Android разработка. Подготовительный курс

Лекция 4. Основы ООП. Наследование, инкапсуляция, полиморфизм, абстракция

Урок 4. ООП – объектно-ориентированное программирование

- Методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определённого класса, а классы образуют иерархию наследования.(Wikipedia)
- Образ мышления при котором мы представляем то что есть в коде как объекты реального мира. Проще будет вспомнить то что было на прошлом уроке, а именно классы Человек и Машина. Это объекты реального мира, части которых мы переносим в программы с требующимся нам функционалом.

Урок 4. Основные принципы ООП

- Абстракция — отделение концепции от ее экземпляра;
- Полиморфизм — реализация задач одной и той же идеи разными способами;
- Наследование — способность объекта или класса базироваться на другом объекте или классе. Это главный механизм для повторного использования кода. Наследственное отношение классов четко определяет их иерархию;
- Инкапсуляция — размещение одного объекта или класса внутри другого для разграничения доступа к ним.

Урок 4. На чем будем учиться

- Представляю вам класс Транспортное средство(Vehicle):

```
public abstract class Vehicle{
  protected int speed;
  public Vehicle(int speed){
        this.speed = speed;
  }

  public abstract int ride(int km);
}
```

Урок 4. Абстракция

- Это способ представления объектов реального мира в коде. Возьмем в пример машину. Машина это четырех колесное транспортное средство которое едет. Один из этих параметров мы описали у себя в программе когда сделал класс машина. Таким образом мы представили объект реального мира в виде объекта в программе.

Урок 4. Наследование

- Классы могут наследовать друг друга. Что означает термин «наследовать»? Он означает перенимать свойства своего родителя. Кто такой родитель? Это тот от кого наследуются.
  Представим реальный мир. Наследники у нас это обычно дети. Они наследуют какие-то черты своих родителей. В программировании то же самое, но это могут быть не только люди. Любой класс может быть наследником и может наследовать. Например наш класс Водитель мог бы наследовать класс Человек, который был описан в другом задании. Ведь у Водителя есть и возраст и рост и имя. Однако не все это нам было интересно в нашем втором задании.

Урок 4. Наследование, пример

- А теперь возьмем и унаследуем нашу машину(Car) от транспортного средства(Vehicle):

```
public class Car extends Vehicle{

  public Car(int speed){
        super(speed);
  }

  @Override
  public int ride(int km){
        return speed*km;
  }

}
```

Урок 4. Наследование немного разъяснений слов

- extends – ключевое слово которое означает «Наследуется от». После него пишут класс от которого наследуется данный. В Java отсутствует множественное наследование по-этому наследоваться можно только от одного класса
- abstract – означает абстрактный. Мы как будто говорим что он будет в дальнейшем и будет называться так но сейчас его нет. Необходимо ОБЯЗАТЕЛЬНО переопределить его у наследников
- @Override – говорит о том что метод переопределяет метод родителя, помеченный словом abstract.

Урок 4. Инкапсуляция

- Помимо областей видимости в виде фигурных скобок есть также области видимости между классами. Например некоторые из вас могли сделать доступ к переменным напрямую в задании поменять имя а не через метод. Вот и получаем что любой разработчик может поменять как угодно переменную, ведь обычно на одном проекте работает больше одного человека. Вы как тот кто изначально писал проект не хотите чтобы переменная менялась как попало, а только чтобы по описанным вам правилам. В этом вам поможет инкапсуляция.

Урок 4. Инкапсуляция

- Вспомним волшебное слово public. Оно могло стоять перед классом, перед методом, перед переменной. Оно означает область видимости внутри других классов. В Java существует 4 главных ключевых слова:
- public – публичный. Видно везде. Нужно ставить если хотим разрешить доступ из вне
- package-private – стоит если ничего не написать. Видно внутри одного пакета.
- protected – защищенный. Защищает видимость при наследовании, а именно: защищенные видно наследникам но не видно из вне.
- private – приватный. Видно только внутри этого класса.

Урок 4. Инкапсуляция пример

- Мы можем заметить ключевые слова public и protected внутри классов Vehicle и Car. Как мы видим и то и то мы можем использовать внутри друг друга. Добавим переменную типа private в Vehicle. И убедимся что нельзя использовать её из класса Car.
- ```
  public abstract class Vehicle{
    protected int speed;
    private String name;
    public Vehicle(int speed){
          this.speed = speed;
    }

    public abstract int ride(int km);
  }
  ```
  ```

Урок 4. Полиморфизм

- Полиморфизм это когда мы можем использовать наследников как их родителей. Так как все наследники наследуют методы и поля родителей то можно использовать их. В нашем случае можно объявить переменную Vehicle и поместить в неё экземпляр класса Car. Но для наглядности давайте создадим еще одного наследника класса Vehicle. Представляю вам Велосипед!
- Public class Bicycle extends Vehicle{

```
public Bicycle(int speed){
      super(speed);
}

@Override
public int ride(int km){
      return km;
}

}
```

Урок 4. Полиморфизм практика

- Теперь создадим 2 экземпляра класса Car и Bicycle соответственно и запихнем их в переменную типа Vehicle. Результат скажет все за себя.
- public class Main {

```
public static void main(String[] args) {
        runVehicle(new Car(14));//1400
        runVehicle(new Bicycle(14));//100
    }
    public static void runVehicle(Vehicle vehicle){
        int time = vehicle.run(100);
        System.out.println(time);
    }
}
```

Урок 4. Класс + дз

- Повторить мой успех с Транспортным средством и Машиной и Велосипедом.
- Добавить поле бензобака внутри Vehicle и переопределить метод заправки внутри наследников Vehicle. Создать абстрактный метод для заправки. Машина заправляется на +n, велосипед не надо заправлять
- Создать класс Самолет который будет по-своему реализовывать заправку и полет. Например: заправляется на n*10

Урок 4. Класс + дз. Вам не выжить

- Добавить всем водителя из прошлого урока. Точно так же расчитывать скорость(скорость * мастерство водителя).
- Унаследовать водителя от класса Человек(у человека должно быть имя).
- Дополнить класс Человек полем вес.
- Добавить класс Грузовик в который можно сажать Пассажиров. Пассажиры унаследовать от Человека. Грузовик – от Транспортного средства. Посадить 1 пассажира и поставить формулу вычисления скорости как скорость-вес пассажира. Устроить гонки.

Урок 4. Класс + дз. Немного алгоритмов

- День программиста отмечается в 255-й день года (при этом 1 января считается нулевым днем). Требуется написать программу, которая определит дату (месяц и число григорианского календаря), на которую приходится День программиста в заданном году.
- В григорианском календаре високосным является:

  год, номер которого делится нацело на 400

  год, номер которого делится на 4, но не делится на 100
- Входные данные

  В единственной строке входного файла INPUT.TXT записано целое число от 1 до 9999 включительно, которое обозначает номер года нашей эры.
- Выходные данные

В единственную строку выходного файла OUTPUT.TXT нужно вывести дату Дня программиста в формате DD/MM/YYYY, где DD — число, MM — номер месяца (01 — январь, 02 — февраль, ..., 12 — декабрь), YYYY — год в десятичной записи.

Куда отправлять ДЗ?

db@bigdig.com.ua
Обязательно подписывайте.

Файлики *.java запихиваете в архив и присылаете по почте.

Файлики лежат по пути место_хранения_проекта\untitled1\src\com\company\
(где untitled1 – название проекта, com\company – название пакета)

Ссылка на доп инфо - https://javarush.ru/groups/posts/1880--principih-oop