Защита второй лабы

1 Объектно-ориентированное программирование. Основные понятия: объекты, наследование, полиморфизм, инкапсуляция

Java — Это полностью объектно-ориентированный язык.

Объектно ориентированная программа состоит из объектов, каждый из которых облдает определенными функциональными возможностями, предоставляемыми в распоряжение пользователей, а также скрытой реализацией.

Подход ООП подразумевает построение Java-программы на основе взаимодействии объектов, которые базируются на классах.

Объекты можно брать либо в готовом виде из библиотек, либо проектировать самостоятельно. Но покуда можно расширить уже готовый объект до того класса который нам нужен, мы будем использовать готовые объекты.

Традиционное структурное программирование заключается в разработке алгоритмов(набора процедур) и для данного алгоритма потом ищется подходящий способ хранения данных.

Поход ООП ставит на первое место данные, а на второе место процедуры обработки этих данных.

Подход ООП облегчает поиск ошибок в программе тк легче найти ошибку в 100 классах заключающих по 20 методов чем искать ошибку в коде содержащем 2000 процедур.

Классы

Наиболее важным понятием ООП является **класс** — шаблон по которому создается объект.

Конструирование объекта на основе некоторого класса называется **получением экземпляра этого класса**.

Объекты

В ООП определены следующие ключевые свойства объектов:

- Поведение объекта — какие методы к нему можно применять.
- Состояние объекта — как этот объект реагирует на применение методов.

  Состояние объекта не полностью описывает его, поскольку каждый объект имеет свою собственную идентичность
- Идентичность объекта — чем этот объект отличается от других характеризуюхихся таким же поведением и состоянием.

Основные свойства объектов могут окахывать взимное влияние.

Наследование

Данный принцип гарантирует то, что один класс можно построить на основе других классов. В этом случае говорят что новый класс расширяет тот класс, на основе которого он создан. Язык Java создан на основе "Глобального суперкласса"Object. Все остальные классы расширяю его.

Java не поддерживается множественное наследование — каждый тип может иметь одного родителя (суперкласс) и неограниченное количество наследников (подклассов).

Расширяющий класс содержит все свойства и методы расширяемого класса + в него добавляются новые методы и поля данных.

Расширение класса и получение на его основе нового называется **наследованием**.

Полиморфизм

Полиморфизм – это способность программы идентично использовать объекты с одинаковым интерфейсом без информации о конкретном типе этого объекта.

Вопрос, что дает нам использование полиморфизма?

- Позволяет подменять реализации объектов. На этом основано тестирование.
- Обеспечивает расширяемость программы — становится гораздо легче создавать задел на будущее. Добавление новых типов на основе существующих — наиболее частый способ расширения функциональности программ, написанных в ООП стиле.
- Позволяет объединять объекты с общим типом или поведением в одну коллекцию или массив и управлять ими единообразно (как в наших примерах, заставляя всех танцевать – метод dance или плыть – метод swim).
- Гибкость при создании новых типов: вы можете выбирать реализацию метода из родителя или переопределить его в потомке.

Инкапсуляция(иногда называется сокрытием информации)

Данные в объекте называются **полями экземпляра**, а функции и процедуры, выполняющие операции над данными — его **методами**.

В конкретном объекте поля экземпляра имеют **определенные значения**. Множество этих значений называется **текущим состоянием объекта**. Вызов любого метода для объекта **может изменить его состояние**.

Основной принцип инкапсуляции зыключается в **запрещении прямого доступа к полям экземпляра данного класса из других классов.** (Программы должны взаимодействовать с данными объекта только через методы этого объекта.)

Этот принцип имеет рещающее значение для повторного их использования и надежности работы программ.

Также это означает что в классе **можно полностью изменить способ хранения данных**.

2 Понятие класса. Классы и объекты в Java.

Отношение между классами:

1. Зависимость(Один класс зависит от другого класса)
2. Агрегирование(Объект класса A содержит объекты класса B)
3. Наследование(Отношение между конкретным и более общим объектом)

3 Члены класса. Модификаторы доступа.

Для создания новых экземпляров класса в Java служат конструкторы — специальный методы, предназначенные для создания и инициализации экземляра класса.

Имя конструктора всегда совпадает с именем класса.

Чтобы создать объект конструктор нужно объеденить с операцией new.

Есть объектная переменная которая просто ссылается на объекты типа класса, а есть объекты содержат объекты.

Модификаторы доступа — это чаще всего ключевые слова, которые регулируют уровень доступа к разным частям твоего кода. Почему «чаще всего»? Потому что один из них установлен по умолчанию и не обозначается ключевым словом.

- **Private** — наиболее строгий модификатор доступа. Он ограничивает видимость данных и методов пределами одного класса.

  Собственно, **ограничение доступа к полям и реализация геттеров-сеттеров** — самый распространенный пример использования private в реальной работе.

  То есть реализация инкапсуляции в программе — **главное предназначение этого модификатора.**

- **protected**

  Поля и методы, обозначенные модификатором доступа protected, будут видны:

  - в пределах всех классов, находящихся в том же пакете, что и наш
  - в пределах всех классов-наследников нашего класса.

- package visible(default)

  Он не обозначается ключевым словом, поскольку установлен в Java по умолчанию для всех полей и методов.

  Чаще всего default-доступ используется в пакете, где есть какие-то классы-утилиты, не реализующие функциональность всех остальных классов в этом пакете.

- public

  Ведь public создан для того, чтобы отдавать что-то пользователям.

  Виден всем.

4 Создание и инициализация объектов. Вызов методов.

Cat cat = new Cat();

1. Сначала для хранения объекта выделяется память.
2. Далее Java-машина создает ссылку на этот объект (в нашем случае ссылка — это Cat cat).
3. В завершение происходит инициализация переменных и вызов конструктора (этот процесс мы рассмотрим подробнее).

Кроме того, из лекции о жизненном цикле объекта ты наверняка помнишь, что он длится до тех пор, пока на него существует хоть одна ссылка.

Теперь перейдем к примеру:

1. проинициализируются статические переменные класса родителя.
2. После инициализации статических переменных класса-предка инициализируются статические переменные класса-потомка.
3. Инициализация переменных продолжается. Третьими по счету будут инициализированы нестатические переменные класса-предка
4. Наконец, дело дошло до конструкторов! Точнее, до конструктора базового класса. Начало его работы — четвертый пункт в процессе создания объекта.
5. Теперь пришла очередь инициализации нестатических полей класса-потомка
6. Вызывается конструктор дочернего класса

Метод — это именованный блок кода, объявляемый внутри класса. Он содержит некоторую законченную последовательность действий (инструкций), направленных на решение отдельной задачи, который можно многократно использовать.

- Методы с возвращаемым значением
- void

  Не у всех методов есть возвращаемое значение. Некоторым или нечего, или не нужно ничего возвращать. Что же тогда делать? Тогда в сигнатуре метода на место возвращаемого значения мы пишем void.

Методы могут иметь (или не иметь) определенные данные, которые будут поступать снаружи, а именно — с места, где и был вызван метод.

В методе можно использовать неограниченное количество параметров, но больше 7 — не рекомендуется.

Перегрузка методов — это использование одного и того же имени метода несколько раз при его объявлении в классе. С точки зрения синтаксиса языка, не может быть двух одинаковых имен в некотором локальном пространстве. Но при этом допускается объявление методов с одинаковыми именами но отличающимися аргументами.

Здесь мы видим, что методы не обязаны содержать одинаковый модификатор доступа (как и возвращаемый тип).

Если вызывается перегруженный метод, то из нескольких объявленных методов компилятор автоматически определяет нужный по параметрам, которые указываются при вызове.

5 Области видимости переменных.

Для видимости используютмя модификаторы доступа:

- public

  К переменной, методу или классу, помеченному модификатором public, можно обращаться из любого места программы. Это самая высокая степень открытости — никаких ограничений нет.

- private

  К переменной или методу, помеченному модификатором private, можно обращаться только из того же класса, где он объявлен. Для всех остальных классов помеченный метод или переменная — невидимы и «как бы не существуют». Это самая высокая степень закрытости – только свой класс.

- Без модификатора

  Если переменная или метод не помечены никаким модификатором, то считается, что они помечены «модификатором по умолчанию». Переменные или методы с таким модификатором (т.е. вообще без какого-нибудь) видны всем классам пакета, в котором они объявлены. И только им. Этот модификатор еще иногда называют «package», намекая, что доступ к переменным и методам открыт для всего пакета, в котором находится их класс

6 Модификаторы final и static.

Модификаторы в Java – это ключевые слова, которые придают классу, полю класса или методу определенные свойства.

1. static

   Когда мы описали поле класса или метод как static, его можно вызвать без использования экземпляра класса. То есть вместо такой конструкции: Cat cat = new Cat(); cat.method(), можно написать просто Cat.method(). При условии, что метод объявлен как static. Статические переменные едины для всех объектов класса. У них одна ссылка.

   Еще одно важное замечание, которое нужно сказать по поводу static модификаторов: статические поля инициализируются во время загрузки класса.

2. final

   Применяя final модификатор Вы говорите, что поля не могут быть изменены, методы переопределены, а классы нельзя наследовать (о наследовании будет отдельная статья). Этот модификатор применяется только к классам, методам и переменным (также и к локальным переменным).

7 Пакеты, инструкция import.

Файлы в компьютере группируются по папкам. Классы в Java (а каждый класс лежит в отдельном файле) группируются по пакетам, которые являются папками на диске. Ничего принципиально нового. Но есть два замечания

Запомните, что при импорте пакета классам, не производным от классов из данного пакета в импортирующем коде, будут доступны только те элементы пакета, которые объявлены как public.

Организация классов в виде пакетов позволяет избежать конфликта имен между классами.

1. «Полным уникальным именем класса» является «имя пакета» + «имя класса».

   Полное имя класса всегда уникально!

   Каждый раз писать длинное имя, например java.util.ArrayList, очень неудобно. Поэтому в Java добавили возможность «импортировать классы». В своем коде ты можешь пользоваться коротким именем других классов, но ты должен в начале своего класса явно указать, какой именно класс будет использоваться. Делается это конструкцией вида «import java.util.ArrayList;»

2. Лучше всегда класть классы в пакеты, а не в корень папки src.

   Когда классов мало, это ещё не представляет проблему, но когда классов много – очень легко запутаться. Поэтому всегда создавай классы только в пакетах.