



Java

Введение

Программа курса

- Введение в язык программирования Java и объектно-ориентированное программирование.
- Базовые типы данных. Операции.
- Управляющие операторы.
- Классы.
- Наследование.
- Работа со строками. Пакеты. Интерфейсы.
- Исключения.
- Ввод/вывод
- Классы коллекций

Литература и источники

Официальная документация:

<https://docs.oracle.com/javase/>

Книги:

- Брюс Эккель - «Философия Java»
- Г. Шилдт - Руководство для начинающих
- Хорстманн К., Корнелл Г. - Java. Библиотека профессионала. Том 1, 2

Также есть довольно неплохие tutorials в интернете.

Например, официальный:

<https://docs.oracle.com/javase/tutorial/>

История создания Java – Star7

- Декабрь 1990 – в Sun поставлена задача – «создать что-нибудь необычайное»
- 1 февраля 1991 – Патрик Нотон, Джеймс Гослинг и Майк Шеридан (Mike Sheridan) вплотную приступили к реализации проекта, который получил название Green.
- Апрель 1991 – принято решение разработать небольшое универсальное устройство с жидкокристаллическим сенсорным экраном для управления бытовой, аудио и видео техникой (Star7)
- Июне 1991 – Гослинг начинает разработку замены C++, который он называет OaK (дуб)
- 4 сентября 1992 – состоялась рабочая демонстрация устройства Star7



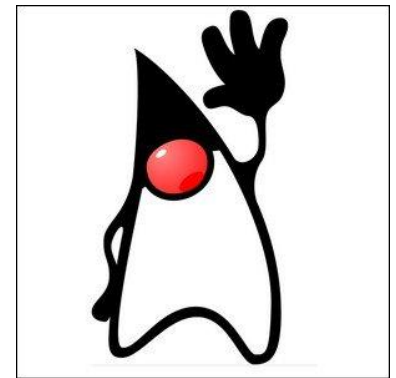
История создания Java – Int.TV

- 1 ноября 1992 – создается компания FirstPerson, которая безуспешно пытается внедрить свою разработку
- Март 1993 – FirstPerson переключается на задачу развертывания сети интерактивного телевидения, но её опережает другая компания
- Осень 1993 – создание игровых приставок 3DO с помощью платформы OaK. Однако директор 3DO потребовал полные права на новый продукт, и сделка не состоялась.
- Начало 1994 – стало понятно, что идея интерактивного телевидения оказалась нежизнеспособной, а Sun требует от FirstPerson новый бизнес-план



История создания Java – WWW

- 1991 г. – начало разработки HTML в Европейском институте физики частиц (CERN)
- Апрель 1993 – Марк Андрессен и Эрик Бина, выпустили первую версию графического браузера Mosaic 1.0. Впервые обычные пользователи персональных компьютеров без всякой специальной подготовки могли пользоваться глобальной сетью
- Начало 1994 – находящаяся под угрозой закрытия FirstPerson начинает адаптацию *OaK* к WWW. Оказалось, что эта платформа удивительным образом отвечает всем требованиям Internet-программирования, хотя и создавалась во времена, когда про WWW никто даже и не думал.
- Июль 1994 – разработан браузер *WebRunner*, поддерживающий платформу *OaK*
- Осень 1994 – демонстрация *WebRunner*, показывающая возможности программной платформы на стороне клиента заканчивается бурными авациями.

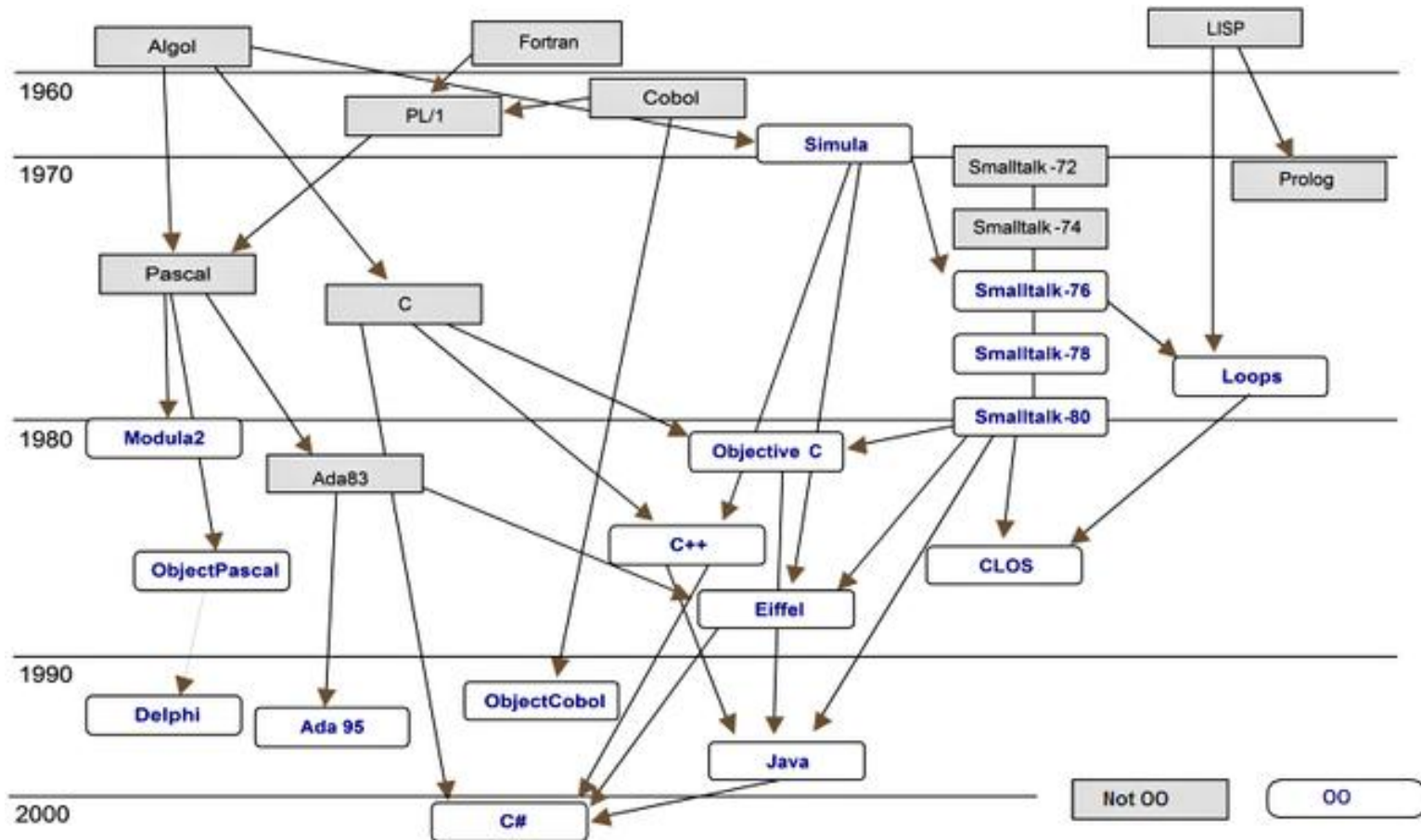


История создания Java

- Начало 1995 года за дело взялись маркетологи. В результате их исследований OaK был переименован в Java, а WebRunner стал называться HotJava.
- Март 1995 – количество скачиваний OaK и WebRunner переваливает за 10 тыс.
- 23 марта 1995 – первая статья с описанием платформы Java и первое публичное упоминание сайта <http://java.sun.com/>.
- 23 мая 1995 – технологии Java и HotJava были официально объявлены Sun и тогда же представители компании сообщили, что новая версия самого популярного браузера Netscape Navigator 2.0 будет поддерживать новую технологию.
- Java становится такой же неотъемлемой частью WWW



Языки программирования

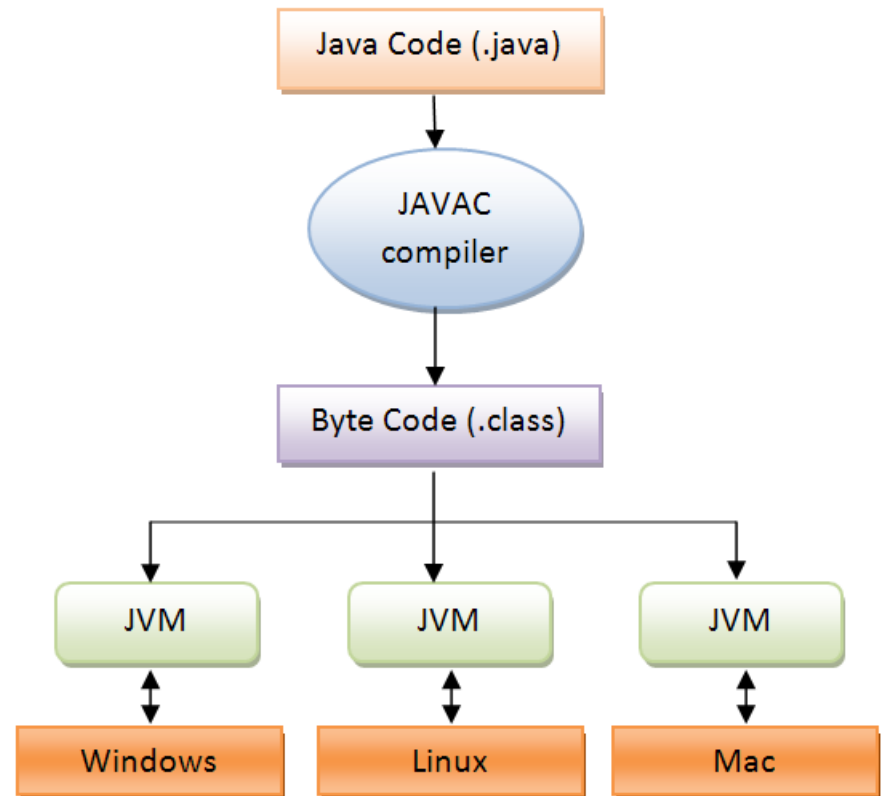


Версии Java

1. JDK Alpha and Beta (1995)
2. JDK 1.0 (January 23, 1996)
3. JDK 1.1 (February 19, 1997)
4. J2SE 1.2 (December 8, 1998)
5. J2SE 1.3 (May 8, 2000)
6. J2SE 1.4 (February 6, 2002)
7. J2SE 5.0 (September 30, 2004)
8. Java SE 6 (December 11, 2006)
9. Java SE 7 (July 28, 2011)
10. Java SE 8 (March 18, 2014)

Платформа

- Как правило под платформой понимают сочетание аппаратной архитектуры (в основном определяется типом процессора) и ОС.
- Приложения на языке Java исполняются в специальной, универсальной среде, которая называется Java Virtual Machine. JVM - это программа, которая пишется специально для каждой реальной платформы, чтобы, с одной стороны, скрыть все ее особенности, а с другой - предоставить единую среду исполнения для Java -приложений. *Виртуальная машина Java* интерпретирует байт-код Java, предварительно созданный из исходного текста Java-программы компилятором Java.

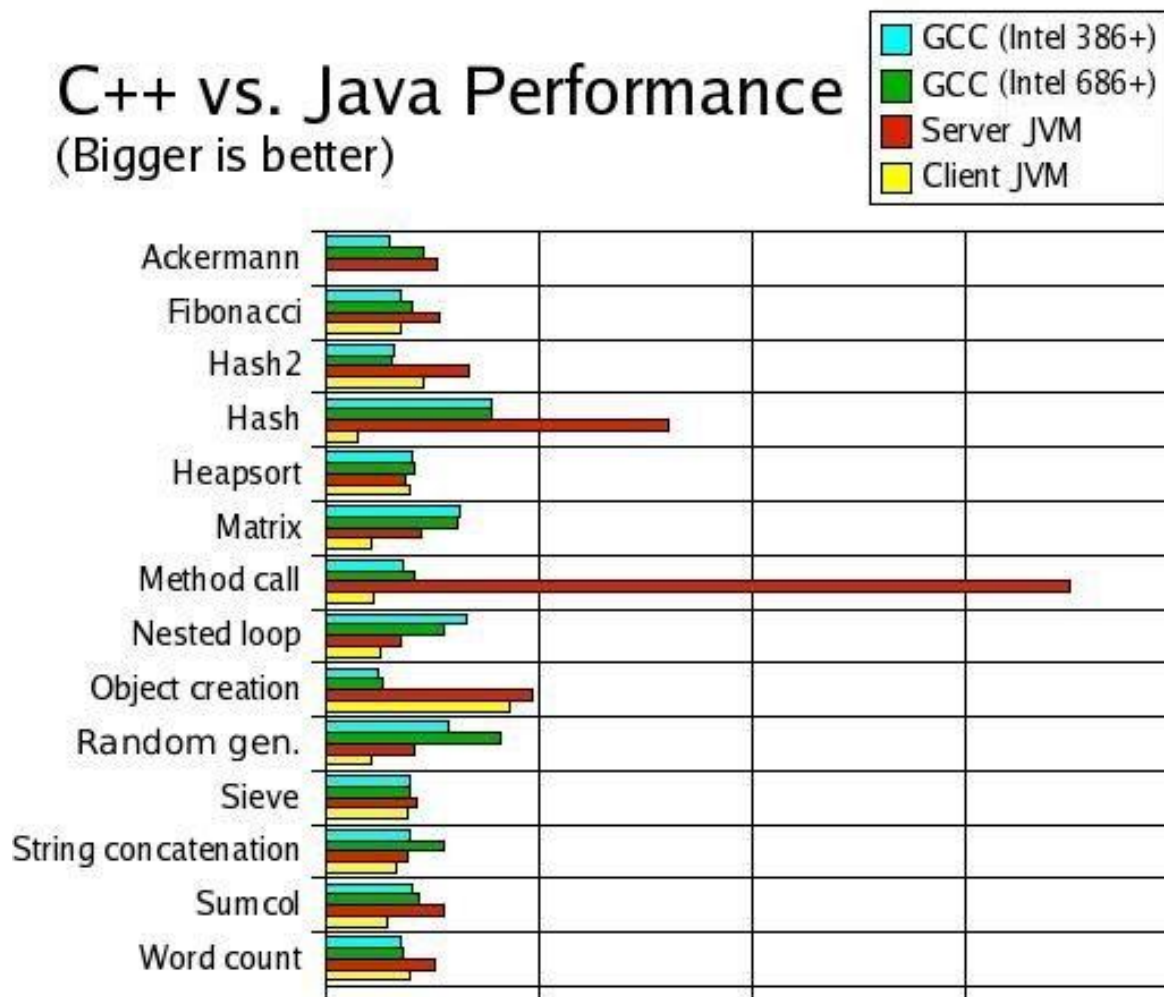


Свойства Java

- кроссплатформенность
- объектная ориентированность – эффективная ООП модель
- строгая, статическая, явная типизация
- привычный синтаксис C/C++
- механизм автоматической сборки мусора - garbage collector
- безопасность :
 - Автоматическая очистка памяти
 - JVM контролирует действия исполняемого кода. Можно устанавливать определенные права (например, апплеты не получали по умолчанию доступа к файловой системе)
 - механизм подписания апплетов и других приложений, загружаемых по сети. Специальный сертификат гарантирует, что пользователь получил код именно в том виде, в каком его выпустил производитель.
- ориентация на Internet-задачи, сетевые распределенные приложения
- динамичное развитие языка, множество opensource проектов

Производительность Java

C++ vs. Java Performance (Bigger is better)



Платформы Java

- Область применения Java
- Серверное ПО
 - Плагины
 - Мобильные приложения
 - Android
 - Сервисы
 - Настольное ПО
 - Апплет
 - JSP
 - Smart-TV
 - ...
- **J2SE** предназначается для использования на рабочих станциях и персональных компьютерах. Standard Edition - основа технологии Java и прямое развитие JDK (средство разработчика было переименовано в j2sdk).
 - **J2EE** содержит все необходимое для создания сложных, высоконадежных, распределенных серверных приложений. Условно можно сказать, что Enterprise Edition - это набор мощных библиотек (например, Enterprise Java Beans, EJB) и пример реализации платформы (сервера приложений, Application Server), которая их поддерживает. Работа такой платформы всегда опирается на j2sdk.
 - **J2ME** является усечением Standard Edition, чтобы удовлетворять жестким аппаратным требованиям небольших устройств, таких как карманные компьютеры и сотовые телефоны

Некоторые компоненты платформы Java

- Java language specification (JLS) - спецификация языка Java (описывающая лексику, типы данных, основные конструкции и т.д.);
- спецификация JVM - предназначена в первую очередь для создателей виртуальных машин.
- Java Development Kit (JDK) - средство разработчика, состоящее в основном из утилит, стандартных библиотек классов и демонстрационных примеров. Оно не IDE, а оперирует только уже существующими Java -файлами.
 - `javac` – компилятор
 - `java` - виртуальная машина
 - `javadoc` - средство автоматической генерации документации на основе исходного кода
- *Just-In-Time (JIT) – компилятор, транслирующий байт-код программы Java в "родной" код операционной системы. Таким образом, время запуска программы увеличивается, но зато выполнение может ускоряться.*
- *Java Runtime Environment, JRE - среда выполнения Java, то есть это реализация виртуальной машины, необходимая для исполнения Java - приложений, без компилятора и других средств разработки.*

Стандартные библиотеки Java

java.lang	базовая функциональность языка и основные типы
java.util	коллекция классов структур данных
java.io	операции ввода-вывода
java.math	математические операции
java.nio	фреймворк для неблокирующего ввода-вывода
java.net	операции с сетями, сокетами, DNS-запросами
java.security	генерация ключей, шифрование и дешифрование
java.sql	Java Database Connectivity (JDBC) для доступа к базам данных
java.awt	иерархия основных пакетов для родных компонентов GUI
javax.swing	иерархия пакетов для платформенной независимости GUI компонентов

Объектно-ориентированное программирование

- Все является объектом. Объект как хранит информацию, так и способен ее обрабатывать. Теоретически любой элемент решаемой задачи может представлять собой объект.
- Программа - совокупность объектов, указывающих друг другу что делать, что делать, посредством сообщений
- Каждый объект имеет свою собственную «память» состоящую из других объектов. Таким образом новый объект создается на основе существующих
- У каждого объекта есть тип (класс). Он определяет какие сообщения объекты могут посылать друг другу. Все объекты одного типа могут получать одинаковые сообщения.

То есть объект обладает состоянием, поведением и индивидуальностью (Буч)

Объектно-ориентированное программирование

ООП – это парадигма программирования, в которой программа представляет собой набор взаимодействующих объектов, которые описывают соответствующие сущности предметной области.

Абстракция — это способ выделить набор значимых характеристик объекта, исключая из рассмотрения незначимые. Соответственно, абстракция — это набор всех таких характеристик.

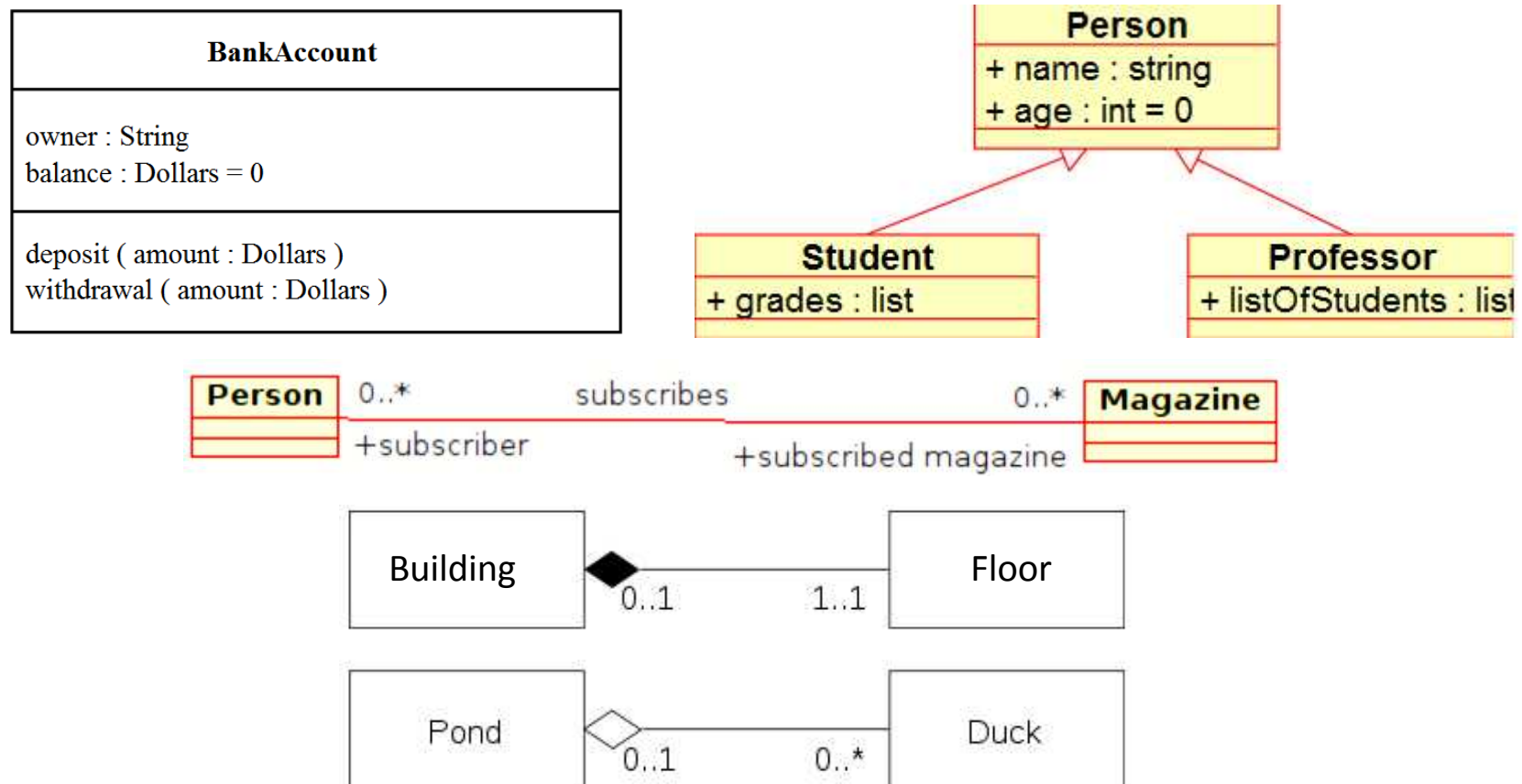
Инкапсуляция — это свойство системы, позволяющее объединить данные и методы, работающие с ними в классе, и скрыть детали реализации от пользователя.

Наследование — это свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствуемой функциональностью. Класс, от которого производится наследование, называется базовым, родительским или суперклассом. Новый класс — потомком, наследником или производным классом.

Полиморфизм — возможность объектов с одинаковой спецификацией иметь различную реализацию. .

Диаграмма классов

Диаграмма классов (Class diagram) — статическая структурная диаграмма, описывающая структуру системы, демонстрирующая классы системы, их атрибуты, методы и зависимости между классами.



Первая программа

Файл: Hello.java

```
class MyFirstClass {  
    public String name;  
  
    public void welcome() {  
        System.out.println("Hello, " + name);  
    }  
}  
  
public class Hello {  
    public static void main(String[] args) {  
        MyFirstClass myOb = new MyFirstClass();  
        myOb.name = "Bob";  
        myOb.welcome();  
    }  
}
```

Где здесь состояние, поведение и индивидуальность?

Стиль идентификаторов в Java

CamelCase — стиль написания составных слов, при котором несколько слов пишутся слитно без пробелов, при этом каждое слово пишется с заглавной буквы.

Примеры CamelCase-написания: BackColor, backColor, CamelCase.

В языке Java принято использовать UpperCamelCase для наименования классов и lowerCamelCase — для наименования экземпляров классов и методов.

```
class Cycle {  
    void ride() {  
        System.out.println ("Cycle ride");  
    }  
    void buy() {  
        System.out.println ("Cycle has been bought");  
    }  
}
```

Лексика программы

Компилятор, анализируя программу, разделяет ее на:

- разделители (white spaces);
- комментарии (comments);
- основные лексемы (tokens).

Комментарии – элементы исходного текста программы, которые игнорируются компилятором. В Java бывают двух типов:

- Строчные комментарии
 - `int y=1970; // год рождения`
- Блочные комментарии
 - `/* Этот цикл не может начинаться с нуля
из-за особенностей алгоритма */`

Лексема (от греч. *lexis* — слово, выражение) - минимальная единица языка, имеющая самостоятельный смысл

1. Зарезервированные слова и специальные символы
(if, while, class, {, }, /, *, ...)
2. Литералы (Неименованные константы)
(2, 2.5, "Hello, World!!!",...)
3. Идентификаторы
(iProb, rSpeed, bSuccess, Integer,...)

Литералы

- Целочисленные - целочисленные значения в десятичном, восьмеричном и шестнадцатеричном виде:

`10; 012; 0xA`

- Дробные литералы

`3.14; 2.; .5; 7e10; 3.1E-20`

- Логические литералы

`true` и `false`

- Символьные литералы

`'A'; '\u0041';`

`\b \u0008` backspace BS - забой

`\t \u0009` horizontal tab HT - табуляция

`\n \u000a` linefeed LF - конец строки

`\f \u000c` form feed FF - конец страницы

`\r \u000d` carriage return CR - возврат каретки

`\\" \u0022` double quote " - двойная кавычка

`\' \u0027` single quote ' - одинарная кавычка

`\\ \u005c` backslash \ - обратная косая черта

`\шестнадцатеричный код` от `\u0000` до `\u00ff` символа
в шестнадцатеричном формате.

Литералы

- Строковые литералы

`" " // литерал нулевой длины`

`"\" " //литерал, состоящий из одного символа "`

`"Простой текст" //литерал длины 13`

`// выражение-константа, составленное из двух`

`// литералов`

`"Длинный текст " +`

`"с переносом";`

`"Hello, world!\r\nHello!"`

- Null-литерал: `null`

Типы данных

Переменная определяется идентификатором, типом и значением. (Java - ЯП с сильной статической типизацией)

Объявление переменной:

<тип> <идентификатор> [<инициализатор>]

Типы данных в Java делятся на:

- Простые (примитивные - primitive)
- Ссылочные (reference)

Инициализация:

```
int a;  
int b = 0, c = 3+2;  
int d = b+c;  
int e = a = 5;
```


Примитивные типы

Primitive Type	Size	Minimum Value	Maximum Value
char	16-bit	Unicode 0	Unicode $2^{16}-1$
byte	8-bit	-128	+127
short	16-bit	-2^{15} (-32,768)	$+2^{15}-1$ (32,767)
int	32-bit	-2^{31} (-2,147,483,648)	$+2^{31}-1$ (2,147,483,647)
long	64-bit	-2^{63} (-9,223,372,036,854,775,808)	$+2^{63}-1$ (9,223,372,036,854,775,807)
float	32-bit	32-bit IEEE 754 floating-point numbers	
double	64-bit	64-bit IEEE 754 floating-point numbers	
boolean	1-bit	true or false	
<i>void</i>	-----	-----	-----

Целочисленные типы

```
int i = 1
int i = -2147483648
int i = 2147483648L
long i = 0L
long i = 1111111111111111111L
```

Ошибка?

Над целочисленными аргументами можно производить следующие операции:

- операции сравнения (возвращают булево значение)
 - <, <=, >, >=
 - ==, !=
- числовые операции (возвращают числовое значение)
 - унарные операции + и -
 - арифметические операции +, -, *, /, %
 - операции инкремента и декремента (в префиксной и постфиксной форме): ++ и --
 - операции битового сдвига <<, >>, >>>
 - битовые операции ~, &, |, ^
- оператор с условием ?:
- оператор приведения типов
- оператор конкатенации со строкой +

Тип операций по умолчанию int, кроме операторов ++, -- и сдвига

Работа с целочисленными типами

Пример:

```
System.out.println(1000*60*60*24*7);
```

```
System.out.println(1000*60*60*24*30);
```

```
//604800000
```

```
//-1702967296
```

Работа с целочисленными типами

Пример:

```
System.out.println(1000*60*60*24*7);
```

```
System.out.println(1000*60*60*24*30);
```

```
//604800000
```

```
//-1702967296
```

```
System.out.println(1000*60*60*24*7);
```

```
System.out.println(1000*60*60*24*30L);
```

```
//604800000
```

```
//2592000000
```

Целочисленные операции

1.

```
byte b=1;  
byte c=b+1;
```

2.

```
int x=2;  
long y=3;  
int z=x+y;
```

3.

```
byte b=5;  
byte c=-b;
```

4.

```
byte x=5;  
byte y1=x++;  
byte y2=x--;  
byte y3=++x;  
byte y4=--x;  
  
print(y1);  
print(y2);  
print(y3);  
print(y4);
```

5.

```
byte x=-128;  
print(-x);  
  
byte y=127;  
print(++y);
```

6.

```
byte x=2;  
byte y=3;  
byte z=(x>y) ? x : y;  
byte abs=(x>0) ? x : -x;
```

Целочисленные переменные со строками и символами

1.

```
int x=1;  
print("x="+x);
```

2.

```
print(1+2+"text");  
print("text"+1+2);
```

3.

```
char c1=10;  
char c2='A';  
int i=c1+c2-'B';  
print(i);
```

4.

```
char c='A';  
print(c);  
print(c+1);  
print("c="+c);  
print('c'+'=' +c);
```

?

Целочисленные переменные со строками и символами

1.

```
int x=1;  
print("x="+x);
```

2.

```
print(1+2+"text");  
print("text"+1+2);
```

3.

```
char c1=10;  
char c2='A';  
int i=c1+c2-'B';
```

4.

```
char c='A';  
print(c);  
print(c+1);  
print("c="+c);  
print('c'+'='+c);
```

1. x=1

2. 3text text12

3. // латинская буква A (\u0041, код 65) // **i=9**

4. A 66 c=A 225

Дробные типы

Название типа	Длина (байты)	Область значений
float	4	3.40282347e+38f ; 1.40239846e-45f
double (по умолчанию)	8	1.79769313486231570e+308 ; 4.94065645841246544e-324

Над дробными аргументами можно производить следующие операции:

- операции сравнения (возвращают булево значение)
 - <, <=, >, >=
 - ==, !=
- числовые операции (возвращают числовое значение)
 - унарные операции + и -
 - арифметические операции +, -, *, /, %
 - операции инкремента и декремента (в префиксной и постфиксной форме): ++ и --
- оператор с условием ?:
- оператор приведения типов
- оператор конкатенации со строкой +

Важно!

Для дробных вычислений появляется уже два типа переполнения – overflow и underflow. Тем не менее, Java и здесь никак не сообщает о возникновении подобных ситуаций. Более того, даже деление на ноль не приводит к некорректной ситуации.

Они определяются спецификацией IEEE 754:

- положительная и отрицательная бесконечности (positive/negative infinity) (1f/0f; -1d/0d) (POSITIVE_INFINITY и NEGATIVE_INFINITY);
- значение "не число", Not-a-Number, сокращенно NaN (0.0/0.0; (1.0/0.0)*0.0) (NaN)
- положительный и отрицательный нули.

Пример:

- `System.out.println(1e20f*1e20f);`
- `System.out.println(-1e200*1e200);`

Булев тип

boolean: всего два возможных значения – true и false

- Над булевыми аргументами можно производить следующие операции:
- операции сравнения (возвращают булево значение)
 - ==, !=
- логические операции (возвращают булево значение)
 - !
 - &, |, ^
 - &&, || - короткая схема выполнения
- оператор с условием ?:
- оператор конкатенации со строкой +

Пример

```
1  import java.util.Scanner;
2
3  public class C2 {
4      public static void main (String[] args){
5          float d1 = 6.0f;
6          float d2 = 1.0f;
7
8          Scanner in = new Scanner (System.in);
9          d2 = in.nextFloat();
10
11         System.out.println(-d1/d2);
12     }
13 }
```

Ссылочный тип данных

Выражение ссылочного типа имеет значение либо null, либо ссылку, указывающую на некоторый объект в виртуальной памяти JVM.

Ссылочный тип данных называют классом. Класс – абстрактный тип данных, описывающий некоторую сущность. Объект – экземпляр класса.

Описание класса:

```
class Point {  
    int x, y;  
  
    draw () {  
        ...  
    }  
  
    Point (int newX, int newY) {  
        x=newx;  
        y=newy;  
    }  
}
```

The diagram is a blue rectangular box containing two rows of text. The top row shows '<ТИП>' followed by a hyphen and '<ПЕРЕМЕННАЯ>'. Below each of these terms are two wavy lines pointing downwards. The bottom row shows '<КЛАСС>' followed by a hyphen and '<ОБЪЕКТ>'. This illustrates that a type is a variable and a class is an object.

<ТИП> - <ПЕРЕМЕННАЯ>
 〰 〰
<КЛАСС> - <ОБЪЕКТ>

Описание класса в Java

```
<доп.атрибуты> class <ИмяКласса> extends <БазовыйКласс>  
                implements <Реализуемые интерфейсы>  
  
{  
    //описание полей/элементов класса:  
    <доп.атрибуты> <тип1> <поле1>;  
    ...  
    //описание методов класса:  
    <доп.атрибуты> <тип2> <имяМетода> (<список  
                                         параметров>) {  
        // тело метода  
    }  
}
```

Создание объекта

Объекты создаются при помощи оператора new.

```
Point p1 = new Point(3,5);  
print(p1.x);  
print(p1.y);
```

```
class Point {  
    int x, y;  
  
    draw () {  
        ...  
    }  
  
    Point (int newX, int newY) {  
        x=newx;  
        y=newy;  
    }  
}
```

Различие между примитивными и ссылочными типами

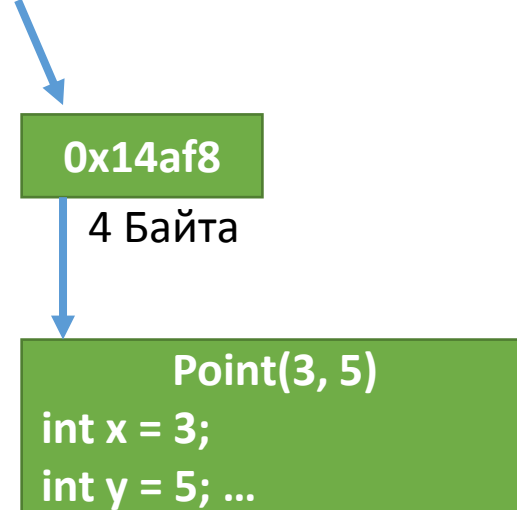
Экземпляры примитивных типов (переменные)

```
int i = 10;
```



Экземпляры ссылочных типов (объекты)

```
Point p = new Point (3,5);
```



Различие между примитивными и ссылочными типами

1.

```
int a=5;  
int b=a;  
a=3;  
print(b);
```

2.

```
class Point {  
    int x, y;  
}  
  
Point p1 = new  
Point(3,5);  
  
Point p2=p1;  
p1.x=7;  
print(p2.x);
```

3.

```
Point p1 = new  
Point(3,5);  
  
Point p2=p1;  
p1 = new Point(7,9);  
print(p2.x);
```


Расширение/наследование класса

```
// Объявляем класс Parent  
class Parent {  
}
```

```
// Объявляем класс Child и наследуем  
// его от класса Parent  
class Child extends Parent {  
}
```

```
Parent p = new Child();
```

```
Parent p = new Child();    Child c = (Child) p;
```

Операции над объектами

- обращение к полям и методам объекта
- оператор `instanceof`
- операции сравнения `==` и `!=`
- оператор приведения типов
- оператор с условием `?:`
- оператор конкатенации со строкой `+`

instanceof

```
class Parent {  
}  
class Uncle {  
}  
class Child extends Parent {  
}  
class Child2 extends Parent {  
}  
class GrandChild extends Child {  
}  
public class InstOf {  
    public static void main (String[] args){  
        Parent p = new GrandChild();  
        System.out.println (p instanceof Child);  
    }  
}
```

Класс Object

Класс Object – общий предок всех классов в Java.

Важные методы класса Object:

Modifier and Type	Method and Description
Object	clone() Creates and returns a copy of this object.
boolean	equals(Object obj) Indicates whether some other object is "equal to" this one.
void	finalize() Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
Class <?>	getClass() Returns the runtime class of this Object.
int	hashCode() Returns a hash code value for the object.
String	toString() Returns a string representation of the object.

getClass()

Этот метод возвращает объект класса Class, который описывает класс, от которого был порожден этот объект.

```
public class ObjMeth {
    public static void main (String[] args){
        String s = "Hello";
        Point p = new Point (3,5);
        Class cl=s.getClass();
        System.out.println(cl.getName());
        System.out.println(p.getClass().getName());
    }
}

// java.lang.String
// p1.Point
```

Класс Class

Является метаклассом для всех классов Java.

Например для вызова:

```
class Flight {  
    ...  
}  
...  
Flight f1 = new Flight(400);
```

в системе будет созданы следующие объекты:

- объект типа `Flight`, описывающий рейс №400;
- объект класса `Class`, описывающий класс `Flight`;
- объект класса `Class`, описывающий класс `Object`;
- объект класса `Class`, описывающий класс `Class`.

Modifier and Type	Method and Description
<U> Class <? extends U>	asSubclass (Class <U> clazz) Casts this Class object to represent a subclass of the class represented by the specified class object.
static Class <?>	forName (String className) Returns the Class object associated with the class or interface with the given string name.
static Class <?>	forName (String name, boolean initialize, ClassLoader loader) Returns the Class object associated with the class or interface with the given string name, using the given class loader.
Constructor <?>[]	getConstructors () Returns an array containing Constructor objects reflecting all the public constructors of the class represented by this Class object.
Method []	getDeclaredMethods () Returns an array of Method objects reflecting all the methods declared by the class or interface represented by this Class object.
Class <?>	getEnclosingClass () Returns the immediately enclosing class of the underlying class.
Method	getEnclosingMethod () If this Class object represents a local or anonymous class within a method, returns a Method object representing the immediately enclosing method of the underlying class.
Field	getField (String name) Returns a Field object that reflects the specified public member field of the class or interface represented by this Class object.
Class <?>[]	getInterfaces () Determines the interfaces implemented by the class or interface represented by this object.
Method	getMethod (String name, Class <?>... parameterTypes) Returns a Method object that reflects the specified public member method of the class or interface represented by this Class object.
String	getName () Returns the name of the entity (class, interface, array class, primitive type, or void) represented by this Class object, as a String.
Package	getPackage () Gets the package for this class.
boolean	isArray () Determines if this Class object represents an array class.
boolean	isEnum () Returns true if and only if this class was declared as an enum in the source code.
boolean	isInterface () Determines if the specified Class object represents an interface type.
boolean	isPrimitive () Determines if the specified Class object represents a primitive type.
boolean	isSynthetic () Returns true if this class is a synthetic class; returns false otherwise.
String	toString () Converts the object to a string.

instanceof vs getClass()

```
class Parent {  
}  
class Uncle {  
}  
class Child extends Parent {  
}  
class Child2 extends Parent {  
}  
class GrandChild extends Child {  
}  
public class InstOf {  
    public static void main (String[] args){  
        Parent p = new GrandChild();  
        System.out.println (p instanceof Parent);  
        System.out.println (p.getClass()==Parent.class);  
    }  
}
```


hashCode()

Возвращает хэш-код объекта.

Хэш-код – это результат преобразования входного массива данных произвольной длины в выходную битовую строку фиксированной длины. Алгоритм преобразования заранее известен.

```
public int hashCode()
```

```
public class hashCodeObj {  
    public static void main (String[] args){  
        Object a = new Object();  
        Object b = new Object();  
        System.out.println (a.hashCode());  
        System.out.println (b.hashCode());  
    }  
}
```

386164770 575708156

hashCode()

return $10 * x + 1 * y$

```
class Flight {
    int flightNumber;
    Flight (int flNum){
        flightNumber = flNum;
    public int hashCode()
    {
        return flightNumber;
    }
}

}}


public class HashCodeFlightNum {
    public static void main (String[] args){
        Flight a = new Flight(5000);
        Flight b = new Flight(5000);

        System.out.println (a.hashCode());
        System.out.println (b.hashCode());
    }
}
```

hashCode()

```
class Flight {  
    int flightNumber;  
    Flight (int flNum){  
        flightNumber = flNum;  
    }  
}
```

```
public int hashCode() {  
    final int prime = 31;  
    int result = 1;  
    result = prime * result +  
        flightNumber;  
    return result;  
}
```



```
public class HashCodeFlightNum {  
    public static void main (String[] args){  
        Flight a = new Flight(5000);  
        Flight b = new Flight(5000);  
  
        System.out.println (a.hashCode());  
        System.out.println (b.hashCode());  
    }  
}
```

equals()

Предназначен для проверки, являются ли объекты одинаковые по значению.

```
public boolean equals (Object obj) ;
```

Object не содержит полей, поэтому результат сравнения оператором == и методом equals объектов класса Object будет одинаковый:

```
public class ObjComparison {  
    public static void main (String[] args){  
        Object a = new Object();  
        Object b = new Object();  
  
        System.out.println(a == b);  
        System.out.println(a.equals(b));  
    }  
}
```

equals()

```
class Flight {  
    int flightNumber;  
    Flight (int flNum){  
        flightNumber = flNum;  
  
    public boolean equals(Object obj){  
        return ((Flight)obj).flightNumber == flightNumber;  
    }  
}
```


```
public class FlightComparison {  
    public static void main (String[] args){  
        Object a = new Flight(100);  
        Flight b = new Flight(100);  
  
        System.out.println(a == b);  
        System.out.println(a.equals(b));  
    }  
}
```

Переопределяем метод **equals**:

equals()

```
class Flight {  
    int flightNumber; Flight (int  
    flNum){  
        flightNumber = flNum;  
    }  
}
```

```
public boolean equals(Object obj) {  
    if (this == obj)  
        return true;  
    if (obj == null)  
        return false;  
    if (getClass() != obj.getClass())  
        return false;  
    Flight other = (Flight) obj;  
    if (flightNumber != other.flightNumber)  
        return false;  
    return true;  
}
```



```
public class FlightComparison {  
    public static void main (String[] args){  
        Object a = new Flight(100);  
        Flight b = new Flight(100);  
  
        System.out.println(a == b);  
        System.out.println(a.equals(b));  
    }  
}
```

Требования к реализации equals() и hashCode()

Для поведения equals() и hashCode() существуют некоторые ограничения, которые указаны в документации по Object. В частности, метод equals() должен обладать следующими свойствами:

- Симметричность: Для двух ссылок, a и b, a.equals(b) тогда и только тогда, когда b.equals(a)
- Рефлексивность: Для всех ненулевых ссылок, a.equals(a)
- Транзитивность: Если a.equals(b) и b.equals(c), то тогда a.equals(c)
- Совместимость с hashCode(): Два тождественно равных объекта должны иметь одно и то же значение hashCode

toString()

Возвращает текстовое представление объекта. Создавая новый класс, данный метод можно переопределить и возвращать желаемое описание объекта.

```
public String toString()
```

Для объектов класса Object возвращается следующее выражение:

```
getClass().getName() + '@' + Integer.toHexString(hashCode())
```

```
package p1;
```

```
public class toStringObj {  
    public static void main (String[] args){  
        Object ob = new Object();  
        Flight fl = new Flight(400);  
  
        System.out.println(ob.toString());  
        System.out.println(fl.toString());  
    }  
}
```

```
java.lang.Object@2012a961  
p1.Flight@1af
```


toString()

Чтобы реализовать свое описание объекта, необходимо переопределить метод toString() в создаваемом классе.

```
class Flight {  
    int flightNumber;  
    Flight (int flNum){  
        flightNumber = flNum;  
    }  
}
```

?

```
}  
public class toStringObj {  
    public static void main (String[] args){  
        Object ob = new Object();  
        Flight fl = new Flight(400);  
  
        System.out.println(ob.toString());  
        System.out.println(fl.toString());  
    }  
}
```

java.lang.Object@292e2fba flight number is 400

toString()

Чтобы реализовать свое описание объекта, необходимо переопределить метод toString() в создаваемом классе.

```
class Flight {
    int flightNumber;
    Flight (int flNum){
        flightNumber = flNum;
    }

    public String toString (){
        return "flight number is " + flightNumber;
    }
}

public class toStringObj {
    public static void main (String[] args){
        Object ob = new Object();
        Flight fl = new Flight(400);

        System.out.println(ob.toString());
        System.out.println(fl.toString());
    }
}
```

java.lang.Object@292e2fba
flight number is 400

clone()

Создает и возвращает копию объекта x, такую, что:

`x.clone() != x` - true

`x.clone().getClass() == x.getClass()` - true

`x.clone().equals(x)` - true

Клонирование

```
class Wheel {
    int d;
    Wheel (int di){
        d = di;
    }
}

public class Bike implements Cloneable{
    int wheelCount;
    Wheel w;
    Bike (int wCount, int wDiameter) {
        wheelCount = wCount;
        w = new Wheel(wDiameter);
    }

    public static void main (String[] arg) throws CloneNotSupportedException{
        Bike bike1 = new Bike(2, 26);
        Bike bike2 = (Bike) bike1.clone();

        bike1.wheelCount = 1;
        bike1.w.d = 29;

        System.out.println (bike2.wheelCount);
        System.out.println (bike2.w.d);
    }
}
```

Задача 1

1. Модифицировать пример с клонированием так, чтобы клонированный объект велосипеда обладал своим типом колес (своим объектом класса `Wheel`). Использовать для этого переопределение метода `clone(...)` класса `Object`.
2. Для проверки корректности решения сделать консольный вывод исходного и клонированного объектов. Доказать корректность решения
3. Также переопределить в соответствии с правилами (см. ранее) методы `equals()` и `hashCode()` для классов `Bike` и `Wheel`.

Задача 2

На рынке представлены разные виды велосипедов: *одноколесные, двухколесные и трехколесные*. Несмотря на наличие общих черт, у них есть различия. Например, *ремонт, использование и сборка* велосипедов отличаются, а *покупка* велосипедов – нет.

Написать программу, реализующую описанную предметную область. Реализовать наиболее подходящую на Ваш взгляд ОО модель.

Примечания:

- Main – метод должен создать все виды описанного транспорта и выполнить с ними все перечисленные операции.
- Ремонт велосипедов поручить отдельному классу
- Все операции можно заменить выводом на консоль соответствующего текста. Например, для операции ремонта двухколесного велосипеда можно вывести: «Ремонт двухколесного велосипеда»