +2

Ralik 11 уровень 9 января, 10:33

"Наиболее оптимально" - нет такого понятия: бывает только один оптимальный вариант. Поправьте пожалуйста.

Ответить 0

sergey sergeev 13 уровень, Москва 20 декабря 2017, 20:13

ну ну, отличная. Все что тут написано и есть абстракция. Вот вообще не понятно как это к программированию относится. А вот когда увидишь код, тога становится понятней.

Ответить +2

vinsler 27 уровень, Санкт-Петербург 14 декабря 2017, 20:29

Лекция отличная, не на 5+ на дипломе, но вполне отличная.)) Я бы доработал с примерами, рисунками и некоторыми обьяснениями, но вполне доступно и понятно. +1

Ответить +1

Ильяс 23 уровень, Москва 21 октября 2017, 10:25

Отличная лекция! Мне нравится как они простыми словами пытаются объяснить то, что в интернете я встречал довольно сложными словами.

Всегда найдутся те, кто скажет, что чем-то недоволен.

Ответить +2

Shohrux Yusupov 12 уровень 15 сентября 2017, 00:52

да что с вами, народ? **нормальное объяснение.** читаю без повтора. не знаю как еще можно упростить

Ответить +6

Roman Beskrovnyi 33 уровень, Харьков 12 сентября 2017, 23:00

"Убери хотя бы одну, и вся система станет неустойчивой."...
Серьёзно? Плоскость строится по трем точкам... думаю здесь можно как-то подправить.

Ответить +9

Fonzy 32 уровень, Москва 2 октября 2017, 11:40

Если убрать одну ножку у стула, который задуман, как стул с 4 ножками, то да, действительно стул станет неустойчивым.

Ответить +9

Егор Кореньков 13 уровень, Севастополь 5 ноября 2017, 09:27

только у стула есть еще и центр тяжести и он лежит не в плоскости соприкосновения ножек с полом

Ответить 0

Сергей Ирюпин 24 уровень 23 августа 2017, 16:58

Инкапсуляция – это «сокрытие реализации». Ага, а как насчёт упаковки данных и кода в единый компонент? Это ведь тоже важная составная часть инкапсуляции.

```
1   Object Encapsulation = Hiding(Data, Methods) + Packing(Data, Methods);
```

Ответить 0

sergey sergeev 13 уровень, Москва 20 декабря 2017, 20:16

Мда, гуру объявились смотрю. Инкапсуляция - это скрыть то, что должно быть скрыто и показать то, что должно быть доступно. В общем смысле это приватные переменные в классе (которые должны быть скрыты) и публичные методы для работы с ними.

Ответить 0

Сергей Ирюпин 24 уровень 21 декабря 2017, 21:37

то есть по-вашему, инкапсуляция - только сокрытие? а как насчёт связывания? не?

Ответить 0

sergey sergeev 13 уровень, Москва 25 декабря 2017, 08:55

Не. Читать умеете?

Encapsulation is the technique of making the fields in a class private and providing access to the fields via public methods. If a field is declared private, it cannot be accessed by anyone outside the class, thereby hiding the fields within the class. For this reason, encapsulation is also referred to as data hiding.

Encapsulation can be described as a protective barrier that prevents the code and data being randomly accessed by other code defined outside the class. Access to the data and code is tightly controlled by an interface.

The main benefit of encapsulation is the ability to modify our implemented code without breaking the code of others who use our code. With this feature Encapsulation gives maintainability, flexibility and extensibility to our code.

Ответить 0

Сергей Ирюпин 24 уровень 26 декабря 2017, 12:22

А если прочесть ещё и вот это?

Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit. In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class. Therefore, it is also known as data hiding.

https://www.tutorialspoint.com/java/java_encapsulation.htm

Ответить 0

sergey sergeev 13 уровень, Москва 27 декабря 2017, 13:46

Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit.

Мда... начнем с первого твоего утверждения: а как насчёт упаковки данных и кода в единый компонент?

Где там было слово "компонент"? Для программиста слово "компонент" и собирательное выражение "отдельная единица" имеет весьма разное значение.

Дальше читаем по твоей ссылке:
To achieve encapsulation in Java
− Declare the variables of a class as private.
- Provide public setter and getter methods to modify and view the variables values.

Как видно, основной упор в разъяснении что такое инкапсуляция сделан не на некий "компонент" или "упаковку данных", а на доступ к переменным и на публичные геттеры\сеттеры. То есть ровно на то, о чем я написал в своем первом комментарии. Причем после прочтения книги по подготовке к OCA, я видел только такое определение. Что тоже косвенно подтверждает незначительность утверждения о некой "упаковке данных" и уж тем более о существовании в синтаксисе Java каких-то выдуманных тобой компонентов. Данный спор мне не интересен, так что если не отвечу на твой следующий выпад, не считай что у меня кончились аргументы.
Тем более в java есть КЛАСС Component, поэтому слово "компонент" нельзя использовать в общем контексте, не уточнив что именно подразумевается. Ты походу плохо синтаксис языка знаешь.

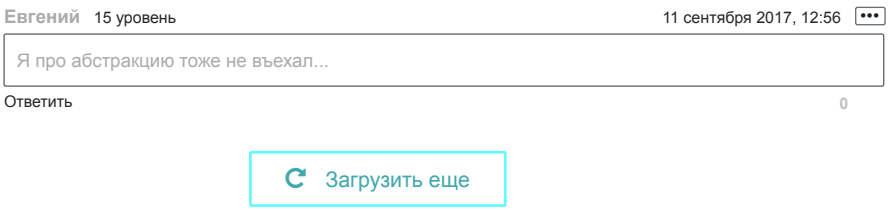
Ответить 0

Alexey Adver Lyovin 23 уровень, Москва 6 июля 2017, 16:56

> разбиение чего-то большого, монолитного на множество маленьких составных частей

Всю жизнь думал, что это декомпозиция, а тут вот оно как.

Ответить +13



ush.ru/) (https://plus.google.com/114772402300089087607) (https://twitter.com/javarush_ru)

JAVARUSH
Программистами не рождаются
© 2018