10. ОСТАТКИ СЛАДКИ

Лекции по информатике для студентов
первого курса Высшей школы ИТИС
2019 год

МИХАИЛ АБРАМСКИЙ
старший преподаватель
Высшая школа ИТИС КФУ

01. КОНСОЛЬ И ФАЙЛЫ

Ну-ка

cmd - java FirstClass

```
F:\praxis\601>javac FirstClass.java

F:\praxis\601>java FirstClass
100500_
```

«Что у него написано в программцах, моя прелесть?»

Правда?

- int x = scanner.nextInt();
- double z = -1 * scanner.nextDouble();
- String s = scanner.next();
- String s = scanner.nextLine()
- …

Классическое считывание с текстового источника

- Все упомянутые примеры будут работать!
- При привычном считывании мы вводим информацию на экран – а это символы (текст).
- При вызове операторов ввода происходит конвертация из строки в тот тип данных, который необходим.
  - Pascal: read
  - C, C++: scanf, cin
  - Java: Scanner действует так же.
  - Python: так не делает, там raw_input, а дальше сам конвертируй

Lazy

- Но чтобы 100 раз не вводить, нужны…
- Файлы!
- А единственная ли это причина для использования файлов?

Файлы

- Важная причина использования файлов как источников данных в программе – необходимость хранить данные между запусками
    - «ААААА, я не засейвился»
- Любое сохранение используется для этих целей!

Консоль и файлы. Общее

- Мы пока будем говорить о текстовых файлах.

- Т.е. информация туда
  - Вводится пользователем заранее
  - Поступает извне, но в текстовом виде

- Примеры текстовых файлов?
- Примеры нетекстовых файлов?

Консоль и файлы. Разное

- Консоль – интерактивна.
  - Ждет событий пользователя
    - ввод чередуется с обработкой данных.
    - конец ввода определяется пользователем в реальном времени.
      - » введи n, n чисел
      - » суммируй, пока не ввели 0
        - обратите внимания завершение в обоих случаях
- Файл – нет
  - Введен заранее.
    - значит размер конечен – значит можно говорить о том, что конец ввода уже определен заранее
    - логично, что это позволяет обрабатывать файл по особому

EOF

- В любом языке программирования – End Of File
- Пока файл не кончился, читаем данные
  - While file is not over, read data
  - While (!file.over()) { data = file.read(); }

Собираем все вместе. Класс File

- Класс в Java, собирающий понятие «файла» (и папки).

- `File f = new File("input.txt")`

  ! расширение
  ! абсолютный и относительный пути
  ! что мы напишем, не обязательно существует
  ! когда нам это все равно?
  ! когда нам это важно?

hasNext...

```
Scanner scanner =
      new Scanner(new File("input.txt"));

int s = 0;

do {
     int x = scanner.nextInt();
     s += x;
}
while (scanner.hasNextInt());
```

Собираем

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class SumOfFileClass {

    public static void main(String[] args)
            throws FileNotFoundException {

        Scanner scanner = new Scanner(new File("input.txt"));
        int s = 0;
        do {
            int x = scanner.nextInt();
            s += x;
        }
        while (scanner.hasNextInt());
        System.out.println(s);
        scanner.close();
    }
}
```

Почему не все файлы текстовые

- Взял текстовый редактор, написал туда 126
  - Что я сделал по факту?
  - Сколько занимает 126?

02. КВАНТОРЫ И СОСТОЯНИЯ

Жзн

Вы хотите купить себе велосипед.

В городе 100 точек, где продаются велосипеды, удобные вам.

Куда вы пойдете?

Суть ответов

- Куда бы вы ни пошли, как бы ни обходили магазины, **вы остановитесь**, когда в очередном магазине его купите!
  - Из 100 магазинов это может быть и второй, и третий, и сотый магазин,
  - ВАЖНО: после покупки вы все равно остановитесь – в следующий магазин не пойдете.

Переносимся на урок

- Дан массив, проверить в нем наличие элемента, равного 0.

```
for (int i = 0; i < a.length; i++) {
    if (a[i] == 0) {
        System.out.println(a[i]);
    }
}
System.out.println("НЕТ!");
```

Формулировочка

- Нас не спрашивают
  - Ни сколько таких элементов
  - Ни какой индекс у нуля

- Нас спрашивают
  - Есть или нет.

! Предикат

Условие поиска – всегда предикат

- P(x) – x – нулевой
- P(x) – x – велосипед, который мне понравился

Но предикат отражает только условие над одним объектом.

- *А мы ведь ищем нулевой «среди элементов массива»*
- *Подходящий велосипед среди множества магазинов.*

Нужно уметь формулировать предикатное утверждение над множеством данных

Кванторы

- Не просто сокращенная запись «для любого» или «существует»
- Но оператор в математической логике
- $\forall x\ P(x)$, $\exists x\ P(x)$
  - для любого x верно P(x)
  - существует x, что для него верно P(x)

Кстати

- Не нужно писать *if (p == true)*
  - Если p - true, то (p == true) – тоже true
  - Если p - false, то (p == true) – тоже false
  - Значит, p и p == true – одно и то же выражение
- Аналогично if (p == false)
  - p - true – значит (p == false) – false
  - p - false – значит (p == false) – true
  - Значит p == false – это p наоборот
    - а p наоборот – это !p

Кванторов не два, а один.

- «Верно, что существует такое x, что P(x)»
- «Не верно, что все x – не P(x)»

- и наоборот

Итак

- «Ладно, я понял, я должен вернуть булевское значение – это и есть ответ квантора. Вот мой код»

```
boolean flag = false;
for (int i = 0; i < array.length; i++) {
    if (array[i] == 0) {
        flag = true;
    }
}
```

Долго. Нужно прерывание

! Смысл переменной flag – состояние «найдено». В начале оно – false
Если бы был квантор А – то начальное значение – true – “пока не опровергли, верно)

Что такое квантор по факту

$\forall i\ P(a[i]) = P(a[0])\ \&\ P(a[1])\ \&\ldots\&\ P(a[n-1]$

$\exists\ i\ P(a[i]) = P(a[0])\ V\ P(a[1])\ V\ldots V\ P(a[n-1]$

Что мы знаем про И и ИЛИ на нескольких аргументах?

- Вот именно. Зачем лишний раз считать, если ответ уже понятен.
- Нужно прерывать обработку, если ответ понятен.

Прерывание. Способ 1. Классика

```
boolean flag = false;
for (int i = 0; i < array.length && !flag; i++) {
    if (array[i] == 0) {
        flag = true;
    }
}
```

Прерывание. Способ 2.

```
boolean flag = false;
for (int i = 0; i < array.length; i++) {
    if (array[i] == 0) {
        flag = true;
        break;
    }
}
```

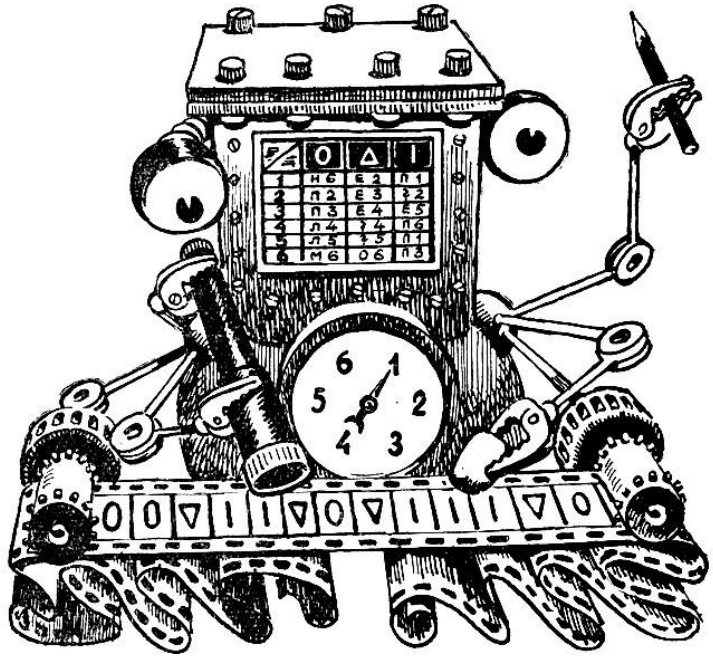
break и булева переменная дублируют функции друг друга.

Прерывание. Еще один способ

```
public static boolean checkArray(int [] array) {
    for (int x: array) {
        if (x == 0)
            return true;
    }
    return false;
}

public static void main(String[] args) {

    int array [] = new int[]{…};
    …
    System.out.println(checkArray(array));

}
```

Вспомним устройство МТ

- Алфавит
- Состояния (память)
- Лента (*бесконечная*)
- Считывающая головка
- Программа



	s1	s2
0	s1 →	1 stop
1	s1 →	0 s2 ←
^	s2 ←	1 stop

Это функция f(x) = x + 1

Казанский федеральный УНИВЕРСИТЕТ
ВЫСШАЯ ШКОЛА информационных технологий и информационных систем

Работа Машины Тьюринга

	s1	s2
0	s1 →	1 stop
1	s1 →	0 s2 ←
^	s2 ←	1 stop

Это функция f(x) = x + 1

Вход (аргумент функции, которую реализует МТ)
101 – двоичный код числа 5.

Выход, результат работы, значение функции f(x) = x + 1
110 – двоичный код числа 6.

			s1						
…	^	^	1	0	1	^	^	…	
				s1					
…	^	^	1	0	1	^	^	…	
					s1				
…	^	^	1	0	1	^	^	….	
						s1			
…	^	^	1	0	1	^	^	…	
					s2				
…	^	^	1	0	1	^	^	…	
				s2					
…	^	^	1	0	0	^	^	…	
				stop					
…	^	^	1	1	0	^	^	…	

Что есть состояние?

- Память о проведенной работе, закодированная в число.

- Состояние в реальном мире – светофор:

Квантовые задачи

- Там тоже есть модель состояний.
- Состояний всего 2:
  - «нет/есть» для E
  - «все/не все» для A
- Поэтому состояние можно было хранить где?

Вот это поворот

Модель состояний применима для решения задач проверки входа, что он в некотором смысле «правильно написан»

- Фактически обобщение квантора
  - Существует ли ошибка в задаче или нет

Пример - идентификатор

```
int state = 0;
String s = scanner.next();
i = 0
while (state != 2 && i < s.length()) {
    switch (state) {
        case 0:
            if (s.charAt(i) - буква || s.charAt(i) == "_")
                state = 1;
            else
                state = 2;
            break;
        case 1:
            if (s.charAt(i) - буква или цифра || s.charAt(i) == "_")
                state = 1;
            else
                state = 2;
            break;
    }
    i++;
}
if (state == 2) - все плохо
else - все хорошо
```

Дисклеймер для знатоков

«Так регулярками ж можн!»

Регулярные выражения:

- решают только определенный класс задач:
  - С помощью регулярных выражений нельзя проверить, что во входной строке одинаковое количество нулей и единиц
  - А алгоритм вы легко и просто закодите.

03. ИСКЛЮЧЕНИЯ

Ой, ошибка

```
int a = 2;
int b = 0;
int c = a / b;
```

Не скрываем возможные ошибки

- Если не мы (разработчики) в них виноваты
- Но скрываем их от пользователей

Пример

```
class MyMath {
    public int fact(int n) {
        if (n < 0)
            System.out.println("Can't take it!");
            // и что return? 0? -1?
        if (n == 0) {
            return 1;
        } else {
            return n * fact(n - 1);
        }
    }
}
```

Exception

- Объект, означающий исключительную ситуацию.

- Генерируется, когда java-код не может выполниться так, как предписано синтаксически.

Правильный пример

```
class MyMath {
    public int fact(int n) throws Exception {

        if (n < 0)
            throw new Exception("Can't take it!");

        if (n == 0) {
            return 1;

        } else {
            return n * fact(n - 1);
        }
    }
}
```

Не работает (и не компилируется). Что надо сделать?

```
public static void main(String[] args)  {

    Scanner sc = new Scanner(System.in);
    int f = MyMath.fact(sc.nextInt());

}
```

2 варианта

- сказать «*Да, у нас тут исключение, используешь наш метод – обработай!*»

- сказать «*Ого, тут исключение, надо срочно его поймать и не выпускать!*»

Можно делать, пока учимся

```
public static void main(String[] args) throws Exception {

    Scanner sc = new Scanner(System.in);
    int f = MyMath.fact(sc.nextInt());

}
```

Грамотный подход

```
public static void main(String[] args) {

    Scanner sc = new Scanner(System.in);
    try {
        int f = MyMath.fact(sc.nextInt());
    } catch (Exception e) {
        // действия, когда исключение было
    }

}
```

Класс исключения здесь должен быть тем же, или выше (суперклассом) для исключения, которое генерируется в fact.

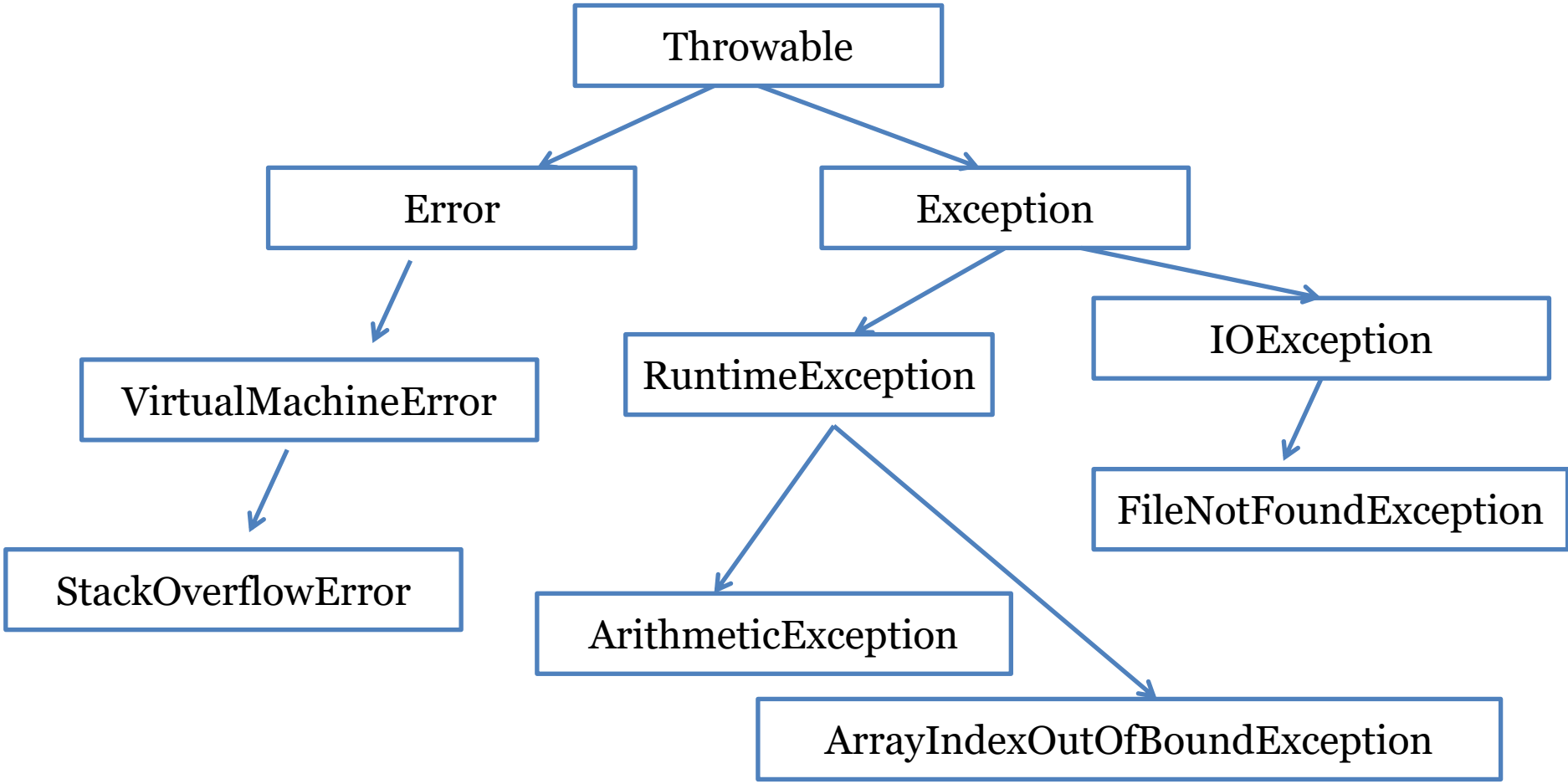
Ругается избирательно...

```
import java.io.File;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(new File("1.txt"));
        int b = 0;
        int c = 10 / b;
        int [] arr = new int[2];
        arr[2] = 10;
    }
}
```

Unhandled exception: java.io.FileNotFoundException

Почему?

Checked & Unchecked



```
public static void testIfPositive(int x) throws Exception {
    if (x < 0) {
        throw new Exception("Negative!");
    } else {
        System.out.println("x - positive");
    }
}

public static void main(String[] args) {
    try {
        Scanner sc = new Scanner(new File("1.txt"));
        int x = sc.nextInt();
        testIfPositive(x);
    } catch (Exception e) {
        System.out.println("So bad!");
    } catch (FileNotFoundException e){
        e.printStackTrace();
    }
```

Что не так?

Проблема

- FileNotFoundException – наследник Exception (не ясно, какое исключение сработало)

```
} catch (Exception e) {
        System.out.println("So bad!");
    } catch (FileNotFoundException e){
        e.printStackTrace();
    }
```

- Java не даст скомпилироваться этому коду

Правильно

```
catch (FileNotFoundException e){
        e.printStackTrace();
    }
catch (Exception e) {
        System.out.println("So bad!");
    }
```

finally

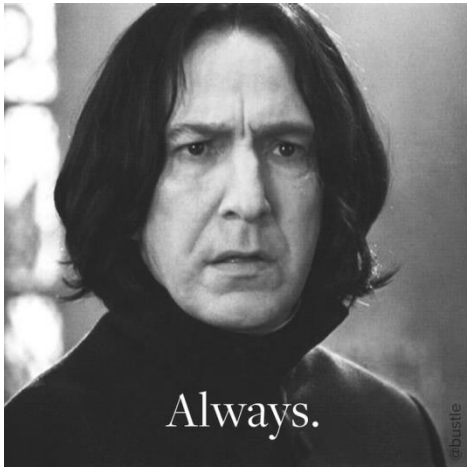
```
Scanner sc = new Scanner(new File("1.txt"));
int x = sc.nextInt();
try {
    testIfPositive(x);
} catch (Exception e) {
    System.out.println("So bad!");
} finally {
    sc.close();
}
```

Как выполняется finally:

finally

```
Scanner sc = new Scanner(new File("1.txt"));
int x = sc.nextInt();
try {
    testIfPositive(x);
} catch (Exception e) {
    System.out.println("So bad!");
} finally {
    sc.close();
}
```

Как выполняется finally:



Загадка

```
public class Main {
    public static boolean f() {
        try {
            return true;
        } finally {
            return false;
        }
    }
    public static void main(String[] args) {
        System.out.println(f());
    }
```

Что вернет метод f()?

4. ТАЙНА ПРИСВАИВАНИЙ

Тайна присваиваний

Так код работает:

```
int x, y, z;
x = y = z = 2;
```

не так важно, что он работает.

важно, почему он работает.

На самом деле

- Оператор присваивания возвращает значение
  - Если не понятно: он равен чему-то.
  - Да и понятно, в общем-то, чему.

- *Как вы думаете, чему равно (x = 2)?*

  = - это присваивание, а не сравнение.

Тайна присваиваний

- Значению присваиваемого выражения.
- Это и объясняет работу тайны #1:

```
x = (y = (z = 2));
```

Более того

- *Помните циклы со считыванием?*

```
x = scanner.nextInt()
while (x != 0) {
        …
        x = scanner.nextInt();
}
```

Дублирование раздражает и мы писали do while:

```
do {
        x = scanner.nextInt()
}
while (x != 0);
```

Ха! А можно обычным while устранить дублирование!

```
while ((x = scanner.nextInt()) != 0) {
        …
}
```

Вот только читаемость кода падает.

Поэтому не стоит злоупотреблять этой конструкцией.

5. МАССИВЫ КАК ШКАФЫ

Массив

- Попробуйте ответить на вопрос:
  - int [] array = new int[10].
  - сколько элементов в данном массиве после выполнения этой строчки?

Двоякое понимание

- Да, мы знаем, что массив сразу получает память под все свои 10 элементов.
  - Значит, у него 10 элементов
- Но с точки зрения наличия данных – мы же ничего не ввели еще в него!
  - Вы купили фотоальбом на 100 фотографий, но не положили пока туда ни одной фотографии. Сколько в фотоальбоме фотографий?

size и CAPACITY

- Второй случай – это использование массива как хранилища (контейнера, коллекции) данных.
  - Во втором семестре для этого будут использовать специальные классы.

- Но что, если нам надо реализовать самим? Тогда вводится понятие реального размера (size) и максимального размера (CAPACITY).
  - В этом смысле, при создании массива мы указываем CAPACITY, а size объявляем равным 0.
  - Почему пишу разным регистром?

Объявление

```
final int CAPACITY = …;
int array [] = new int[CAPACITY]{};
int size = 0;

// готовы к вставке элементов
```

Вставка в массив нового элемента (позиция не важна)

- Раз позиция не важна, то вставлять можно в конец.

- А конец – это?

Вставка в массив нового элемента (позиция не важна)

- Раз позиция не важна, то вставлять можно в конец.

- А конец – size.
- Вставим значение x:

```
array[size] = x;
size++;
```

!!!

Вставка в массив нового элемента (позиция важна)

- Просто так элементы не вставишь
  - Представьте стопку книг, стоящую подряд, и вы хотите вставить новую в центр
- Вставить x на позицию k
  - Отодвигаем элементы
  - Вставляем на освободившееся место

```
for (int j = size; j > k; j--) {
    array[j] = array[j-1];
}
array[k] = x;
```

Очистить массив

- Уже говорилось на прошлых парах: удаленные данные – это данные, которые назвали удаленными.
  - Никто их физически не стирал, но место, которое они занимают, объявлено свободным для записи (и их можно затереть)
    - Аналогия со пустым стулом, про который говорили, что он занят.
- Ну и как же очистить массив?

Молча

```
size = 0;
```

И вставка пойдет уже с нулевого элемента.