**Инкапсуляция**

[Java Core](https://javarush.ru/quests/QUEST_JAVA_CORE)

[Уровень 1](https://javarush.ru/quests/lectures?quest=QUEST_JAVA_CORE&level=1), Лекция 7

— Привет, Амиго! Хочу посвятить сегодняшнюю лекцию инкапсуляции. Ты уже знаешь в общих чертах, что это такое.

[](https://cdn.javarush.ru/images/article/3643bfbf-3555-4a07-a18f-abd3db9abd5e/original.jpeg)

В чем же преимущества инкапсуляции? Их достаточно много, но я могу выделить четыре, на мой взгляд, основных:

**1) Валидное внутреннее состояние.**

В программах часто возникают ситуации, когда несколько классов, взаимодействуют с одним и тем же объектом. В результате их совместной работы нарушается целостность данных внутри объекта — объект уже не может продолжить нормально работать.

Поэтому объект должен следить за изменениями своих внутренних данных, а еще лучше – проводить их сам.

Если мы не хотим, чтобы какая-то переменная класса менялась другими классами, мы объявляем ее **private**, и тогда только методы её же класса смогут получить к ней доступ. Если мы хотим, чтобы значения переменных можно было только читать, но не изменять, тогда нужно добавить **public getter** для нужных переменных.

Например, мы хотим, чтобы все могли узнать количество элементов в нашей коллекции, но никто не мог его поменять без нашего разрешения. Тогда мы объявляем переменную **private int count** и метод **public getCount()**.

Правильное использование инкапсуляции гарантирует, что ни один класс не может получить прямой доступ к внутренним данным нашего класса и, следовательно, изменить их без контроля с нашей стороны. Только через вызов методов того же класса, что и изменяемые переменные.

Лучше исходить из того, что другие программисты всегда будут использовать твои классы самым удобным для них образом, а не самым безопасным для тебя (для твоего класса). Отсюда и ошибки, и попытки заранее избавиться от них.

**2) Контроль передаваемых аргументов.**

Иногда нужно контролировать аргументы, передаваемые в методы нашего класса. Например, наш класс описывает объект «человек» и позволяет задать дату его рождения. Мы должны проверять все передаваемые данные на их соответствие логике программы и логике нашего класса. Например, не допускать 13-й месяц, дату рождения 30 февраля и так далее.

— А зачем кому-то указывать в дате рождения 30 февраля?

— Во-первых – это может быть ошибка ввода данных от пользователя.

Во-вторых, прежде чем программа будет работать как часы, в ней будет много ошибок. Например, возможна такая ситуация.

Программист пишет программу для определения людей, у которых день рождения послезавтра. Например, сегодня 3 марта. Программа добавляет к текущему дню месяца число 2 и ищет всех, кто родился 5 марта. Вроде бы все верно.

Вот только, когда наступит 30 марта программа не найдет никого, т.к. в календаре нет 32 марта. В программе становится гораздо меньше ошибок, когда в методы добавляют проверку переданных данных.

— Помню, когда мы изучали ArrayList, я смотрел его код, и там была проверка индекса в методах get и set: index больше или равен нулю и меньше длины массива. Там еще кидалось исключение, если в массиве нет элемента с таким индексом.

— Да, это классический пример проверки входных данных.

**3) Минимизация ошибок при изменении кода классов.**

Представим, что мы написали один очень полезный класс, когда участвовали в большом проекте. Он так всем понравился, что другие программисты начали использовать его в сотнях мест в своем коде.

Класс оказался настолько полезен, что ты решил его улучшить. Но если ты удалишь какие-то методы этого класса, то код десятков людей перестанет компилироваться. Им придется срочно все переделывать. И чем больше переделок, тем больше ошибок. Ты поломаешь кучу сборок, и тебя будут ненавидеть.

А когда мы меняем методы, объявленные как private, мы знаем, что нигде нет ни одного класса, который вызывал бы эти методы. Мы можем их переделать, поменять количество параметров и их типы, и зависимый код будет работать дальше. Ну, или как минимум, компилироваться.

**4) Задаем способ взаимодействия нашего объекта со сторонними объектами.**

Мы можем ограничить некоторые действия, допустимые с нашим объектом. Например, мы хотим, чтобы объект можно было создать только в одном экземпляре. Даже если его создание происходит в нескольких местах проекта одновременно. И мы можем сделать это благодаря инкапсуляции.

[](https://cdn.javarush.ru/images/article/85b1f2fa-09fe-4777-890e-05d3a824a2bf/original.jpeg)

Инкапсуляция позволяет добавлять дополнительные ограничения, которые можно превратить в дополнительные преимущества. Например, класс String реализован как immutable (неизменяемый) объект. Объект класса String неизменяем с момента создания и до момента смерти. Все методы класса String (remove, substring, …), возвращают новую строку, абсолютно не изменяя объект, у которого они были вызваны.

— Ничего себе. Вот оно как, оказывается.

— Инкапсуляция очень интересная штука.

— Ага.