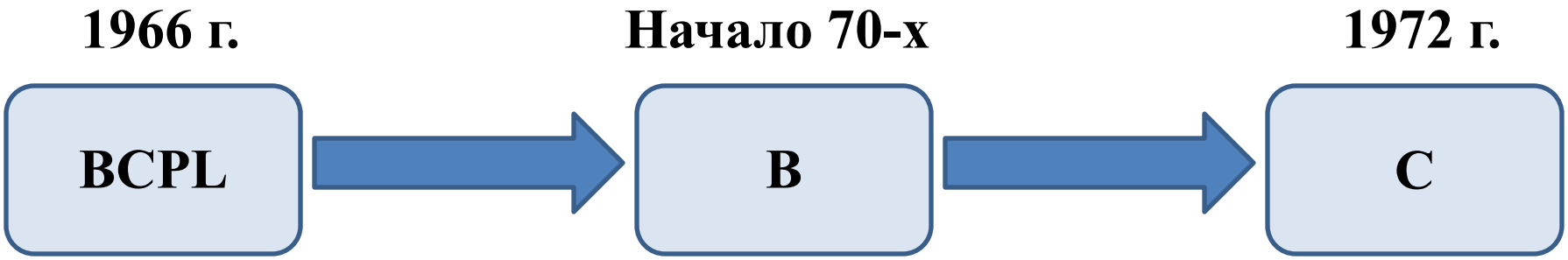
Лекция 1. Введение в Java

- История Java
- Виртуальная машина Java
- Основы ООП
- Ввод-вывод на примерах

История Java



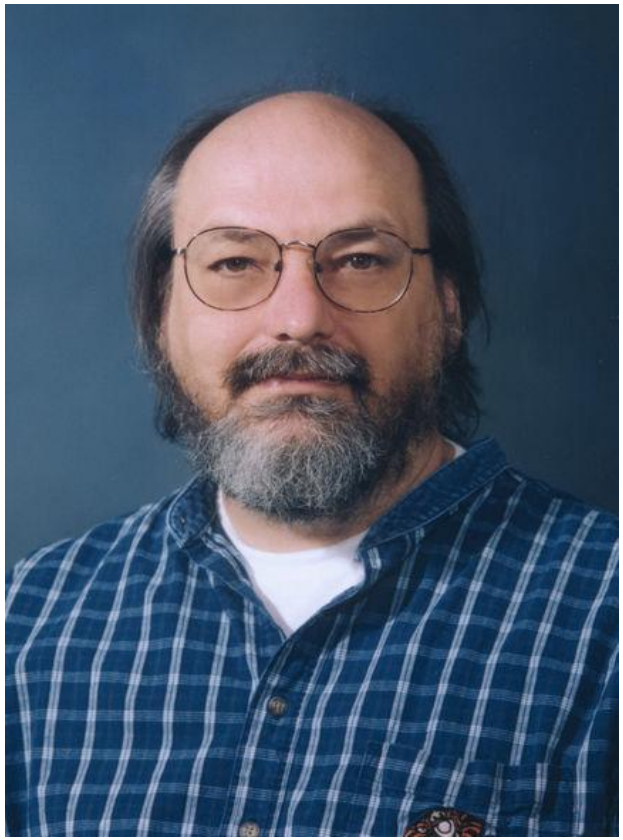
Мартином Ричардсоном был разработан язык BCPL, основное предназначение которого заключалось в написании компиляторов.

BCPL повлиял на язык, получивший название B, который был изобретен **Кеном Томпсоном** и в начале 70-х гг. привел к появлению языка C.

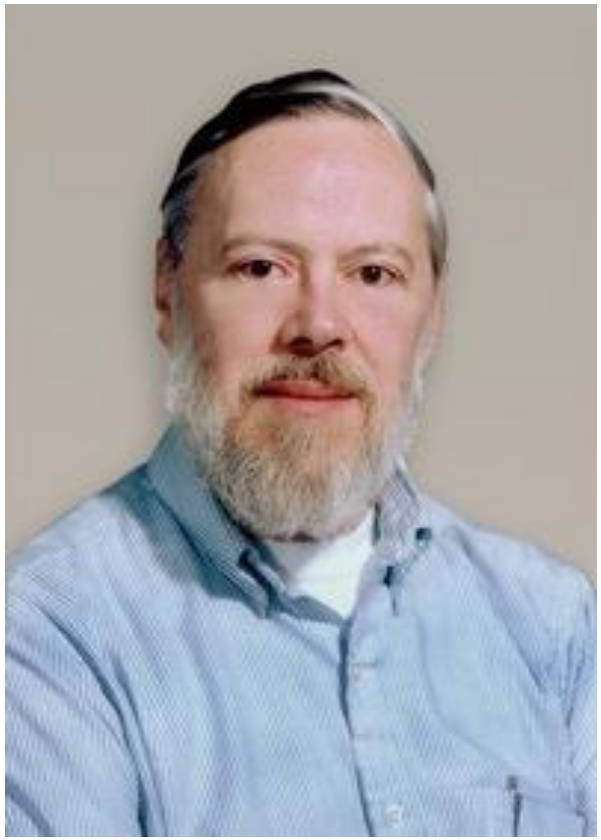
Язык C изобретенный и впервые реализованным **Деннисом Ритчи** на компьютере DEC PDP-11, работающем под управлением операционной системы UNIX.

История Java

Мартин Ричардсон



Кен Томпсон

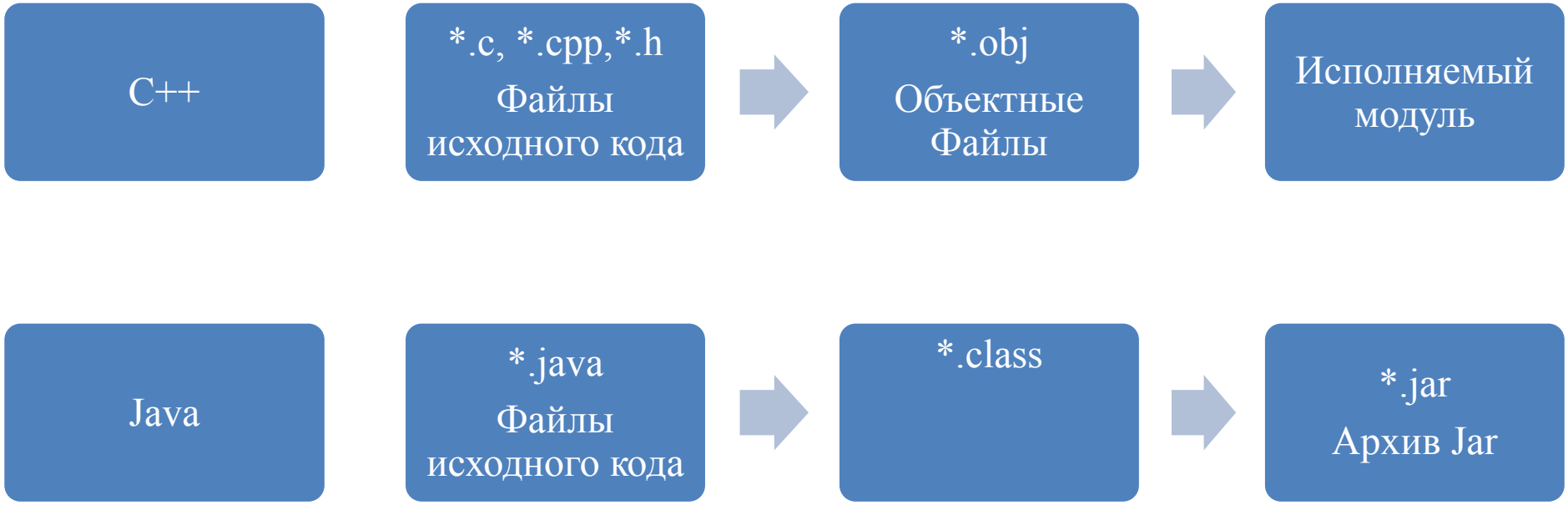


Деннис Ритчи

Отличия Java от C++

C++	Java
Переносимость на уровне исходного кода. Принцип: Write once, compile anywhere (WOCA)	Переносимость на уровне байт-кода. Принцип: Write once, run anywhere (WORA)
Разрабатывался для системного программирования	Разрабатывался для прикладного программирования бытовых электронных устройств
Возможно использование процедурного и ООП	Полностью объектно-ориентированный язык
Явное управление памятью, поддержка указателей	Отсутствие указателей, не допускается прямое обращение к памяти
Отсутствие контроля границ массивов	Контроль границ массивов
Программа сама управляет выделением и освобождением памяти	Автоматическая сборка мусора
Поддержка множественного наследования	Множественное наследование реализовано с помощью интерфейсов
Отсутствует стандартный механизм документирования исходного кода	Документирование с помощью JavaDoc
Поддержка оператора goto	Отсутствие оператора goto. Поддержка меток в циклах
Нестрогий контроль типов	Строгий контроль типов

Компиляция программ



Выполнение программ

C++

Исполняемый модуль

Имеет формат, поддерживаемый ОС, содержит инструкции архитектуры процессора

ОС

Java

Архив Jar или файл *.class

Содержит платформенно-независимый байт-код

JVM – виртуальная машина Java

Преобразует байт-код в инструкции целевого процессора во время выполнения

ОС

Виртуальная машина Java

Java Runtime Environment, сокращенно **JRE** – это исполнительная среда Java в которой выполняются программы. JRE является частью JDK. Это минимальная реализация виртуальной машины, необходимая для исполнения Javaприложений, без компилятора и других средств разработки.

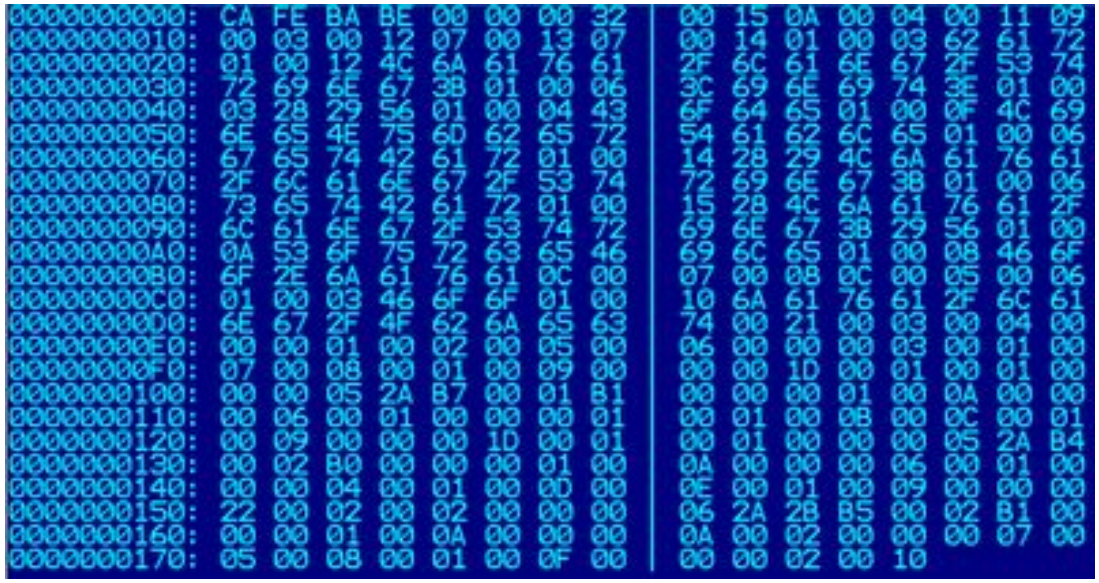
Java Virtual Machine, сокращенно **JVM** – это виртуальная машина Java — основная часть исполняющей среды **JRE**. Виртуальная машина Java интерпретирует и исполняет байт-код Java. Байт код получают посредством компиляции исходного кода программы с помощью компилятора Java. Библиотеки Java-классов входят в состав JRE.

Java Development Kit, **JDK** – это комплект разработчика приложений на языке Java, включающий в себя компилятор Java, стандартные библиотеки классов Java, примеры, документацию, различные утилиты и исполнительную систему Java (JRE).

Виртуальная машина Java

Код виртуальной машины

Код виртуальной машины (bytecode) – это в высшей степени оптимизированный набор инструкций, предназначенных для исполнения системой времени выполнения Java, называемой виртуальной машиной Java.



Трансляция программы Java в код виртуальной машины значительно упрощает ее выполнение в широком множестве сред, поскольку на каждой платформе необходимо реализовать только JVM.

Код Java и его Bytecode (мнемоническое представление)

```
Public class Foo {
        Private String
          bar;
        Public String
          getBar(){
          return bar;
        }
        Public void
          setBar(String
          bar){
          this.bar=bar;
        }
}
```

```
public class Foo extends java.lang.Object{
public Foo();
Code:
0:aload_0
1:invokespeciel
4: return
public java.lang.String getbar();
Code: 0: aload_0
1:getfield
4: areturn
public void setBar(java.lang.String);
Code:
0: alomd_0
1: aload_1
2: putfield
5: return
```

Типы данных

Примитивные типы Java не являются объектами. К ним относятся:

boolean - булев тип, может иметь значения true или false
byte - 8-разрядное целое число
short - 16-разрядное целое число
int - 32-разрядное целое число
long - 64-разрядное целое число
char - 16-разрядное беззнаковое целое, представляющее собой символ UTF-16 (буквы и цифры)
float - 32-разрядное число в формате IEEE 754 с плавающей точкой
double - 64-разрядное число в формате IEEE 754 с плавающей точкой

Типы данных

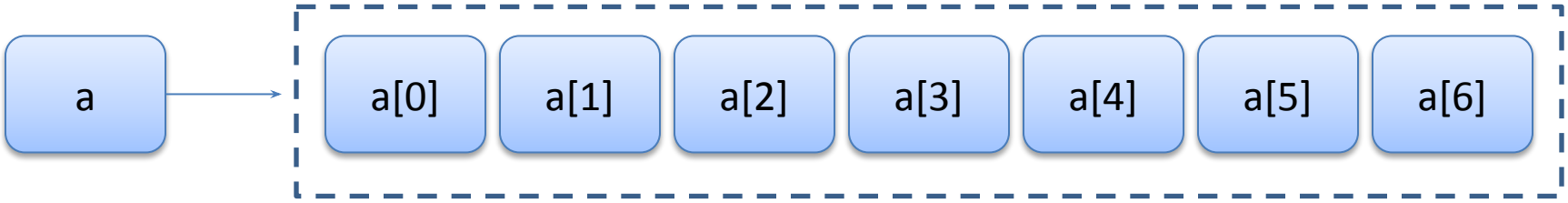
Тип	Оболочечный класс	Ширина в битах	Область допустимых значений
long	Long	64	-92337203685477580 8… 9223372036854775807
int	Integer	32	-2147483648… 2147483647
short	Short	16	-32768… 32767
byte	Byte	8	-128… 127

Массивы

Массив – группа однотипных переменных, обращение к которым выполняется по общему имени. Доступ к элементу массива осуществляется по его индексу. Java допускает создание массивов любого типа. Массивы в Java могут иметь одно или более измерений. Массивы в Java являются объектным типом данных.

Объявление массива:

Тип имя_переменной[]; или **тип[] имя_переменной;**



Размещение данных в массиве

Многомерные массивы

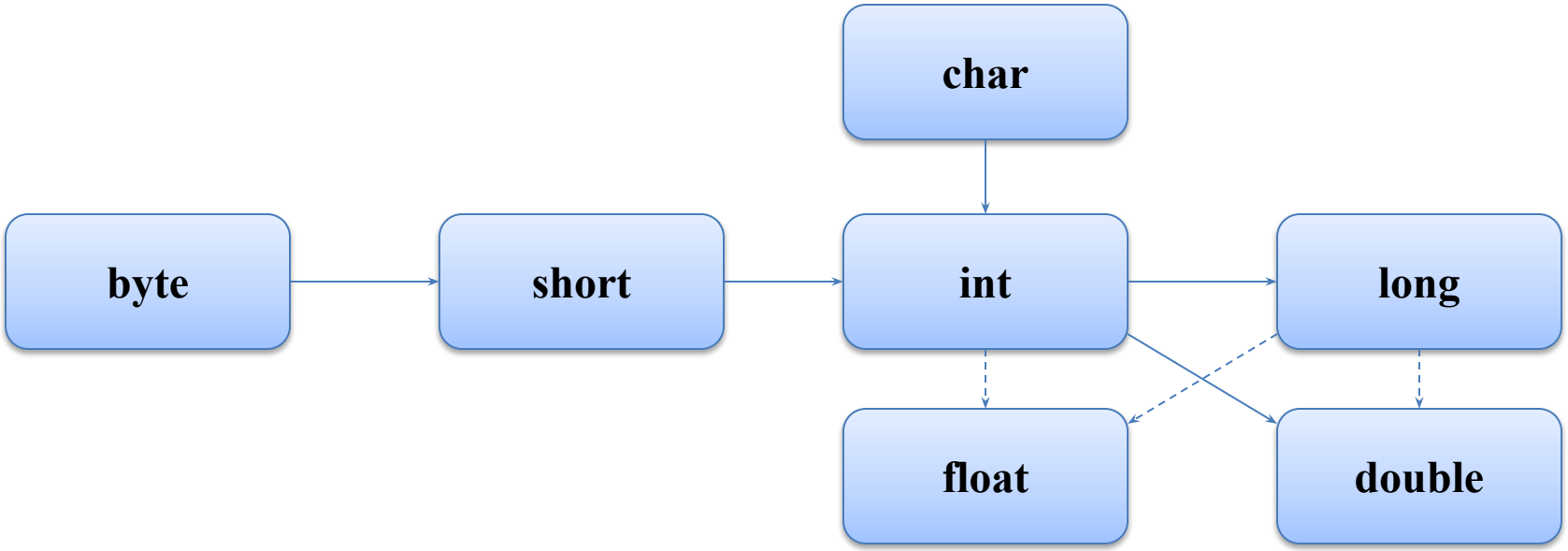
```
public class Matrix {
    public static void main (String[] args) {
        int[][] matrixB = {
            {-9, 1, 0},
            {4, 1, 1}.
            {-2, 2, -1}
};
for (int i = 0; i < 3; I ++) {
    for ( int j = 0; j < 3; j++) {
        System.out.print(matrixB[i][j]);
}

System.out.println();
}
}
}
```

Многомерные несимметричные массивы

```
int[][] a = new int[5][5];// двумерный массив
int[][][] b = new int[3][4][5];// трехмерный массив
int[][][][] c = new int[2][4][5][5];// четырехмерный массив
int[][] a1 = new int[5][];// двумерный массив с 5 строками

 a1[0] = new int [1];
 a1[1] = new int [2];
 a1[2] = new int [3];
 a1[3] = new int [4];
 a1[4] = new int [5];
for(int i = 0; i<a1.length; i++){
    for(int j = 0; j<a1[i].length; j++){
          System.out.print(a1[i][j] + " ");
                    }
    System.out.println();
}
```

Приведение типов

Когда мы производим какие-то действия с переменными, то нужно следить за типами. Нельзя умножать котов на футбольные мячи, это противоречит здравому смыслу. Также и с переменными. Если вы присваиваете переменной одного типа значение другого типа, то вспоминайте теорию. Например, вы без проблем можете присвоить значение типа **int** переменной типа **long**.



Основные конструкции. Условный оператор if

if (условие) оператор; *// если условие истинно, то выполняется оператор*

```
int x = 18;
     if(x>18){
System.out.print("Да");
        }
       if (true) x++;
            else x--;
       if(x==18)
             x++;
```

Java

Основные конструкции. Оператор switch

Команду **switch** часто называют командой выбора. Выбор осуществляется в зависимости от целочисленного выражения. Форма команды выглядит так:

```
switch(ВыражениеДляСравнения) {
    case Совпадение1:
            команда;
            break;
case Совпадение2:
            команда;
            break;
default:
            оператор;
            break;
}
```

Основные конструкции. Оператор while и do-while

Форма цикла **while** следующая:

```
while(условие) {

 // тело цикла

}
```

```
int counter = 10;
do {
textViewInfo.append("Осталось " + counter + "
сек.\n");
counter--;
} while (counter > 0);
```

Основные конструкции. Цикл for

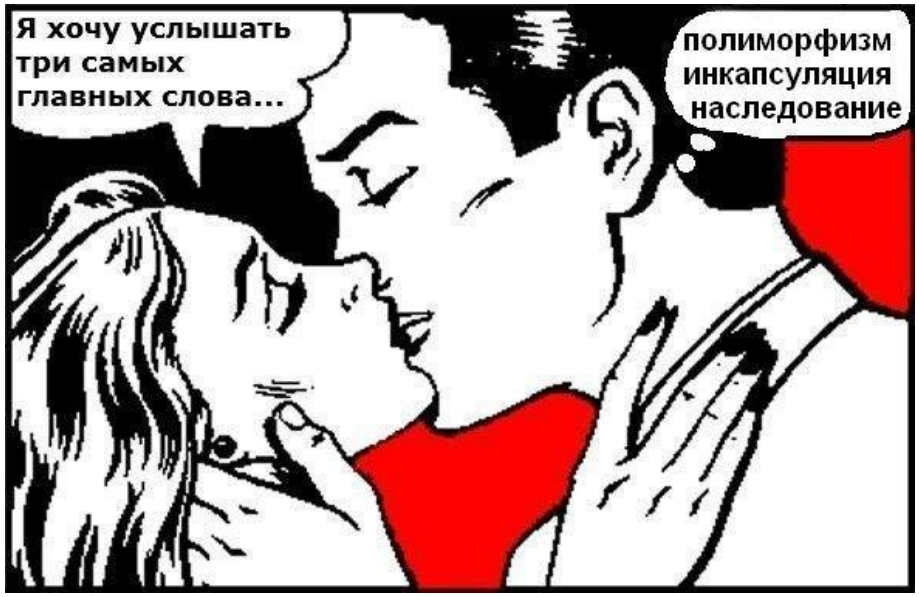
```
for (int kitten = 1; kitten < 10; kitten++) {

infoTextView.append("\nСчитаем котят: " + kitten);
resultEditText.setText("Ура! Нас подсчитали");

}

for (int y=0; y<100; y++) {
    System.out.println("Вывод №:" + y);
}
```

Java

Принципы ООП

Три принципа ООП:

- Инкапсуляция.
- Наследование
- Полиморфизм.



Java

Принципы ООП
Инкапсуляция

В Java основной инкапсуляции является класс. Класс определяет структуру и поведение, которые будут совместно использоваться набором объектов.

Принципы ООП

Наследование

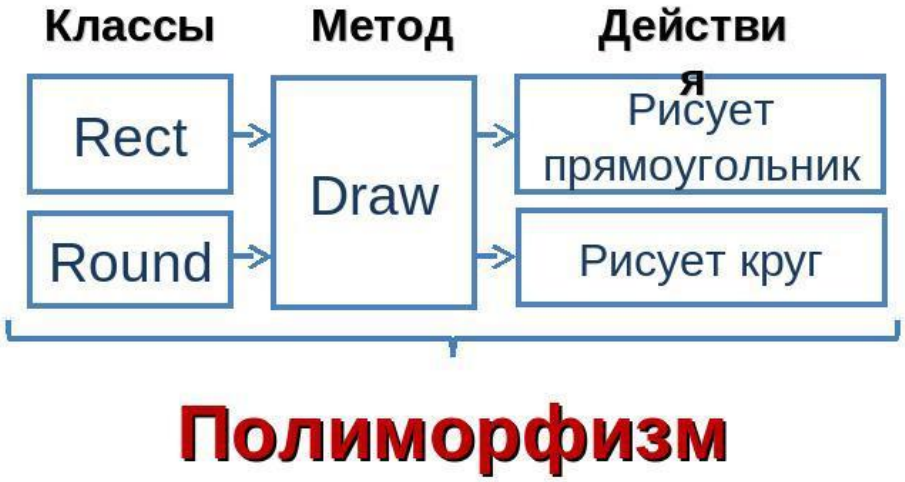
При наследовании один объект получает свойства другого, добавляя их к своим.

Java

Принципы ООП

Наследование

Полиморфизм, инструмент, позволяющий использовать один и тот же интерфейс (шаблон, форму) для общего класса действию.



Примеры принципов ООП. Инкапсуляция

```
public class Robot
{
    // Текущая координата X
    private double x = 0;
    // Текущая координата Y
    private double y = 0;
    // Текущий курс (в градусах)
    private double course = 0;

    // Передвижение на дистанцию distance
    public void forward(int distance) {
        // Обращение к полю объекта X
        x = x + distance * Math.cos(course / 180 * Math.PI);
        // Обращение к полю объекта Y
        y = y + distance * Math.sin(course / 180 * Math.PI);
    }
    // Печать координат робота
    public void printCoordinates() {
        System.out.println(x + "," + y);    }
```


Примеры принципов ООП. Инкапсуляция

```
    public double getX() {
        return x;
    }
    public double getY() {
        return y;
    }

    public double getCourse() {
        return course;
    }

public void setCourse(double course) {
        this.course = course;
    }
}
```

Примеры принципов ООП. Наследование

```
Class A{
   int i;}
//наследуемся от класса A
Class B extends A{
   int i; //имя переменной совпадает и скрывает переменную i в
   классе A
   B(int a, int b){
   Super.i=a; //обращаемся к переменной i из класса A
   i=b; //обращаемся к переменной i из класса B}
Void show() {
   System.out.println("i из суперкласса: " + super.i);
    System.out.println("i в подклассе: " + i);}
}
Class MainActivity{
   B subClass = new B(1,2);
   subclass.show();
}
```

Примеры принципов ООП. Полиморфизм.

```
class A {
          void m1(A  a) {
    System.out.print("A");
        }
}
class B extends A {
          void m1(B b)  {
    System.out.print("B");
        }
}
class C extends B {
          void m1(B c) {
    System.out.print("C");
 }}
class D {
    public static void main(String[]  args)  {
        A c1 = new C();
        c1.m1(new B());
} }
```

Классы

Классы могут наследовать свойства от других классов. Родительский класс называется суперклассом. Внутри классов могут быть объявлены поля и методы.

```
class ИмяКласса {
 тип переменная_экземпляра1;

тип имяМетода (список параметров){

 // тело метода

}
 }
```

Классы. Объекты

Новый объект (или экземпляр) создаётся из существующего класса при помощи ключевого слова **new**:

```
Cat barsik = new Cat(); // создали кота из класса
Cat
```

```
class Box {
 int width; // ширина коробки
 int height; // высота коробки
 int depth; // глубина коробки
}
Class Install{
Box setup = new Box();
setup.width = 250;
}
```

Java

Пример

```
Box bigbox = new Box(); // большая коробка
Box smallbox = new Box(); // маленькая коробка
int volume;
// присвоим значения переменным для большой коробки
bigbox.width = 400;
bigbox.height = 200;
bigbox.depth = 250;
// присвоим значения переменным для маленькой коробки
smallbox.width = 200;
smallbox.height = 100;
smallbox.depth = 150;
// вычисляем объём первой коробки
volume = bigbox.width * bigbox.height * bigbox.depth;
tvInfo.setText("Объём большой коробки: " + volume + "\n");
// вычисляем объём маленькой коробки
volume = smallbox.width * smallbox.height * smallbox.depth;
tvInfo.append("Объём маленькой коробки: " + volume);
```

Классы. Методы

- Метод может не иметь параметров, в этом случае используются пустые скобки.
- Методы могут вызывать другие методы.
- Каждый метод начинается со строки объявления, которую называют сигнатурой метода:

```
class Box {
int width; // ширина коробки
int height; // высота коробки
int depth; // глубина коробки

 // вычисляем объём коробки
String getVolume() {
return "Объём коробки: " + (width * height * depth);
}
}
```

Классы. Перегрузка методов

Синтаксис Java позволяет создавать в одном классе методы с одинаковыми именами, различающиеся только принимаемыми аргументами.

Пример:

```
public class Rectangle {
    private double width, height;
    public void setSize(int a) {
        width = a;
        height = a;
    }
    public void setSize(double a) {
        width = a;
        height = a;
    }
    public void setSize(double w, double h) {
        width = w;
        height = h;
    }
}
```

Классы. Наследование

У любого класса в Java может быть только один класс-прародитель. Он указывается с помощью зарезервированного слова **extends** после имени класса. Если класс-прародитель не указан, прародителем считается класс Object.

Пример:

```
class Point {
    // Тело класса
}
class Circle extends Point {
    // Тело класса
}
class Rectangle extends Point {
    // Тело класса
}
```

Классы. Переопределение методов

```
class Point {
    public double x, y;
    public double getSquare() {
        return 0;
    }
}
class Circle extends Point {
    public double r;
    public double getSquare() {
        return Math.PI * r * r;
    }
}
class Rectangle extends Point {
    public double width, height;
    public double getSquare() {
        return width * height;
    }
}
```

Классы. Зарезервированное слово super

```
class Point {
    public double x, y;
    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }
}
class Circle extends Point {
    public double r;
    public Circle(double x, double y, double r) {
        super(x, y);
        this.r = r;
    }
    public boolean inCircle(double x, double y) {
        return ( (super.x - x)*(super.x - x) +
            (super.y - y)*(super.y - y) < r*r);
}
```

Интерфейсы

Интерфейсы в Java предназначены для поддержки возможности множественного наследования.

Объявление интерфейса:

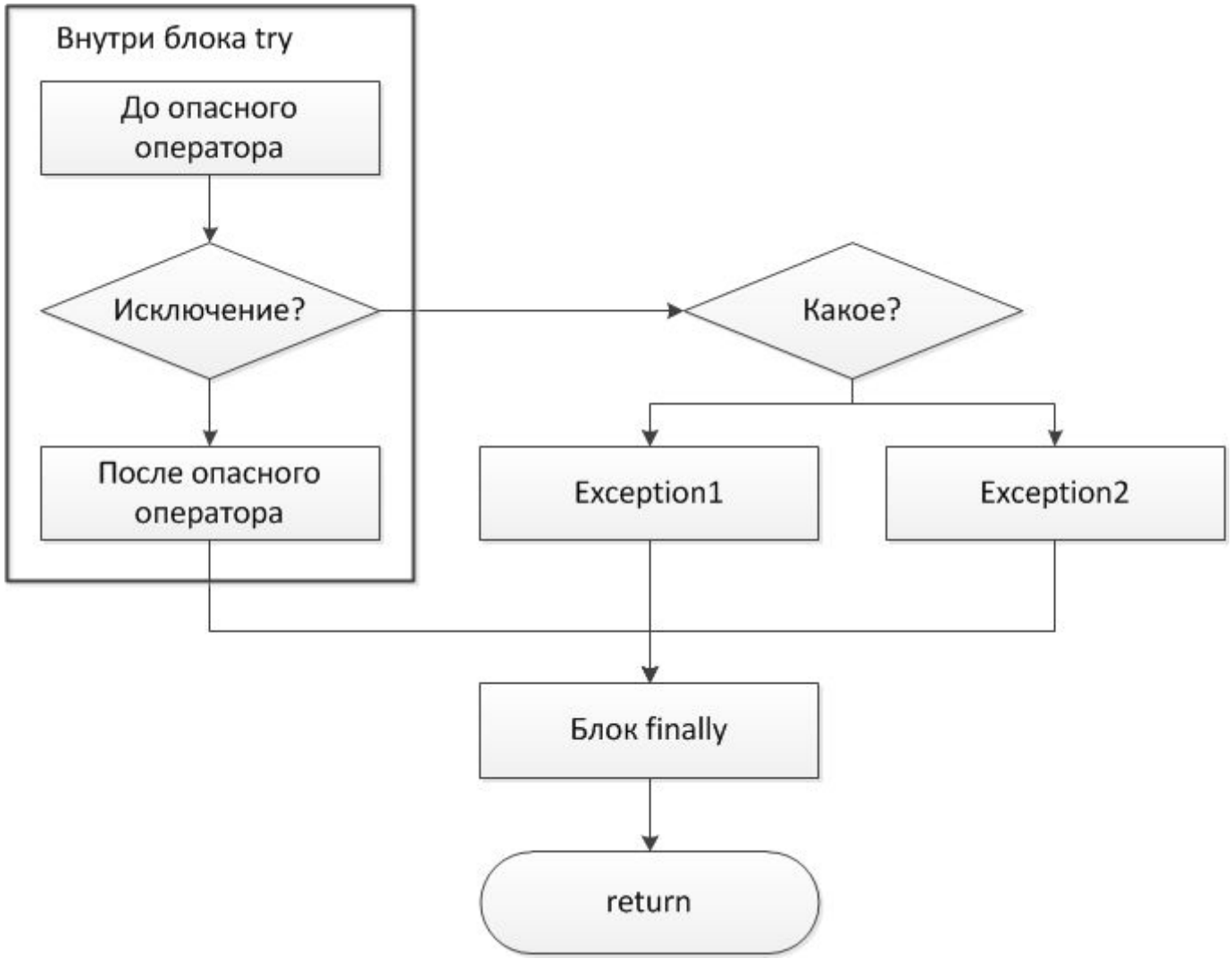
```
спецификатор_доступа interface Имя_интерфейса extends
Базовые_интерфейсы {
    спецификатор_доступа тип константа1 = значение1;
    спецификатор_доступа тип константа2 = значение2;
    /…
    спецификатор_доступа возвращаемый_тип
заголовок_метода1(аргументы);
    спецификатор_доступа возвращаемый_тип
заголовок_метода2(аргументы);
    /…
}
```

Исключения. Конструкция try-catch

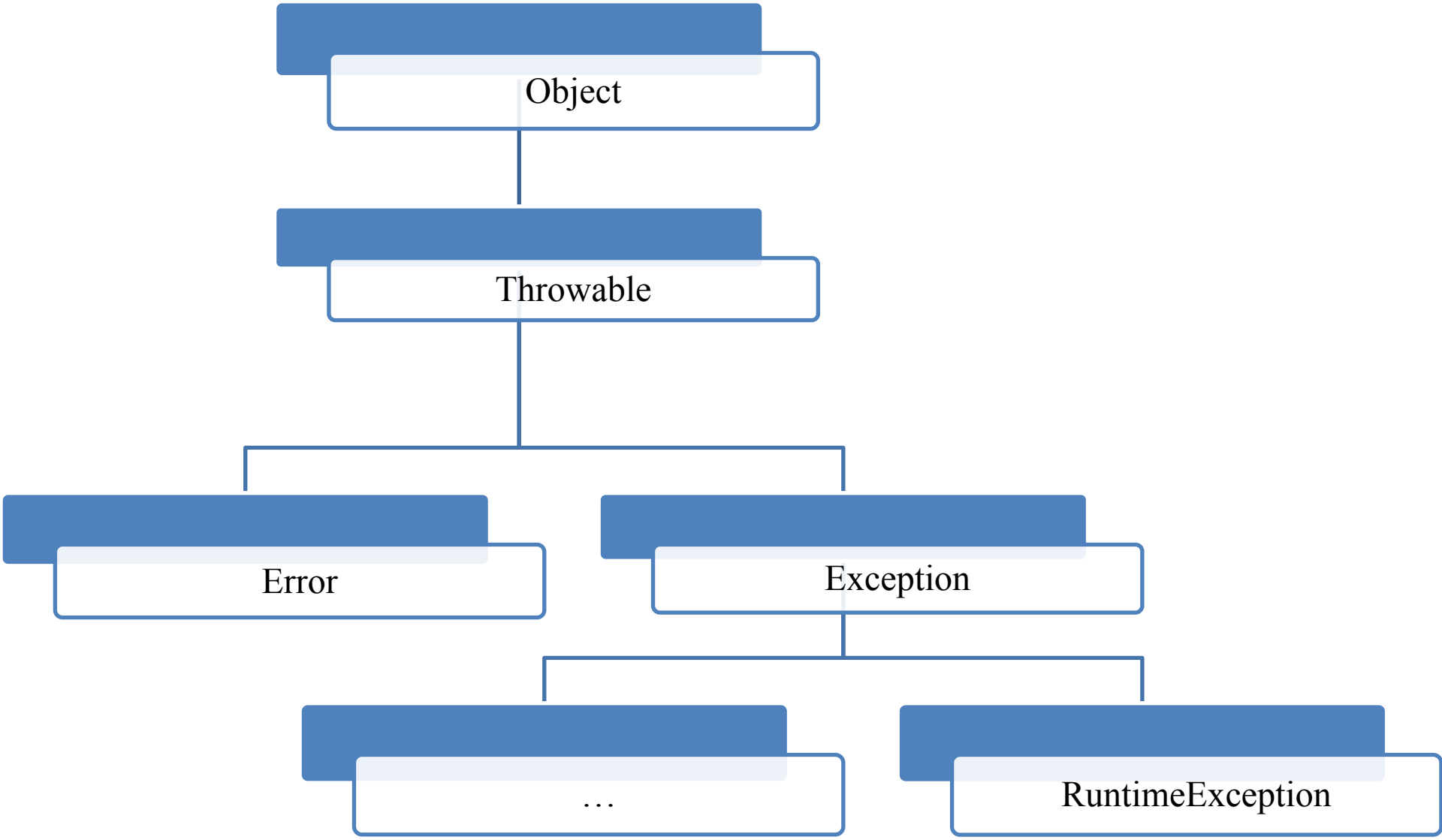
В Java предусмотрен механизм обработки исключений. Исключением называется ошибка времени выполнения программы. Исключения в Java реализованы в виде объектов, описывающих исключительную ситуацию. В случае возникновения ошибки времени выполнения, создаются объект-исключение и управление передаётся соответствующему этому объекту обработчику исключений.

```
try {
    // здесь возможно возникновение исключения
} catch(тип_исключения1 переменная1) {
    // обработчик исключения типа тип_исключения1
} catch(тип_исключения2 переменная2) {
    // обработчик исключения типа тип_исключения2
}
finally {
// код, который выполняется в любом случае после
выполнения блока try или завершения обработки исключения
в блоке catch
}
```

Наглядное представление конструкции try-catch



Приоритет обработчиков исключений



Java

Приоритет обработчиков исключений

```
try {
    int a = 4 / 0; // деление на 0
} catch(Exception e) {
    System.out.println("Исключение: " + e);
} catch(ArithmeticException e) {
    // этот блок не выполнится, потому что класс
ArithmeticException является подклассом класса Exception
    System.out.println("Исключение: " + e);
} finally {
    System.out.println("Этот блок выполнится в любом
случае после завершения блока try или обработки
исключения");
}
```

Организация ввода-вывода

Ввод-вывод в Java организован при помощи потоков. Потоки реализуются как классы, являющиеся абстракцией устройства ввода-вывода. Классы, отвечающие за стандартный ввод-вывод в Java, находятся в пакете **java.io**. Чтобы использовать эти классы, необходимо импортировать пакет **java.io**:

import java.io.*;

или только необходимый класс, например для импорта класса **InputStream**:
import java.io.InputStream;

Базовые классы ввода-вывода

	Байтовый	Классы-переходники	Символьный
Ввод	InputStream	InputStreamReader	Reader
Вывод	OutputStream	OutputStreamWriter	Writer

Байтовый ввод-вывод. Класс InputStream

public abstract int read()	Читает следующий байт из потока ввода. Возвращает значение этого байта в диапазоне от 0 до 255 или -1, если достигнут конец потока.
public int read(byte[] b)	Последовательно читает байты из потока ввода и записывает их в массив b. Может быть прочитано от 0 до (b.length – 1) байт. Возвращает количество прочитанных байт или -1, если достигнут конец потока.
public int read(byte[] b, int off, int len)	Последовательно читает от 0 до len байт потока ввода и записывает их в массив b со смещением off. Возвращает количество прочитанных байт или -1, если достигнут конец потока.
public long skip(long n)	Пропускает n байт потока ввода. Возвращает количество действительно пропущенных байт от 0 до n.
public int available()	Возвращает количество байт потока ввода, которые могут быть прочитаны или пропущены в данный момент.
public void close()	Закрывает поток ввода и освобождает связанные с ним системные ресурсы.

Бинарные данные. Класс OutputStream

Методы класса OutputStream	
public abstract void write(int b)	Записывает байт b в поток вывода. Записываются только младшие 8 бит аргумента b.
public void write(byte[] b)	Записывает последовательно все элементы массива b в поток вывода.
public void write(byte[] b, int off, int len)	Записывает len элементов массива b, начиная с off, в поток вывода.
public void flush()	Сбрасывает данные на физический носитель.
public void close()	Закрывает поток вывода и освобождает связанные с ним системные ресурсы.

Символьный ввод-вывод

public int read()	Читает следующий Unicode-символ из потока. Возвращает значение прочитанного символа в диапазоне от \u0000 до \uFFFF или -1, если достигнут конец потока.
public int read(CharBuffer target)	Читает символы из потока ввода в буфер класса CharBuffer. Возвращает количество прочитанных символов или -1, если достигнут конец потока.
public int read(char[] cbuf)	Последовательно читает символы из потока ввода и записывает их в массив cbuf. Может быть прочитано от 0 до (cbuf.length – 1) символов. Возвращает количество прочитанных символов или -1, если достигнут конец потока.
public abstract int read(char[] cbuf, int off, int len)	Последовательно читает от 0 до len символов потока ввода и записывает их в массив cbuf со смещением off. Возвращает количество прочитанных символов или -1, если достигнут конец потока.
public long skip(long n)	Пропускает n символов потока ввода. Возвращает количество действительно пропущенных символов от 0 до n.
public boolean ready()	Возвращает true, если в данный момент можно прочитать хоть один символ из потока ввода.
public abstract void close()	Закрывает поток ввода и освобождает связанные с ним системные ресурсы.

Пример. Запись файлов. Класс FileWriter.

```
import java.io.*;

public class FilesApp {
public static void main(String[] args) {

    try(FileWriter writer = new
Filewriter("C:\SomeDir\notes3.txt“);
{
    //запись всей строки
    String text = "Мама мыла раму, раму мыла мама";
    writer. write(text);
    //запись по символам
    writer. append('\n');
    writer. append('E');
}
catch(IOException  ex){
System.out.println(ex.getMessage());
}
}
```

Объектно-ориентированное программирование

Потоки

Поток - это абстрактное значение источника или приёмника данных, которые способны обрабатывать информацию. Вы в реальности не видите, как действительно идёт обработка данных в устройствах ввода/вывода, так как это сложно и вам это не нужно. Это как с телевизором - вы не знаете, как сигнал из кабеля превращается в картинку на экране, но вполне можете переключаться между каналами через пульт.
Есть два типа потоков: байтовые и символьные. В некоторых ситуациях символьные потоки более эффективны, чем байтовые.

За ввод и вывод отвечают разные классы Java. Классы, производные от базовых классов **InputStream** или **Reader**, имеют методы с именами **read()** для чтения отдельных байтов или массива байтов (отвечают за ввод данных). Классы, производные от классов**OutputStream** или **Write**, имеют методы с именами **write()** для записи одиночных байтов или массива байтов (отвечают за вывод данных).

Объектно-ориентированное программирование
Потоки. Класс InputStream

Базовый класс **InputStream** представляет классы, которые получают данные из различных источников:

- массив байтов
- строка (String)
- файл
- канал (pipe): данные помещаются с одного конца и извлекаются с другого
- последовательность различных потоков, которые можно объединить в одном потоке
- другие источники (например, подключение к интернету)

Объектно-ориентированное программирование

Потоки. Класс InputStream. Методы класса

- int available() - возвращает количество байтов ввода, доступные в данный момент для чтения
- close() - закрывает источник ввода. Следующие попытки чтения передадут исключение IOException
- void mark(int readlimit) - помещает метку в текущую точку входного потока, которая остаётся корректной до тех пор, пока не будет прочитано **readlimint** байт
- boolean markSupported() - возвращает *true*, если методы **mark()** и **reset()** поддерживаются потоком
- int read() - возвращает целочисленное представление следующего доступного байта в потоке. При достижении конца файла возвращается значение -1

Объектно-ориентированное программирование

Потоки. Класс .BufferedInputStream

Буферизация ввода-вывода является удобным способом оптимизации производительности, позволяя заключить в оболочку любой поток класса **InputStream**.

У класса есть конструктор, где размер буфера устанавливается по умолчанию. Также можно использовать конструктор, где размер буфера устанавливается вручную. Рекомендуется использовать размеры буфера, кратные размеру страницы памяти, дисковому блоку и т.п. и может зависеть от принимающей операционной системы, объёма доступной памяти и конфигурации машины.

Объектно-ориентированное программирование
Пример

Чтение стандартного ввода:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
 public class filelist {
public static void main(String[] args) {
 // 1b. Чтение стандартного ввода:
BufferedReader stdin = new BufferedReader(new
InputStreamReader(System.in));
System.out.print("Enter a line:");
try {
System.out.println(stdin.readLine());
} catch (IOException ex) {
 System.out.println("Reading error");
 }
} }
```

Пример

Чтение файла по строкам:

```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
public class filelist {
public static void main(String[] args) {
 BufferedReader in;
String s;
StringBuffer s2;
try {
in = new BufferedReader(new
FileReader("G:\\programs\\java\\testGUI\\src\\testgui\\filelist.
java"));

}
}
}
```

Пример

Продолжение:

```
s2 = new StringBuffer();
while ((s = in.readLine()) != null) {
s2.append(s + "\n") ;
}
System.out.println(s2);
in.close();
} catch(FileNotFoundException ex) {
 System.out.println(ex);
} catch (IOException ex){
System.out.println(ex);
}
}
}
```

Ввод-вывод в Java

Консольный ввод-вывод. Пример кода в IDE NetBeans

```
1  import java.util.Scanner;
2
3  public class FirstApp {
4
5      public static void main(String[] args) {
6
7          Scanner in = new Scanner(System.in);
8          System.out.print("Введите имя: ");
9          String name = in.nextLine();
10         System.out.print("Введите возраст: ");
11         int age = in.nextInt();
12         System.out.println("Ваше имя: " + name + "    Ваш возраст: " + age);
13     }
14 }
```

Ввод-вывод в Java

Консольный ввод-вывод. Пример вывода в IDE NetBeans

Например, если бы мы запускали проект в NetBeans, то это выглядело бы так: