ТЗ на ”Специализацию Java”

Иван Игоревич Овчинников

2022-08-12 (23:23)

Оглавление

Глава 1

Java Core

1.1. Платформа: история и окружение

Краткая история (причины возникновения); JDK, JRE, JVM, выбор версии; CLI: сборка, множественные источники, пакеты, запуск; структура проекта: пакеты, классы, метод main, комментарии, документирование;

1.1.1. Задания к семинару

С-1 Скомпилировать проект из трёх классов, находящихся в двух пакетах, а также создать для этого проекта стандартную веб-страницу с документацией

1.2. Управление проектом: сборщики проектов

Управление проектом: Jar-файлы; Gradle/Maven: зависимости, выгрузка артефакта, публичные репозитории, свойства проекта, приватные репозитории (хостинг); Bazel

1.2.1. Задания к семинару

С-2 Создать загружаемый артефакт с функцией суммирования двух чисел и загрузить его в локальный кэш артефактов;

1.3. Специализация: данные и функции

Базовые функции языка: математические операторы, условия, циклы, бинарные операторы; Данные: типы, преобразование типов, константы и переменные (примитивные, ссылочные), бинарное представление, массивы (ссылочная природа массивов, индексация, манипуляция данными); Функции: параметры, возвращаемые значения, перегрузка функций;

1.3.1. Задания к семинару

С-3 Написать как можно больше вариантов функции инвертирования массива единиц и нулей за 15 минут (без ветвлений любого рода);

С-3 Сравнить без условий две даты, представленные в виде трёх чисел гггг-мм-дд;

1.4. Специализация: ООП

Инкапсуляция: Классы и объекты (внутренние классы, вложенные классы, static, private/public, final, интерфейс взаимодействия с объектом), перечисления (создание, конструкторы перечислений, объекты перечислений, дополнительные свойства); Наследование: extends, Object (глобальное наследование), protected, преобразование типов, final; Полиморфизм: override, abstract, final;

1.4.1. Сценарий урока

(несмотря на наличие базовых знаний об ООП, глубоко убеждён, что базовое вступление всё равно нужно, поэтому...) Добрый вечер, вот мы с вами и покончили со вступлениями, добравшись до самого интересного. Объектно ориентированное программирование.

Главное, что нам нужно сегодня сделать - это получить понимание. А для этого нам нужно, в первую очередь, расслабиться и не пытаться судорожно запомнить все умные слова которые мы слышим сразу, слова запомнятся сами собой, когда придёт понимание. На деле за аббревиатурой ООП не стоит ничего страшного. Все новички часто пугаются, мол, ООП то, ООП сё, я же призываю вас не думать о сложностях.

Как вы, вероятнее всего, уже слышали и знаете, основа всего языка Java это классы. На классах построен весь язык Java. Все программы, которые мы писали до этого - это тоже классы. Не очень полезные, и особо ничего не делающие, но всё же классы. Самое важное, что надо знать о классах - это то, что классы определяют новый тип данных для нашей программы.

нарисовать котика

Скорее всего, все видят на этой картинке котика? Понятно, что мой художественный талант оставляет желать лучшего, но всё же, это скорее котик, чем, скажем, дом или дерево. Что это значит? То, что у нас в голове есть некоторый шаблон, по которому мы определяем, является-ли объект котиком. Класс это как бы чертёж. Когда мы увидим настоящего котика - мы сразу поймём это, будь он рыжий чёрный или цветной, и какой-бы породы он ни был - мы понимаем, что это кот.

В данной ситуации, кот - это класс. Для объявления класса в Java внезапно используется ключевое слово `class` и далее пишется название класса с большой буквы. В общем случае, когда мы пишем большую программу, в которой используем большое количество классов - они все раскидываются по отдельным файлам, и не должны создавать из кода малочитаемую простыню символов. **Важно, что название публичного класса в файле должно полностью совпадать с названием самого файла, здесь же стоит напомнить о том, что Java - это регистрозависимый язык, то есть заглавные буквы с точки зрения компилятора отличаются от строчных**. Классы обычно представляют модели объектов из реального мира: машина, книга, котик; или некоторые программные абстракции: строка текста, поток ввода, соединение с сервером.

Создадим наш первый класс, котика. У котика, как минимум, должно быть имя, окрас и возраст. Раньше мы работали с уже определёнными типами данных, такими, как инты, байты, стринги и массивы. Теперь у нас есть свой собственный тип данных - кот. Он, как и массив, является ссылочным, но может хранить в себе вот такие разнородные данные, а также содержать методы, но об этом чуточку позже. Как и в случае с массивами, объекты классов надо создавать. Иногда они также называются экземплярами, котик номер один, котик номер два и так далее. Много котиков всё ещё остаются котиками, но каждый из них - отдельный экземпляр. Экземпляр класса - это и есть объект из той самой аббревиатуры “ООП”. Создание экземпляра класса в общем виде выглядит так

```
Cat c1 = new Cat();
```

После выполнения этой инструкции, c1 станет экземпляром класса Cat. Ничего не напоминает? Да именно так мы создавали массивы. Только теперь мы создали не набор однотипных переменных, а набор тех переменных, которые посчитали нужными. Удобно, не правда-ли? По описанному нами шаблону мы теперь можем создать любое количество объектов, экземпляров этого класса, и все они будут независимы друг от друга, если мы явно не укажем обратное. Как мы помним, в нашем классе

были три переменных, значит и в нашем объекте они тоже есть. На первый взгляд, может показаться, что мы в правой части этого выражения только лишний раз продублировали название класса, но это не так. На самом дела - это вызов конструктора, но о конструкторах мы также поговорим немного позже, пока разберёмся с тем, что находится внутри экземпляра класса кота.

В переменных хранится так называемое состояние объекта. В методах-же описывается поведение объекта, то есть те вещи, которые он, объект, умеет делать. Наверняка вы помните, как на более ранних уроках вы мучительно не понимали, зачем нам писать отдельные функции и методы, если можно просто взять и написать всю нашу программу строчками подряд? Вот оно - то самое место, где они, функции и методы, очень нужны и важны. Для описания поведения наших классов. Фактически, этим мы и занимались всё время - описывали поведение нашего основного класса. Все экземпляры всех классов имеют свои адреса в памяти и занимают там отдельные области, а идентификатор хранит ссылку на адрес в памяти, где всё это находится.

Рисовать очередной арт-хаус.

С полями разобрались, поговорим о методах. Все мы знаем, что котики умеют мяукать и смешно прыгать. В целях демонстрации мы в описании этих действий просто делать разные выводы в консоль, хотя мы и можем научить нашего котика выбирать минимальное значение из массива, но это было бы, как минимум, неожиданно. Итак опишем метод, например, подать голос и прыгать, а прыгать наш котик будет только если он моложе, скажем, пяти лет. Теперь когда мы хотим позвать нашего котика, он нам скажет, «мяу, я *имя котика*», а если мы решили что котику надо прыгать, он решит, прилично-ли это - прыгать в его возрасте.

Конструкторы классов нужны для нескольких вещей, например, для того, чтобы наш класс сразу что-то сделал при создании. Например,

`public Cat () { System.out.println("hello, class"); }`. Шучу, это бесполезно.

Например, чтобы мы могли не создать пустой экземпляр и наполнять его, а чтобы сразу задать ему базовые параметры окраса и имени. Или возраста и окраса, если сделать перегрузку конструктора. Все помнят про перегрузку методов? Тут ситуация точно такая же, в зависимости от типов передаваемых параметров, компилятор выберет нужный конструктор.

Обратите внимание, что мы описываем конструкторы и вызываем именно их в правой части выражения, создающего новые экземпляры котиков. Но, что за конструктор мы видели в самом начале, когда ни о каких конструкторах ещё совсем ничего не знали? Это называется, конструктором по-умолчанию, его создаёт компилятор, если не увидит в описании класса никаких конструкторов. То есть, рассуждения компилятора можно представить себе примерно следующим образом:

— так, этот программист ничего не знает о конструкторах и о том что у языка полностью объектно-ориентированная природа, добавлю для него пустой конструктор, пусть не думает о моём внутреннем устройстве;
— о, программист написал какой-то конструктор, значит он знает о том, что это такое и зачем нужно, не буду ему помогать, ему виднее.

Из чего мы можем сделать простой вывод: как только мы добавили любой конструктор в наш класс, конструктор по-умолчанию создаваться не будет, а значит, если он нам нужен, придётся его добавить вручную. Зачем может понадобиться конструктор по-умолчанию, поговорим далее.

1.5. Специализация: Тонкости работы

Файловая система и представление данных; Пакеты `java.io`, `java.nio`, `String`, `StringBuilder`, string pool, ?JSON/XML?

Глава 2

Java Development Kit

2.1. Исключения

Механизм и понятие, введение в **многопоточность**, throw; Наследование и полиморфизм в исключениях; Способы обработки исключений (try/catch, throws, finally); try-with-resources, штатные и нештатные ситуации

2.2. Интерфейсы

Понятие и принцип работы, implements; Наследование и множественное наследование интерфейсов; Значения по-умолчанию

2.3. Обобщённое программирование

2.4. Коллекции

List, Set; Хэш-код, интерфейс Comparable; Map, Object (Использование методов, Переопределение методов); Итераторы, **Многопоточные** коллекции

2.5. Многопоточность

Понятие, Принцип (реальная и псевдопараллельность); Runnable, Thread (Свойства, Особенности создания); Deadlock, Состояние гонки, Object (Ожидание/уведомление); Синхронизация (Синхронизация по монитору, Частичная, по классу)

2.6. Графический интерфейс пользователя

GUI (Swing): Окна и свойства окон, **Многопоточность** и абстргирование асинхронных вызовов; менеджеры размещений и проведение параллелей с веб-фреймворками, разделение на фронт-энд и бэк-энд; JPanel и рисование, Обработка действий пользователя

Глава 3

Java Junior

3.1. JDBC

3.2. Сетевое программирование

Socket, ServerSocket, Многопоточность, абстрагирование сетевого взаимодействия, интерфейсы

3.3. Введение в архитектуры приложений на Java

клиент-серверы, веб-приложения, сервлеты, толстые и тонкие клиенты, выделение бизнес-логики и хранения, создание общих пространств и модульность проекта

3.4. Сериализация, StreamAPI, ReflectionAPI

3.5. Введение в Java EE

3.6. Введение в Spring framework

Глава 4

Java Junior+