IT ШКОЛА SAMSUNG

Объектно ориентированное проектирование на примерах

Яценко Дмитрий

SAMSUNG

Классы и объекты

Класс в Java - это абстрактный тип данных, для создания объектов. Класс определяет состоит из полей (переменных) и методов (функций).

В процессе выполнения Java-программы система использует определения классов для создания экземпляров классов или объектов.

Для того чтобы создать новую программу необходимо произвести проектирование. Проектирование программ в объектно ориентированной парадигме отличается от классической.

Задание 1. Дробь.

Прежде, чем рассмотреть процесс проектирования, рассмотрим пример. Ввести дроби A и B и вычислить и вывести на экран в виде правильной дроби. Решить без применения ООП и с применением ООП.

$$\left(\frac{An}{Ad}+3\right):\left(\frac{Bn}{Bd}-\frac{1}{3}\right)$$

1.1 - ввод

При решении задачи выполняем шаг за шагом последовательно, как изложено в задаче. Вводим дробь, делаем вычисления, выводим результат.

```
Scanner sc = new Scanner(System.in);
System.out.println("Введите первую дробь");
int An = sc.nextInt();
int Ad = sc.nextInt();
System.out.println("Введите вторую дробь");
int Bn = sc.nextInt();
int Bd = sc.nextInt();
```

1.1 - вычисление

```
// считаем первую скобку (приводим к общему знаменателю
// и складываем числитель)
An = An + 3 * Ad;
// считаем вторую скобку (приводим к общему
// знаменателю и складываем числитель)
Bn = 3 * Bn - Bd;
Bd = 3 * Bd;
// считаем деление скобок
An = An * Bd;
Ad = Ad * Bn;
```

1.1 - вывод

```
System.out.println("Результат:");
// печатаем в десятичном виде
System.out.println(1.0 * An / Ad);
if (An / Ad == 0) {
    // печатаем в обычном виде
    System.out.println(An);
    System.out.println("---");
    System.out.println(Ad);
} else {
    // печатаем правильную дробь
    System.out.println("   " + An % Ad);
    System.out.println(An / Ad + "-----");
    System.out.println("   " + Ad);
}
```

1.1 - результат

Результат работы программы:

Введите первую дробь
1
4
Введите вторую дробь
3
5
Результат:
12.1875
3
12-----
16

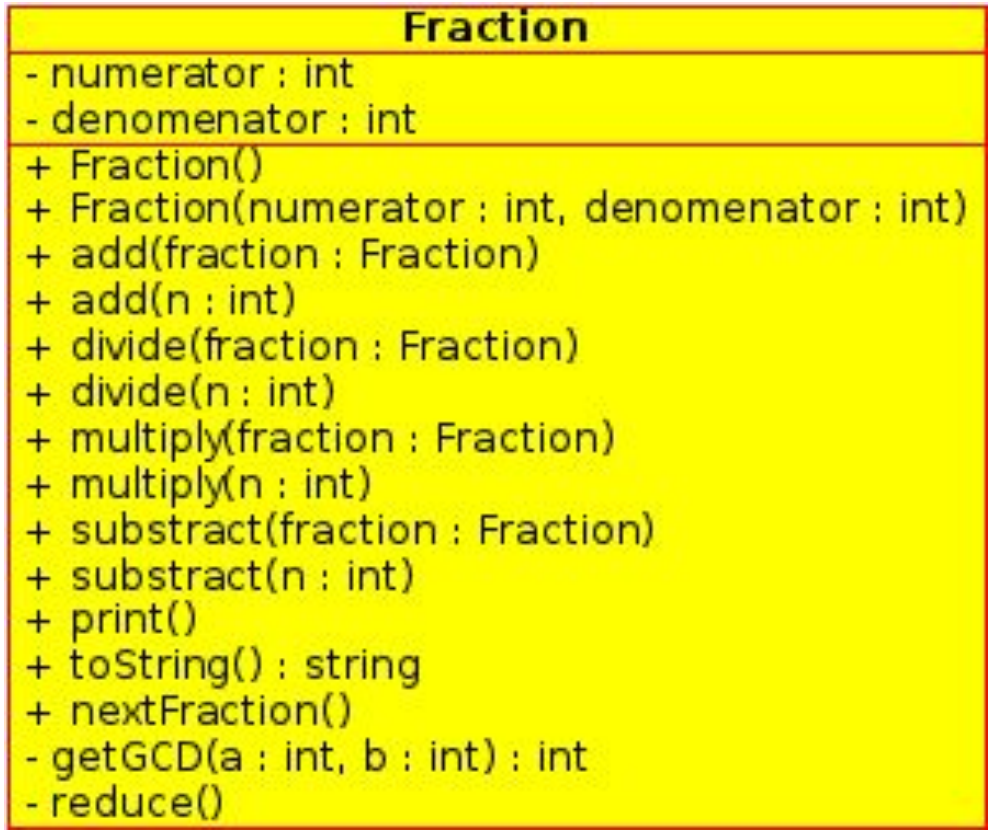
1.2 теперь с ООП

Теперь решим эту же задачу с применением ООП. В отличие от предыдущего примера, при разработке в ООП парадигме мы сначала спроектируем класс так, чтобы в полях хранился числитель и знаменатель, чтобы были методы для работы с дробью – сложить, вычесть, умножить, делить, вывести на экран, чтобы был конструктор для удобного создания дроби. Кроме того, перегрузим эти методы, чтобы мы могли вызывать их для аргументов разных типов.

И только после того как мы получим этот класс, мы займёмся решением непосредственно самой задачи.

1.2 теперь с ООП

На предыдущем слайде мы целую страницу посвятили описанию того как будет устроен наш класс. Есть способ проще и понятнее - UML:



1.2 – класс, поля и методы

```
public class Fraction {
        private int numerator;
        private int denomenator=1;

        public void add(Fraction fraction) {
                numerator = numerator * fraction.denomenator +
fraction.numerator * denomenator;
                denomenator = denomenator * fraction.denomenator;
                reduce();
        }
        public void add(int n) {
                add(new Fraction(n, 1));
        }

        public void substract(Fraction fraction) {
                numerator = numerator * fraction.denomenator -
fraction.numerator * denomenator;
                denomenator = denomenator * fraction.denomenator;
                reduce();
        }
        public void substract(int n) {
                substract(new Fraction(n, 1));
        }
```

```
public void multiply(Fraction fraction) {
                numerator = numerator * fraction.numerator;
                denomenator = denomenator * fraction.denomenator;
                reduce();
        }

        public void multiply(int n) {
                multiply(new Fraction(n, 1));
        }

        public void divide(Fraction fraction) {
                if (fraction.numerator == 0) {
                        System.out.println("На эту дробь делить нельзя!");
                        return;
                }
                multiply(new Fraction(fraction.denomenator,
fraction.numerator));
        }

        public void divide(int n) {
                divide(new Fraction(n, 1));
        }
```

1.2 – класс, конструкторы

```
public void nextFraction() {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int d = sc.nextInt();
        if (d == 0) {
            System.out.println("Знаменатель не может быть
нулевым!Повторите ввод:");
            return;
        }
        numerator=n;
        denomenator=d;
        reduce();
    }

    Fraction(){
    }

    Fraction(int numerator, int denomenator) {
        if (denomenator == 0) {
            System.out.println("Знаменатель не может быть
нулевым!");
            return;
        }
        this.numerator = numerator;
        this.denomenator = denomenator;
        reduce();
    }
```

```
public String toString(){
        return (numerator*denomenator<0?"-":" ")
        +Math.abs(numerator)
    +"/"+Math.abs(denomenator);
    }

    private int getGCD(int a, int b) {
        return b==0 ? a : getGCD(b, a%b);
    }

    private void reduce(){
        int t=getGCD(numerator,denomenator);
        numerator/=t;
        denomenator/=t;
    }
```

1.2 – класс, метод печати

```
    public void print() {
        if(numerator % denomenator == 0){
            System.out.println(numerator/denomenator);
            return;
        }
        if (numerator / denomenator == 0) {
            System.out.println("    " + Math.abs(numerator));
            System.out.println((numerator*denomenator<0?"-":" ")+" ---- или "+1.0 *
                numerator / denomenator);
            System.out.println("    " + Math.abs(denomenator));
        } else {
            System.out.println("    " + Math.abs(numerator % denomenator));
            System.out.println((numerator*denomenator<0?"-":" ")+numerator / denomenator +
                    "---- или "+1.0 * numerator / denomenator);
            System.out.println("    " + Math.abs(denomenator));
        }
    }

}
```

1.2 – вычисление

Мы строили, строили и наконец построили. Класс готов и теперь мы можем наконец приступить к вычислениям задачи:

```
public class test {
    public static void main(String[] args) {
        Fraction A = new Fraction();
        Fraction B = new Fraction();
        A.nextFraction();
        B.nextFraction();
        A.add(3);
        B.substract(new Fraction(1,3));
        A.divide(B);
        A.print();
        System.out.println(A);
    }
}
```

1.2 - результат

Запустив test.java и вбив те-же значения, получаем такой же результат:

```
Введите дробь
1
4
Введите дробь
3
5
================
12.1875
  3
12----
  16
================
```

Задача 1 – анализ.

Мы решили задачу используя два разных подхода. Сравним (при одинаковом результате):

- В первой программе порядка 30 строк. Процесс разработки прост и понятен
- Во второй программе порядка 100 строк. Процесс разработки более громоздкий. Результат стал доступен только в конце.

А теперь вопрос, если результат одинаков, то зачем платить больше? Казалось бы первый вариант лучше? **Однако**...

Задача 2.

Ввести 5 правильных дробей и вычислить сумму и произведение дробей. Решить без применения ООП и с применением ООП.

$$\sum_{i=0}^{10} \frac{An_i}{Ad_i} \quad \text{и} \quad \prod_{i=0}^{10} \frac{An_i}{Ad_i}$$

Задача 2. 2.1 - ввод

Решение без ООП – решаем аналогично решению выше. То есть мы удаляем все что мы делали в первой задаче и решаем заново, последовательно шаг за шагом, четко следуя заданию.

```
Scanner sc = new Scanner(System.in);
System.out.println("Введите дроби:");
int n[] = new int[5];
int d[] = new int[5];
for (int i = 0; i < n.length; i++) {
    System.out.println("===============");
    n[i] = sc.nextInt();
    d[i] = sc.nextInt();
}
```

2.1 - вычисление

```
int rez1n = 0;
int rez1d = 1;
int rez2n = 1;
int rez2d = 1;
for (int i = 0; i < n.length; i++) {
      rez1n = rez1n * d[i] + rez1d * n[i];
      rez1d = rez1d * d[i];
      rez2n = rez2n * n[i];
      rez2d = rez2d * d[i];
}
```

2.1 – печать

```
System.out.println("Результат 1:");
// печатаем в десятичном виде
System.out.println(1.0 * rez1n / rez1d);
if (rez1n / rez1d == 0) {
      // печатаем в обычном виде
      System.out.println(rez1n);
      System.out.println("---");
      System.out.println(rez1d);
} else {
      // печатаем правильную дробь
      System.out.println("   "+rez1n%rez1d);
      System.out.println(rez1n/rez1d+"-----");
      System.out.println("   "+rez1d);
}
```

2.1 – печать

```
System.out.println("Результат 2:");
// печатаем в десятичном виде
System.out.println(1.0*rez2n/rez2d);
if (rez2n / rez2d == 0) {
      // печатаем в обычном виде
      System.out.println(rez2n);
      System.out.println("---");
      System.out.println(rez2d);
} else {
      // печатаем правильную дробь
      System.out.println("   "+rez2n%rez2d);
      System.out.println(rez2n/rez2d+"-----");
      System.out.println("   " + rez2d);
}
```

2.1 – результат работы

```
Введите дроби:
==============
1
2
==============
1
2
==============
1
2
==============
1
2
==============
1
2
Результат 1:
2.5
   16
2-----
   32
Результат 2:
0.03125
1
---
32
```

2.2 – решение с ООП

В отличие от решения без ООП, у нас уже имеется класс дроби и мы его просто используем. Поэтому решение выглядит так:

```
public static void main(String[] args) {
    Fraction A[] = new Fraction[5];
    for (int i = 0; i < A.length; i++) {
        A[i] = new Fraction();
        A[i].nextFraction();
    }
    Fraction rez1 = new Fraction(0, 1);
    Fraction rez2 = new Fraction(1, 1);
    for (int i = 0; i < A.length; i++) {
        rez1.add(A[i]);
        rez2.multiply(A[i]);
    }
    System.out.println("Сумма");
    rez1.print();
    System.out.println("Произведение");
    rez2.print();
}
```

Задача 2 - вывод

Давайте опять сравним решение в двух парадигмах.
В варианте без ООП мы написали ~45 строк кода, и кстати опять думали над реализацией операций.

- В варанте ООП размер программы ~20 строк кода. Причем мы теперь не задумывались над тем как именно делаются каждые операции.

Вывод из примеров: По мере усложнения задач, ООП парадигма дает гораздо лучшие результаты в проектировании!

ООП проектирование. Игра.

Задание.

Написать игру текстовый квест «Корпорация». Цель игрока — поднять социальный статус персонажа.

Описание.

В процессе игры игроку рассказывают интерактивную историю. История начинается с первой сцены. Игрок выбирает вариант развития событий из ограниченного набора 2-3шт. Выбранная сцена становиться текущей и у нее также есть варианты развития событий. История заканчивается когда у текущей ситуации нет вариантов развития событий. При выборе варианта у персонажа меняются характеристики: карьерное положение (K), активы(A), репутация(P).

Сюжет.

Пример сюжета.

К=1,А=100тр,Р=50%

1я Сцена - «Первый клиент». Вы устроились в корпорацию менеджером по продажам программного обеспечения. Вы нашли клиента и продаете ему партию МС Windows. Ему достаточно было взять версию “HOME” 100 коробок .

- вы выпишете ему счет на 120 коробок версии “ULTIMATE” по 50тр (К+0,А-10,Р-10)
- вы выпишете ему счет на 100 коробок версии “PRO” по 10тр (К+1,А+100, Р+0)
- вы выпишете ему счет на 100 коробок версии “HOME” по 5тр (К +0, А +50, Р +1)

Решение “в лоб”.

```
Scanner in    = new Scanner(System.in);
int K=1,A=100,P=50;
System.out.println("Вы прошли собеседование и вот-вот станете сотрудниуом Корпорации. \n Осталось уладить формальности - подпись под договором (Введите ваше имя):");
String name;
name=in.next();       System.out.println("=====\nКарьера:"+K+"\tАктивы:"+A+"т.р.\tРепутация:"+P+"%\n=====");
System.out.println("Только вы начали работать и тут-же удача! Вы нашли клиента и продаете ему партию \n ПО МС Виндовс. Ему в принципе достаточно взять 100 коробок версии HOME.");
System.out.println("- (1)у клиента денег много, а у меня нет - вы выпишете ему счет на 120 коробок \n ULTIMATE по 50тр");
System.out.println("- (2)чуть дороже сделаем, кто там заметит - вы выпишете ему счет на 100 коробок \n PRO по 10тр");
System.out.println("- (3)как надо так и сделаем - вы выпишете ему счет на 100 коробок HOME по 5тр");
int a1=in.nextInt();
if(a1==1){
        K+=0;A+=-10;P+=-10;       System.out.println("=====\nКарьера:"+K+"\tАктивы:"+A+"т.р.\tРепутация:"+P+"%\n=====");
        // Следующие ситуации для этой ветки сюжета
} else if(a1==2) {
        K+=1;A+=100;P+=0;         System.out.println("=====\nКарьера:"+K+"\tАктивы:"+A+"т.р.\tРепутация:"+P+"%\n=====");
        // Следующие ситуации для этой ветки сюжета
} else {
        K+=0; A+=50; P+=1;        System.out.println("=====\nКарьера:"+K+"\tАктивы:"+A+"т.р.\tРепутация:"+P+"%\n=====");
        // Следующие ситуации для этой ветки сюжета
}
System.out.println("Конец");
```

ООП проектирование - сложно.

Объектно ориентированной проектирование, как и обычное проектирование проходит стадию анализа и синтеза. Анализ ориентирован на поиск отдельных сущностей (или классов, объектов). Синтез же воссоздаст полную модель предметной области задачи.

Анализ – это разбиение предметной области на минимальные неделимые сущности. Выделенные в процессе анализа сущности формализуются как математические модели объектов со своим внутренним состоянием и поведением.

Синтез – это соединение полученных в результате анализа моделей минимальных сущностей в единую мат-модель предметной области, с учетом взаимосвязей объектов в ней.

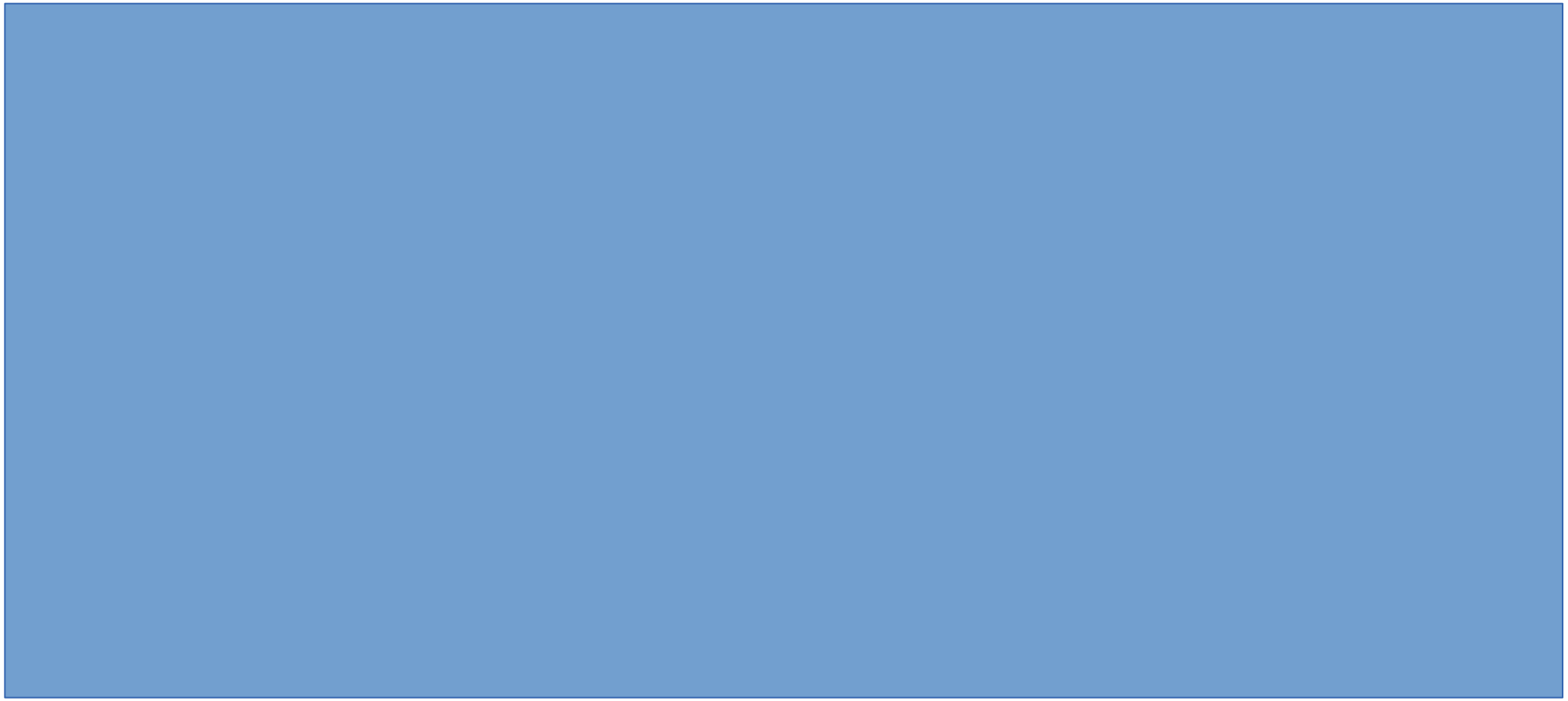
ООП проектирование - просто.

Простыми словами процесс проектирования можно описать так:

- из текста подробного описания предметной области выбираем все подлежащие - это сущности (классы),
- выбираем все глаголы (отглагольные формы, например деепричастия) – это поведение сущностей (методы классов),
- выбираем дополнения, определения, обстоятельства – они скорее всего будут определять состояние сущностей (поля классов).

Наш пример.

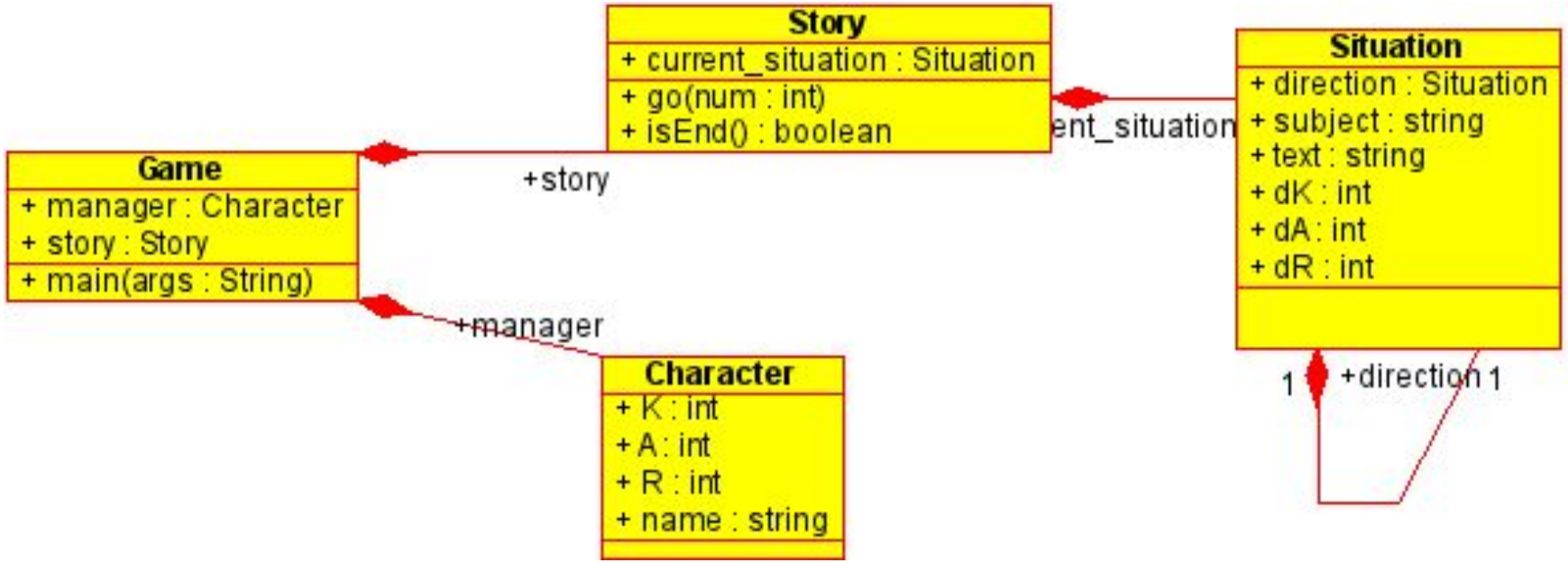
Проведя синтаксический разбор и последующий анализ описания нашей задачи получили такую таблицу:



Наш пример.

После того как мы вычленили из задания сущности, их свойства и действия — построим модели отдельных сущностей (это классы) и объединим их в общую модель, с учетом их взаимосвязей. То есть мы при описании каждой сущности должны учесть, являются ли ее поля простыми типами, или в свою очередь другими сущностями (отношение агрегирования или композиции).

Это и есть разработка приложения. Сделаем это на UML и получим диаграмму классов.


Story
+ current_situation : Situation
+ go(num : int)
+ isEnd() : boolean
Situation
+ direction : Situation
+ subject : string
+ text : string
+ dK : int
+ dA : int
+ dR : int
ent_situation
Game
+ manager : Character
+ story : Story
+ main(args : String)
+story
+manager
Character
+ K : int
+ A : int
+ R : int
+ name : string
1
+direction
1

Генерация кода.

После построения модели вы можете прямо из UML-редактора сгенерировать исходный код (только нужно выбрать язык Java). Однако редактор, зачастую дает правильный, но избыточный для нас код. Мы можем также по диаграмме написать классы сами, тем более, что они не большие.

Отдельно отмечу, что классы **Character** и **Situation** почти пусты; класс **Story** в конструкторе создает всю историю как набор ситуаций; класс **Game** имеет точку входа для консольного выражения он же наш главный метод **main**, где создается персонаж и история и происходит взаимодействие с пользователем и переключение сцен.

Генерация кода.

В случае же если приложение разрабатывалось для андроид, класс **Game** будет расширять класс **Activity**, и в методе **onCreate** (вместо **main**) будет создан объект персонажа и игры, а переключение сцен (метод **go**) должен вызываться в методах типа **onClick** этой активности.

Кстати если поле Direction сделать не обычным массивом, а ассоциативным, то написание самой истории (последовательности сцен) сильно упростится, так как вы будете видеть шаги истории не по номерам индексов в массивах вариантов, а по значащим именам сцен.

Character.java

```
package quest1;

import java.util.*;

public class Character {
	public int K;
	public int A;
	public int R;
	public String name;

	public Character(String name) {
		K = 1;
		A = 100;
		R = 50;
		this.name = name;
	}

}
```

Situation.java

```
package quest1;
public class Situation {

  public Situation[] direction;
  public String subject;
  public String text;
  public int dK;
  public int dA;
  public int dR;

  public Situation (String subject, String text, int variants, int dk,int da,int dr) {
        this.subject=subject;
        this.text=text;
        dK=dk;
        dA=da;
        dR=dr;
        direction=new Situation[variants];
   }

}
```

Story.java

```
package quest1;
public class Story {
	private Situation start_story;
	public Situation current_situation;
	Story() {
		start_story = new Situation(
				"первая сделка (Windows)",
				"Только вы начали работать и тут-же удача! Вы нашли клиента и продаете ему "
					+ "партию ПО МС Виндовс. Ему в принципе достаточно взять 100 коробок версии HOME.\n"
					+ "(1)у клиента денег много, а у меня нет - вы выпишете ему счет на 120 коробок ULTIMATE по 50тр\n"
					+ "(2)чуть дороже сделаем, кто там заметит - вы выпишете ему счет на 100 коробок PRO по 10тр\n"
					+ "(3)как надо так и сделаем - вы выпишете ему счет на 100 коробок HOME по 5тр ",

				3, 0, 0, 0);
		start_story.direction[0]=new Situation("корпоратив", "Неудачное начало, ну чтож, сегодня в конторе копроратив! "
				+ "Познакомлюсь с коллегами, людей так сказать посмотрю, себя покажу", 0, 0, -10, -10);
		start_story.direction[1]=new Situation("совещание, босс доволен", "Сегодня будет совещание, меня неожиданно вызвали,"
				+ "есть сведения что \n босс доволен сегодняшней сделкой.", 0, 1, 100, 0);
		start_story.direction[2]=new Situation("мой первый лояльный клиент", "Мой первый клиент доворлен скоростью и качеством "
				+ "моей работы. Сейчас мне звонил лично \ндиреткор компании и сообщил что скоро состоится еще более крупная сделка"
				+ " и он хотел чтобы по нейт работал именно я!", 0, 0, 50, 1);
		current_situation=start_story;
	}
	public void go(int num) {
		if(num<=current_situation.direction.length)
		current_situation=current_situation.direction[num-1];
		else System.out.println("Вы можете выьирать из "+current_situation.direction.length+" вариантов");
	}
	public boolean isEnd(){
		return current_situation.direction.length==0;
	}
}
```

Game.java

```
package quest1;
import java.util.*;
public class Game {
  public static Character manager;
  public static Story story;
  public static void main(String[] args) {
        Scanner in=new Scanner(System.in);
        System.out.println("... подпись под договором (ваше имя):");
        manager =new Character(in.next());
        story = new Story();
        while (true){
              manager.A+=story.current_situation.dA;
              manager.K+=story.current_situation.dK;
              manager.R+=story.current_situation.dR;
System.out.println("=====\nКарьера:"+manager.K+"\tАктивы:"+manager.A+"\tРепутация:"+manager.R+"\n
=====");
System.out.println("=============="+story.current_situation.subject+"==============");
              System.out.println(story.current_situation.text);
              if(story.isEnd()) {System.out.println("====================the
end!====================");return;}
              story.go(in.nextInt());
        }
}
}
```

Пример 3 – электронный журнал.

В предыдущем примере мы произвели проектирование игры-квеста. При этом мы заметили очевидное преимущество ОО подхода. Однако мы не задействовали всего арсенала ООП. По сути мы использовали лишь инкапсуляцию.

Давайте теперь рассмотрим пример в котором мы сможем задействовать все преимущества ОО. Итак ТЗ – разработать электронный журнал для школы. Разработку будем проводить только в ОО парадигме.

Пример 3.

Детализируем постановку задачи. Итак есть школа как учебное заведение, находящаяся по адресу 344000, г. Ростов-на-Дону, …., в которой происходит обучение детей по стандартной программе среднего образования(11 классов). В школе работают учителя, которые преподают по несколько дициплин, причем некоторые имеют дополнительную нагрузку в виде классного руководства либо факультативных предметов, кружков. Помимо преподавателей, в школе есть прочий персонал, директор, завуч, охранники, повара, уборщики. Вход в школу осуществляется при предъявлении магнитной карты с уникальным ID. Есть множество документов, регламентирующих процесс образования.

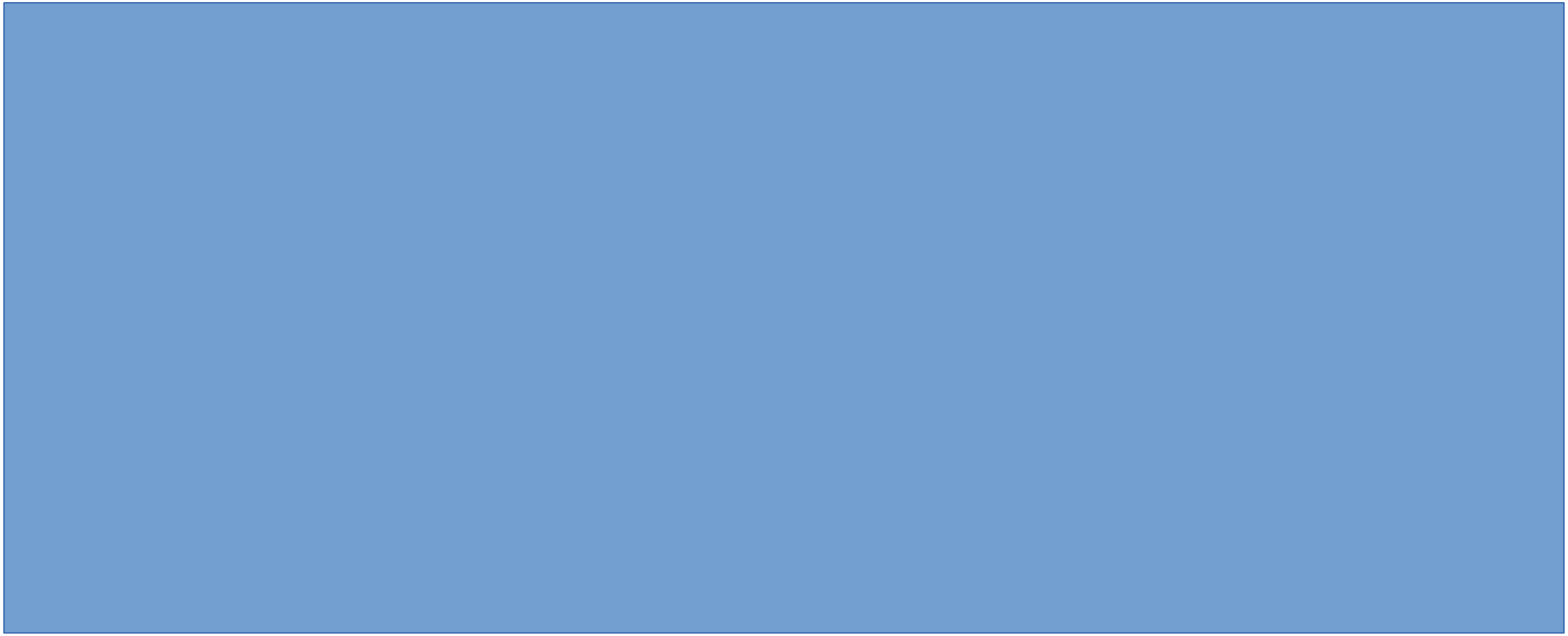
Пример 3.

Нас в этом задании интересуют следующие:

- общий список преподавательского состава с указанием квалификации для ведения отчетности,
- общий список школьников с указанием возраста для ведения отчетности,
- общий список всех людей имеющих доступ в школу, для выдачи магнитных карт,
- список ученков класса вместе с родителями для организации собраний,
- Электронный журнал – каждая страница которого связывает отчетность о посещении/оценках школьников определенного класса по датам, с учебным предметом и преподавателем.

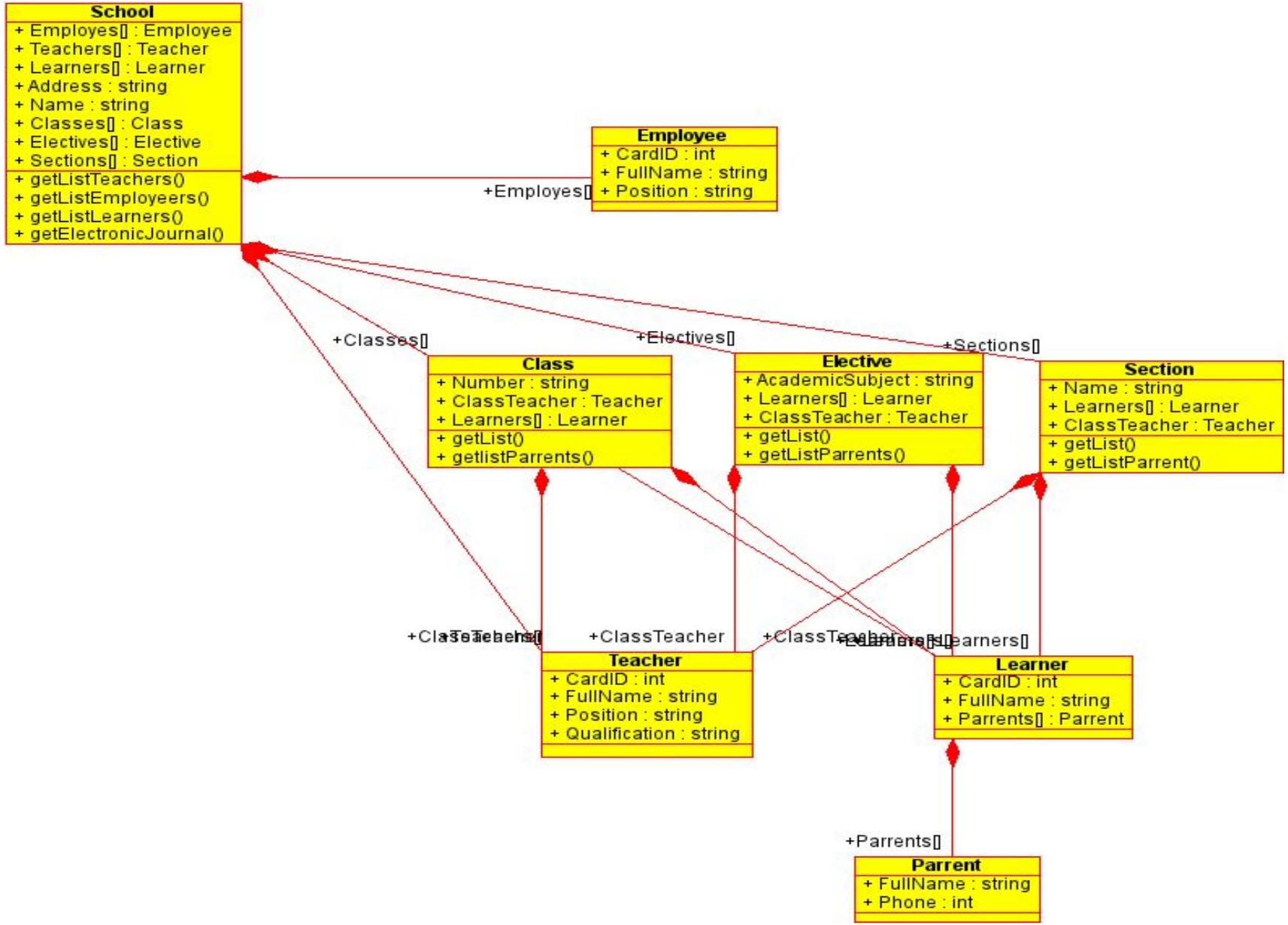
Пример 3.

Описывать процесс формирования журнала не будем – вы и так знаете. Давайте проведем анализ технического задания (сущность,свойства,действия):



Пример 3.

Пример 3.


School
+ Employes[] : Employee
+ Teachers[] : Teacher
+ Learners[] : Learner
+ Address : string
+ Name : string
+ Classes[] : Class
+ Electives[] : Elective
+ Sections[] : Section
+ getListTeachers()
+ getListEmployeers()
+ getListLearners()
+ getElectronicJournal()
Employee
+ CardID : int
+ FullName : string
+ Position : string
+Employes[]
+Classes[]
+Electives[]
+Sections[]
Class
+ Number : string
+ ClassTeacher : Teacher
+ Learners[] : Learner
+ getList()
+ getlistParrents()
Elective
+ AcademicSubject : string
+ Learners[] : Learner
+ ClassTeacher : Teacher
+ getList()
+ getListParrents()
Section
+ Name : string
+ Learners[] : Learner
+ ClassTeacher : Teacher
+ getList()
+ getListParrent()
+ClassTeacher
Teacher
+ CardID : int
+ FullName : string
+ Position : string
+ Qualification : string
Learner
+ CardID : int
+ FullName : string
+ Parrents[] : Parrent
+Parrents[]
Parrent
+ FullName : string
+ Phone : int

Пример 3.

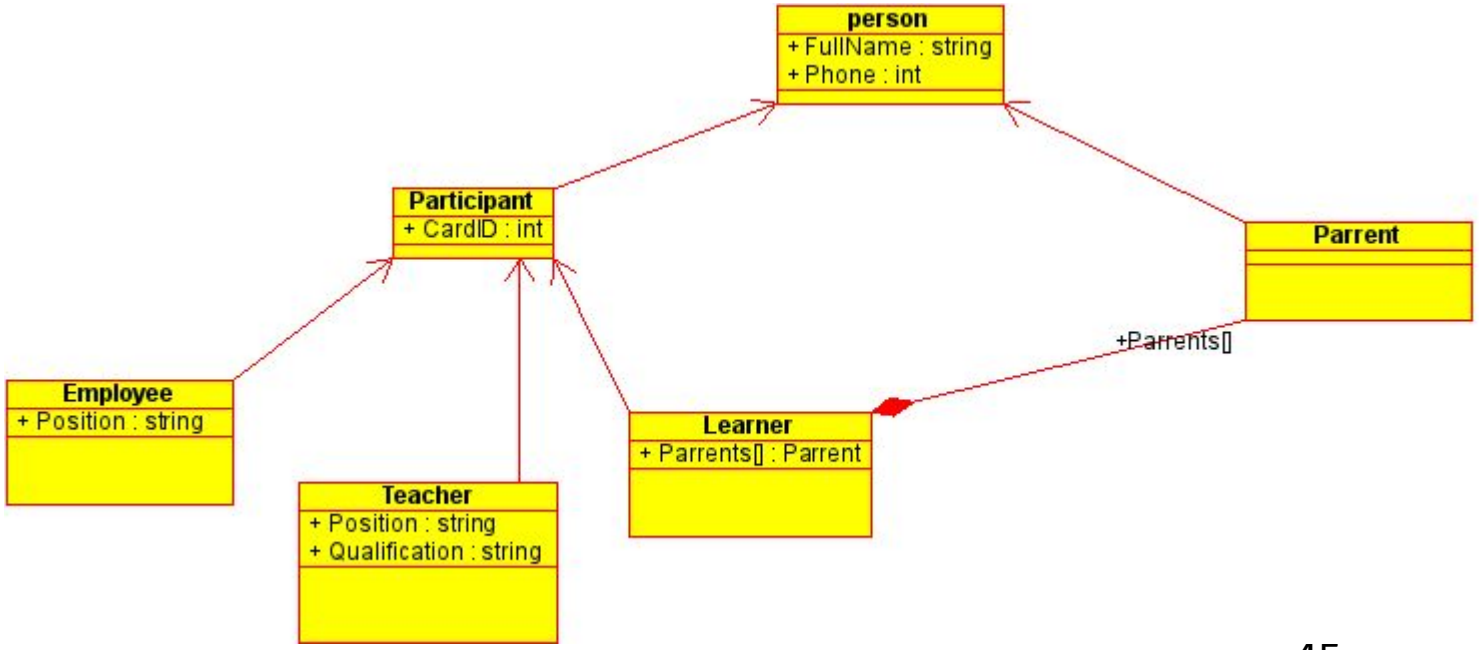
На предыдущем слайде мы получили диаграмму классов нашей будущей программы. Однако в ней имеется ряд вопросов.

Во первых, в некоторых классах есть повторяющиеся свойства (например ФИО,CardID). Это наводит на мысль об общности этих классов.

Во вторых, в постановке задачи был отчет – общий список всех людей имеющих доступ школу. Однако добавить такой метод в класс School не получается, так как в нашей программе нет обобщенного типа для всех участников учебного процесса, а следовательно не возможно вернуть массив объектов неодинаковых типов.

Пример 3.

Для решения этих вопросов выделим суперклассы для всех участников, описанных в анализе ТЗ. Небольшой анализ показывает что самый общий класс для всех – Person (персона, человек) с полями ФИО и тел. Также выделим две разновидности персон – имеющих доступ в школу (с карточкой – персонал, учителя, ученки) и не имеющих доступа (родители)



Пример 3.

Изменяем схему с учетом наследования классов

Пример 3.

В приведенной диаграмме поднятые вопросы разрешены. Теперь в классе School есть метод, который нам вернет массив участников учебного процесса. Обратите внимание, что вследствие полиморфизма вы получите список именно участников (поля ФИО, phone, CardID) не зависомо от того какие роли будут у каждого конкретного участника – ученик, учитель или сотрудник. Однако общие методы этих объектов при вызове буду вызваны именно соотвествуя фактическому классу.

Конечно мы только начали проектировать систему в вышеприведенном UML, но уже видим что использование полиморфизма и наследования дает дополнительную гибкость в проектировании.

Заключение.

Парадигма ООП давно и устойчиво утвердилась как основная при разработке ПО. В представленной презентации я старался рассказать просто о сложном, так как проектирование ПО - это очень сложный процесс, которым обычно занимаются самые высококвалифицированные специалисты – системные архитекторы. При проектировании они, помимо основных принципов опираются на:

- глубокое знание computer science, опыт программирования, построения и управления структурами данных,
- на знание всевозможных фреймворков, программных платформ, сред разработки, серверов баз данных и серверов приложений
- опыт внедрения и сопровождения ПО

для того, чтобы созданный ими проект мог быть не только реализован в кратчайшие сроки наиболее эффективным способом, но и чтобы у заказчика он был наиболее эффективен, то есть выдавал заказанную функциональность с наименьшими затратами.

Заключение.

Однако даже если вы делаете свою маленькую программку, которой кроме вас возможно никто и не воспользуется, все равно не стоит пренебрегать проектированием. Этот этап поможет вам заранее устранить возможные противоречия в ТЗ, поможет уменьшить количество повторных разработок, которые возникают, когда выясняется что изначально был выбран неверный/неэффективный способ решения задачи. Кроме того очень многие крупные проекты начинались как маленькие программки для личного пользования, и грамотное проектирование помогало им развиваться во что то большее.

Сквозное задание 3.1.1

Проведите процесс объектно ориентированной разработки вашего приложения:

- формализация постановки задачи
- анализ ТЗ
- синтез модели
- построение UML диаграммы классов
- написание Java классов
- запуск приложения.

Спасибо!

SAMSUNG