Основы объектно—ориентированного программирования. Лабораторные

Александр Панов

Московский физико-технический институт

февраль 2015 г.

Цели курса

- Освоить идеологию объектно—ориентированного программирования.
- Понять принципы программирования структур данных и типовых решений (patterns).
- Научиться писать программы на объектно—ориентированном языке (Java, C++, Python).
- Начать создавать безопасные и легко понимаемые программы.
- Научиться работать в команде с использованием средств командной разработки кода.
- Освоить основы параллельного программирования.
- Начать пользоваться стандартными и сторонними библиотеками для решения своих задачах.
- Овладеть инструментами компиляции, отладки и сборки сложных программ.

Работа в семестре

- Сформировать команды минимум по 3 человека, максимум — 5 (конец февраля).
- Определиться с языком программирования в команде и темой курсового проекта (конец февраля).
- Подготовить презентацию своего проекта (конец марта).
- Выполнить две семестровых задачи (конец марта).
- Сдать курсовой проект (май).

Среда разработки и система контроля версий — по своему усмотрению.

Литература

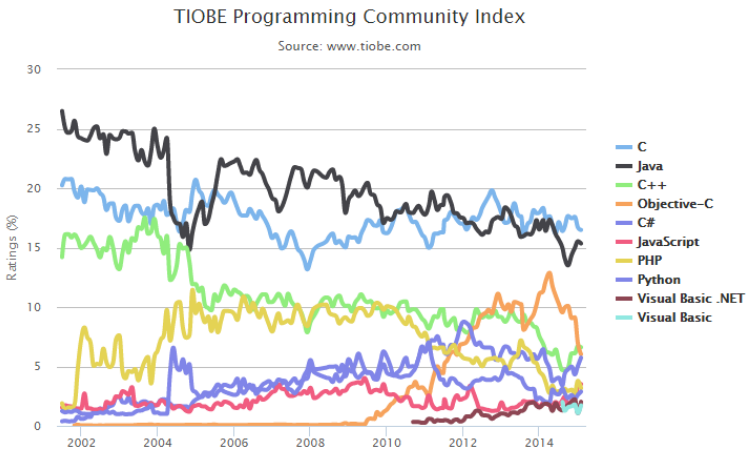

Объектно-ориентированный анализ и проектирование с примерами приложений / Г. Буч, Р. А. Максимчук, М. У. Энгл и др. — 3-е изд. — М. : Вильямс, 2010. — С. 720.


Страуструп Б. Язык программирования C++. — 3-е изд. — М. : Бином, 2008. — С. 1104.


Эккель Б. Философия Java. — 4-е изд. — М. : Питер, 2014. — С. 640.

TIOBE Index

Индекс, оценивающий популярность языков программирования. Основан на подсчёте результатов поисковых запросов, содержащих название языка (Google, Blogger, Wikipedia, YouTube, Baidu, Yahoo!, Bing, Amazon).
http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html



Feb 2015	Feb 2014	Change	Programming Language	Ratings	Change
1	1		C	16.488%	-1.85%
2	2		Java	15.345%	-1.97%
3	4	↑	C++	6.612%	-0.28%
4	3	↓	Objective-C	6.024%	-5.32%
5	5		C#	5.738%	-0.71%
6	9	↑	JavaScript	3.514%	+1.58%
7	6	↓	PHP	3.170%	-1.05%
8	8		Python	2.882%	+0.72%
9	10	↑	Visual Basic .NET	2.026%	+0.23%
10	-	⇈	Visual Basic	1.718%	+1.72%
11	20	⇈	Delphi/Object Pascal	1.574%	+1.05%
12	13	↑	Perl	1.390%	+0.50%
13	15	↑	PL/SQL	1.263%	+0.66%
14	16	↑	F#	1.179%	+0.59%
15	11	⇊	Transact-SQL	1.124%	-0.54%
16	30	⇈	ABAP	1.048%	+0.69%
17	14	↓	MATLAB	1.033%	+0.39%
18	44	⇈	R	0.963%	+0.71%
19	17	↓	Pascal	0.960%	+0.41%
20	12	⇊	Ruby	0.873%	-0.05%

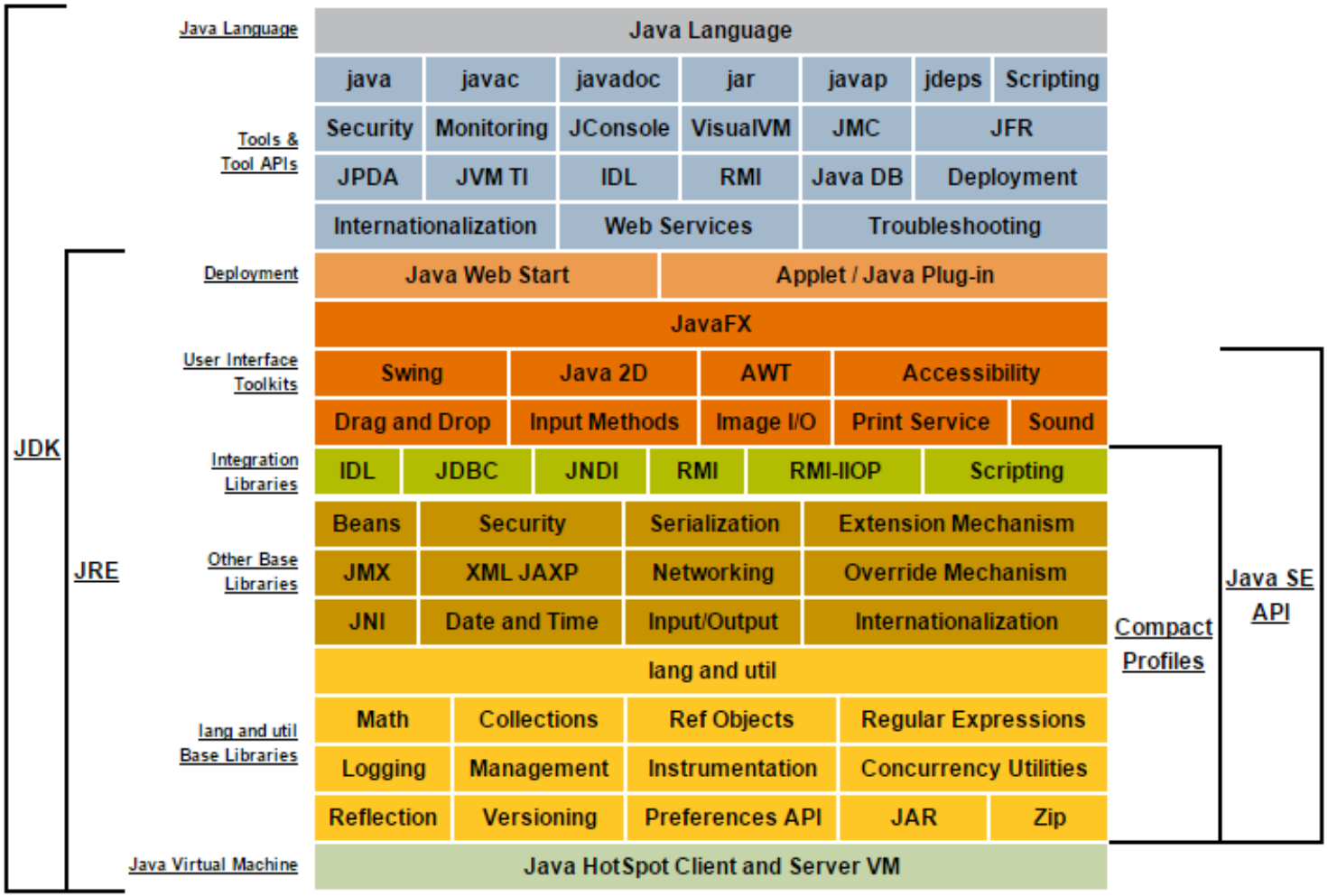
ООП на примере языка C++

- История с 1980 г.: изначально «C with classes», крайняя версия — C++11.
- Стандартизация с 1996 г.
- Ключевая особенность — полная совместимость с C.
- Высокая производительность.
- Наличие совместимости с C приводит к путанице при использовании устаревших функций.
- Большое количество библиотек, в том числе и с дублирующими функциями.

ООП на примере языка Java

- История с 1995 г.: 6 версий — крайняя JDK 1.8.
- Поддержка Sun—Oracle http://docs.oracle.com/javase/8/docs/
- Ключевая особенность — программы транслируются в байт-код, выполняемый виртуальной машиной Java (JVM). JVM реализована для всех типов операционных систем.
- Облегченное управление памятью — сборка мусора garbage collector (GC).
- Программные стеки: JavaSE (desktop—приложения), JavaEE (web—приложения), JavaFX (rich—приложения), Android (мобильные приложения).
- Богатый набор уже написанного кода и большое количество библиотек и фреймворков (frameworks), решающих огромное количество задач.

Компоненты языка Java



Инструменты языка C++

- STandart Library (STL) — библиотека шаблонов.
- Boost — одна из самых известных библиотек инструментов.
- make — инструмент сборки программ.
- gdb — инструмент отладки.

Примеры на Java

```
double a = 1, b = 1, c = 6;
double D = b * b - 4 * a * c;
if (D >= 0) {
    double x1 = (-b + Math.sqrt (D)) / (2 * a);
    double x2 = (-b - Math.sqrt (D)) / (2 * a);
}

int x = 2;
int y = 0;
/* if (x > 0)
        y = y + x*2;
    else
        y = -y - x*4; */
y = y*y;// + 2*x;
```

Hello World! на Java

```
public class Demo {

    public static void main (String args[]) {
        System.out.println("Hello, world!");
    }
}
```

Команда компиляции — javac Demo.java

Команда запуска скомпилированного приложения — java Demo

Лексика языка

- Идентификаторы — это имена, которые даются различным элементам языка для упрощения доступа к ним. Имена имеют пакеты, классы, интерфейсы, поля, методы, аргументы и локальные переменные.
- Ключевые слова — это зарезервированные слова, состоящие из ASCII—символов и выполняющие различные задачи языка: abstract, double, int, class, public, void и т. п.
- Литералы позволяют задать в программе значения для числовых, символьных и строковых выражений, а также null—литералов.
- Операторы используются в различных операциях — арифмтеических, логических, битовых, опреациях сравнения и присваивания: =, ==, >, <, +, - и т. п.

Интернет на виртуальных контейнерах

```
ping 64.0.0.0 -c 2 -w2 || wget -qO -
    "login.telecom.mipt.ru/bin/login.cgi?login=LOGIN
    &memorize=on&password=
$((wget login.telecom.mipt.ru/bin/getqc.cgi -qO -; echo -n
    PASSWORD) | md5sum - | head -c32)"
```

Парадигмы программирования

Парадигма программирования — это совокупность идей и понятий по структурированию своей работы по написанию компьютерных программ.

Императивное программирование — вычисление описывается последовательностью инструкций, которые изменяют состояние данных. Возникает последовательность состояний как в теории автоматов. Базовое понятие — *переменная*.

1. Процедурная парадигма.
2. Структурная парадигма.
3. Объектно-ориентированная парадигма.

Парадигмы программирования

Парадигма программирования — это совокупность идей и понятий по структурированию своей работы по написанию компьютерных программ.

Декларативное программирование — декларирует состояние, а не задаёт путь к его вычислению. Здесь главное описать строение чего-то, а не процесс его создания.

1. Функциональная парадигма: базовое понятие — функция без глобальных переменных (λ-исчисление $\rightarrow$ LISP, Clojure, Scala и др.).
2. Логическая парадигма: заданы факты, правила вывода, на основе метода резолюций происходит автоматическое доказательство теорем (Oz, Prolog).

Процедурная и структурная парадигмы

- Процедурная методология основана на алгоритмах (Марков, Тьюринг, фон Нейман).
- Последовательное выполнение операторов, преобразующих состояние памяти. Чёткое отделение программы от памяти.
- Большие задачи разбиваются на подзадачи — процедуры (функции).
- **Переиспользование** состоит в создании библиотек процедур (функций).
- Модули как совокупности процедур — структурное программирование без goto (Дейкстра).
- Примеры: Ada, Algol, Visual Basic, C, Fortran, Pascal.

Объекты

Гради Буч:
Объект — это мыслимая или реальная сущность, обладающая характерным поведением и отличительными характеристиками и являющаяся важной в предметной области.

Каждый объект имеет состояние, обладает чётко определённым поведением и уникальной идентичностью.

Состояние: в любой момент времени состояние объекта включает в себя перечень (обычно статический) свойств объекта и текущие значения (обычно динамические) этих свойств. Человек сидит и у него есть удочка.

Объекты

Гради Буч:
Объект — это мыслимая или реальная сущность, обладающая характерным поведением и отличительными характеристиками и являющаяся важной в предметной области.

Каждый объект имеет состояние, обладает чётко определённым поведением и уникальной идентичностью.

Поведение: для каждого объекта существует определённый набор действий, которые с ним можно произвести. Файл в ОС можно открыть, создать и т.п.

Объекты

Гради Буч:
Объект — это мыслимая или реальная сущность, обладающая характерным поведением и отличительными характеристиками и являющаяся важной в предметной области.

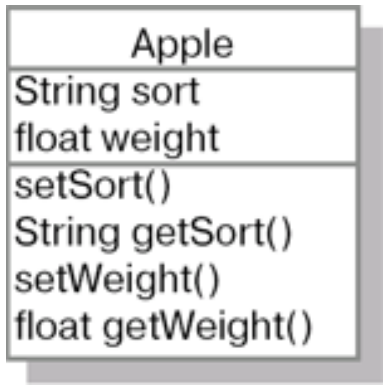
Каждый объект имеет состояние, обладает чётко определённым поведением и уникальной идентичностью.

Уникальность: в машинном представлении под параметром уникальности объекта чаще всего понимается адрес размещения объекта в памяти; уникальность объекта состоит в том, что всегда можно определить, указывают две ссылки на один и тот же объект или на разные объекты. Даже одинаковые монеты (абсолютно все их атрибуты одинаковы: год выпуска, номинал и т.д.), они по-прежнему остаются разными монетами.

Классы

- Совокупность атрибутов и их значений характеризует объект.
- Все объекты одного и того же класса описываются одинаковыми наборами атрибутов.
- Все объекты одного и того же класса обладают одинаковым поведением.

Пример 1: разные объекты класса «Монеты».
Пример 2: конюшня и лошадь как объекты одного класса.



Классы

- Класс имеет **имя**, которое относится ко всем объектам этого класса.
- В классе вводятся имена атрибутов, которые определены для объектов (атрибут=свойство=**поле**).
- Класс является шаблоном поведения объектов (**методы**)
- Класс может иметь **конструктор** (constructor) — специальный метод, который выполняется при создании объектов.
- Класс может иметь **деструктор** (destructor) — специальный метод, который выполняется при уничтожении объектов.

Инкапсуляция

Инкапсуляция (encapsulation) — это сокрытие реализации класса и отделение его внутреннего представления от внешнего (интерфейса).

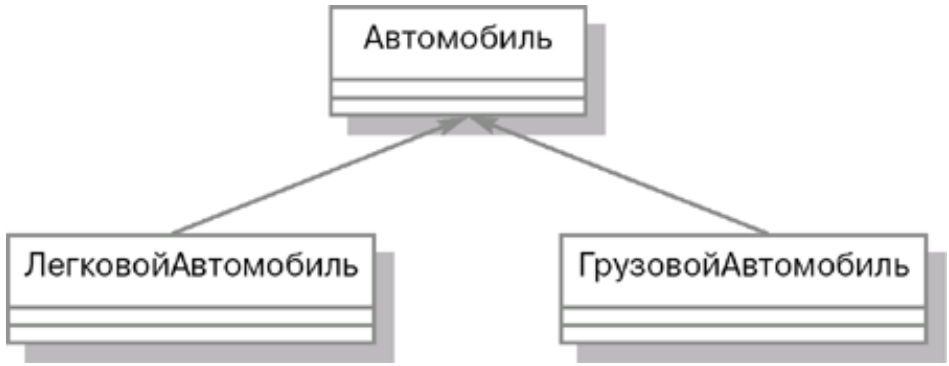
Внутри объекта данные и методы могут обладать различной степенью открытости (или доступности).

- Открытые члены класса составляют внешний интерфейс объекта — это та функциональность, которая доступна другим классам.
- Закрытыми обычно объявляются все свойства класса, а также вспомогательные методы, которые являются деталями реализации и от которых не должны зависеть другие части системы.

Модульность — благодаря сокрытию реализации за внешним интерфейсом класса можно менять внутреннюю логику отдельного класса, не меняя код остальных компонентов системы.

Наследование

Наследование (inheritance) — это отношение между классами, при котором класс использует структуру или поведение другого класса (одиночное наследование), или других (множественное наследование) классов.



Наследование вводит иерархию «общее/частное», в которой **подкласс** наследует от одного или нескольких более общих **суперклассов**.

Типичная задача

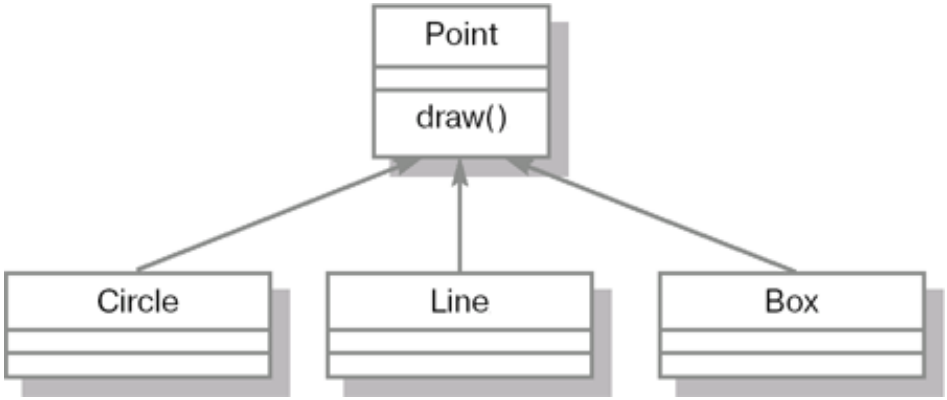
Пример:
Предположим, мы хотим создать векторный графический редактор, в котором нам нужно описать в виде классов набор графических примитивов — Point, Line, Circle, Box и т.д. У каждого из этих классов определим метод draw для отображения соответствующего примитива на экране.

Хотим:
Написать код, который при необходимости отобразить рисунок будет последовательно перебирать все примитивы, на момент отрисовки находящиеся на экране, и вызывать метод draw у каждого из них.

Решение 1

```
Point[] p = new Point[1000];
Line[] l = new Line[1000];
Circle[] c = new Circle[1000];
Box[] b = new Box[1000];
//...
//...
for(int i = 0; i < p.length;i++) {
    if(p[i]!=null) p[i].draw();
}
for(int i = 0; i < l.length;i++) {
    if(l[i]!=null) l[i].draw();
}
for(int i = 0; i < c.length;i++) {
    if(c[i]!=null) c[i].draw();
}
for(int i = 0; i < b.length;i++) {
    f(b[i]!=null) b[i].draw();
}
```

Решение 2




```
Point p[] = new Point[1000];
p[0] = new Circle();
p[1] = new Point();
p[2] = new Box();
p[3] = new Line();
//...
for(int i = 0; i < p.length;i++) {
    if(p[i]!=null) p[i].draw();
}
```

Полиморфизм

Полиморфизм (polymorphism) — положение теории типов, согласно которому имена (например, переменных) могут обозначать объекты разных (но имеющих общего родителя) классов.

Процедурный полиморфизм предполагает возможность создания нескольких процедур или функций с одним и тем же именем, но разным количеством или различными типами передаваемых параметров — **перегрузка** (overloading) функций.

```
void println();
void println(boolean x);
void println(String x);
```

Чтение с консоли

```
import java.util.Scanner;

public class InputExp {
    public static void main(String[] args) {
        String name;
        int age;
        Scanner in = new Scanner(System.in);

        name = in.nextLine();
        age = in.nextInt();
        in.close();

        System.out.println("Name :" + name);
        System.out.println("Age :" + age);
    }
}
```

Чтение из файла

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.util.Scanner;
public class ScannerReadFile {
    public static void main(String[] args) {
        try {
            FileInputStream fileStream = new
    FileInputStream("test.txt");

            Scanner scanner = new Scanner(fileStream);
            while (scanner.hasNextLine()) {
                String line = scanner.nextLine();
                System.out.println(line);
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```