Занятие 3

Preface

1. Подпишитесь на ТГ канал с анонсами и объявлениями!
2. Те, кто дальше не проходят - могут смотреть лекции и делать ДЗ. Но проверки ДЗ не будет, как и шанса на стажировку
3. С MP, в котором домашка - делайте что хотите
4. Доработка ДЗ - по усмотрению преподавателя
5. Срок проверки - до следующей лекции. Людей, которых просто забыли - нет
6. Работы проверяются по очереди

Коллекции

Массивы

Массив - упорядоченная последовательность однотипных данных, объединенная одним именем. Доступ к конкретному элементу осуществляется по индексу за постоянное время

```
int[] a  =
```

8	12	5	9	31

```
a[0] == 8;
a[2] == 5;
a.length == 5
```

Массивы

Декларирование массива: `<тип>[] <имя>`
Тип массива может быть любой

Создание массива:

```
new <тип>[<длина>]
new <тип>[]{<элемент>...<элемент>}
```

Пример

```
int[] a = new int[5]; // массив целых чисел длиной 5
String[] s = new String[]{“a”,”b”,”c”}; // массив строк
```

Массив. Работа с элементами

У массива есть поле length, в котором хранится длина массива. Так что для массива можно использовать цикл for

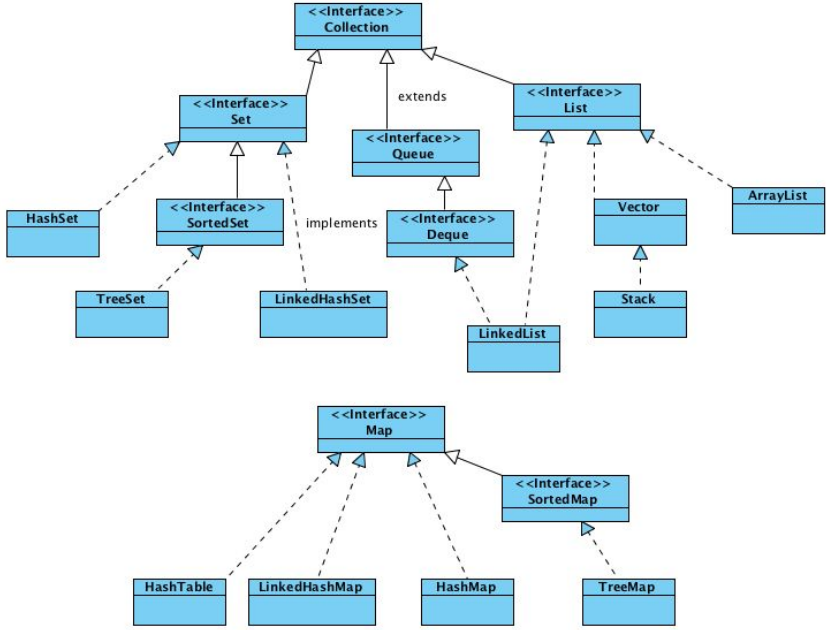
```
int[] a = new int[]{1,4,6,2};
for (int i=0; i < a.length; i++) {
  int elem = a[i];
  // код обработки
}
```

Коллекции

Основная проблема массивов - это фиксированный размер. Все, что можно делать с помощью массивов - хранить элементы по порядку и получать доступ по индексу.

В Java есть так называемые коллекции. Коллекции - это “контейнеры” для хранения объектов, обеспечивающие дополнительные свойства. Одно ограничение - коллекции могут хранить только объектные типы, не примитивные (в отличие от массивов)

Коллекции. Интерфейсы



Коллекции. Основные интерфейсы и классы

- `java.util.List`. Представляет собой упорядоченный список элементов. Ближайшая аналогия - это массив. Как и в массиве, допускаются одинаковые элементы. Самые популярные реализации: `java.util.ArrayList`, `java.util.LinkedList`.
- `java.util.Set`. Представляет собой “множество”. Порядок элементов не гарантируется, но гарантируется уникальность. Самые популярные реализации: `java.util.TreeSet`, `java.util.HashSet` .
- java.util.Map. Представляет собой ассоциативный массив. Хранит и возвращает элементы по ключу. Самые популярные реализации: `java.util.TreeMap`, `java.util.HashMap`

List

```
List<String> list = new ArrayList<>();
list.add(“A”);
list.add(“B”);
list.size(); // 2
list.get(0); // A

for (String element: list) {
  System.out.println(element);
}
```

Set

```
Set<String> set = new HashSet<>();
set.add(“A”);
set.add(“B”);
set.size(); // 2
set.add(“B”);
set.size(); // 2
for (String element: set) {
  System.out.println(element);
}
```

Live Coding Section

Написать программу, которая считывает слова в консоли, одно за одним, разделенные переносом строки (Enter). Окончание ввода - пустая строка.

Найти количество уникальных слов.

В каждом из слов найти количество уникальных символов.

Map. Ассоциативный массив

```
Map<String, String> map = new HashMap<>();
map.put("key1", "Value 1");
map.containsKey("key1"); // true
String val = map.get("key1"); // Value 1
map.put("key2", "Value 2");
map.size(); // 2
for (String key: map.keySet()) {
    String value = map.get(key); // обработка всех пар
}
```

Live Coding Section

Дана строка. Вычислить, сколько раз каждая буква встречается в строке
Ввод: AAABBACBA
Вывод: A-4 B-3 C-1

*Дан массив целых чисел a и число target. Найти пару индексов i и j таких, что a[i] + a[j] = target. Если такой пары нет - то вывести соответствующее сообщение

Обработка исключений

Исключительные ситуации

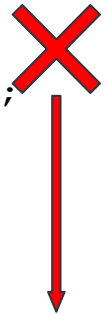
Часто бывает, что программа работает не так, как хочется или не так, как должна. В таких случаях, при возникновении ошибок, возникает так называемое “исключение”

- Деление на 0
- Обращение к несуществующему индексу массива
- Попытка обращения по null ссылке

Что происходит?

При возникновении ошибки создается объект класса-наследника java.lang.Exception

```
// код
...
int[] a = new int[1];
int b = a[15];
System.out.println(“a”);
...
```



`new java.lang.ArrayIndexOutOfBoundsException()`

Объект-исключение как бы “летит” над кодом, пропуская выполнение

Куда летит исключение?

Когда исключение “летит”, его надо “поймать”. Исключение летит до ближайшего блока **catch** с соответствующим типом. Если такого блока нет - выполнение возвращается из текущего метод в место вызова

```
void doSmth() {
  int[] a = new int[1];
  int b = a[15] + 1;
  System.out.println(b);
}
```

```
try {
  ...
  doSmth();
  ...
} catch (Exception ex){
  // что-то делаем
}
```

Куда летит исключение?-2

Если код выбрасывает исключение - **в некоторых** случаях надо это специально обозначить

Если исключение может вылететь из метода - это надо добавить в декларацию метода

```
public void doSmth()  throws
IOException{
 //
}
```

Если исключение возможно в блоке кода, соответствующий код надо обернуть в try-catch

```
try {
  // код, который может выбросить исключение
} catch (Exception ex) {
  // исключение прилетает сюда. тут можно
  // добавить логику обработки
}
```

Блок catch

Блоков catch может быть несколько, с разными типами ошибок. Управление передается в тот блок, который объявляет или класс выброшенного исключения, или класс-родитель выброшенного исключения

```
try {
  // код, который может выбросить исключение
} catch (FileNotFoundException fex) {
  // управление передается сюда, если выброшено
  // FileNotFoundException
} catch (SQLException ioex) {
  // управление передается сюда, если выброшено
  // SQLException
}
```

Если выброшенное исключение не соответствует ни одному типу в catch, то оно “полетит” дальше

Блок finally

Блок finally используется для кода, который должен быть выполнен в любом случае: при стандартном завершении и при исключении

```
try {
  // код, который может выбросить исключение
} catch (FileNotFoundException fex) {
  // управление передается сюда, если выброшено
  // FileNotFoundException
} finally {
  // управление передается сюда в любом случае
}
```

```
try {
  // код, который может выбросить исключение
} finally {
  // управление передается сюда в любом случае
}
```

Если выброшенное исключение не соответствует ни одному типу в catch, то оно “полетит” дальше

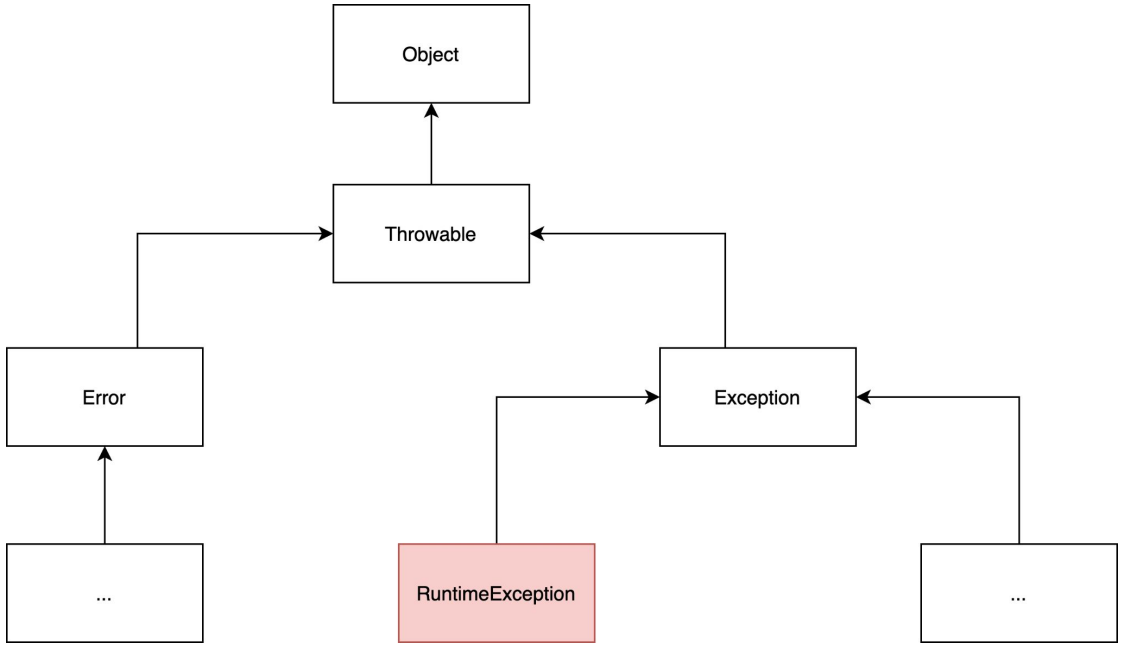
Checked/Unchecked исключения

В случае некоторых типов исключений, не обязательно декларировать, что код может их выбросить. Например, NullPointerException или ArrayOutOfBoundsException может быть выброшено вне зависимости от того, декларируются они или нет.

Такие исключения не обязательно декларировать, но можно обрабатывать как стандартные исключения

```
public boolean isNumber(String s) {
  try {
    Long.parseLong(s);
    return true;
  } catch (NumberFormatException npe) { }
  return false;
}
```

Классы исключений



Наследники RuntimeException - Unchecked исключения

Использование классов исключений

Исключение можно выбросить из кода, с помощью ключевого слова throw

```
try {
  throw new Exception();
}
...
```

Можно создавать собственные типы исключений, наследуя стандартные классы

```
class MyCustomException extends Exception

class WrongArgumentException extends RuntimeException
```

Не рекомендуется выбрасывать экземпляр Throwable и/или Error. Как правило Error - это фатальная ошибка, после которой работа невозможно, напр. OutOfMemoryError

Использование классов исключений

Исключение можно выбросить из кода, с помощью ключевого слова throw

```
try {
  throw new Exception();
}
...
```

Можно создавать собственные типы исключений, наследуя стандартные классы

```
class MyCustomException extends Exception

class WrongArgumentException extends RuntimeException
```

Не рекомендуется выбрасывать экземпляр Throwable и/или Error. Как правило Error - это фатальная ошибка, после которой работа невозможно, напр. OutOfMemoryError

IO

Java IO

Java IO - библиотека работы с вводом и выводом.

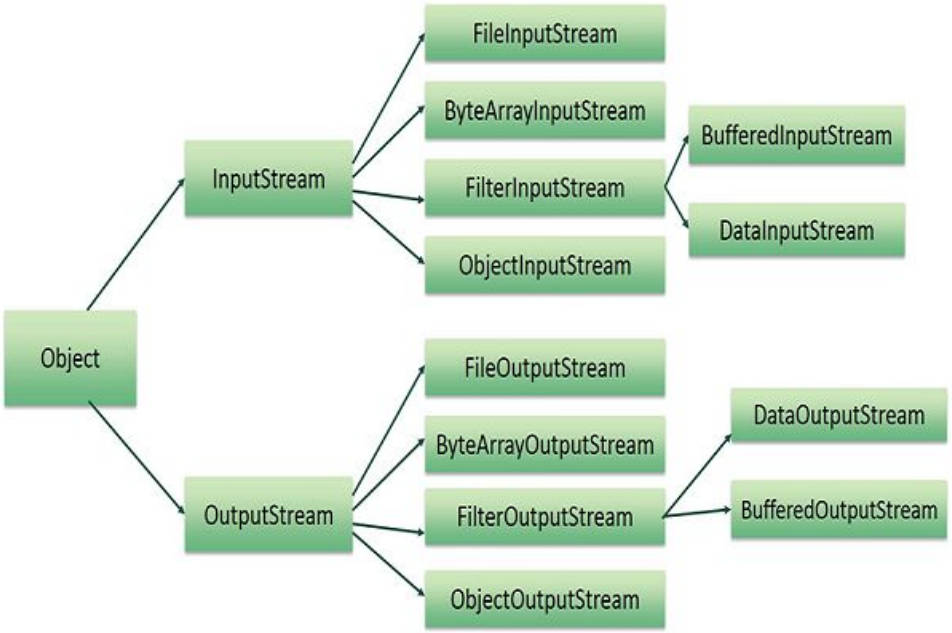
Ввод - чтение данных из внешнего источника (сети, файла и т.д.)

Вывод - запись данных во внешний источник (сеть, файл, т.д.)

Основные интерфейсы:

- OutputStream - имеет метод write для записи int
- InputStream - имеет метод read для записи int
- Reader/Writer - абстрактные классы для работы с символами

Java IO



`System.out` - это статическое поле `out` типа OutputStream у класса System

`System.in` - это статическое поле `in` типа InputStream у класса System

Использование потоков ввода-вывода

Основной подход в работе с потоками состоит в следующем:

1. Создать поток
2. Обработать
3. Вызвать flush();
4. Вызвать close();

Пункт 4 является обязательным, особенно если потоки связаны с файлами. Так что метод close() часто помещают в finally, чтобы он выполнился даже в случае ошибки.
Но есть и другой подход, который неявно помещает метод close в блок finally

```
try (FileInputStream fis = new FileInputStream(new File("1.txt"))) {
  int data = fis.read();
}
```

Чтение из файла

```
File file = new File("data.txt");
try (FileInputStream fileInputStream = new FileInputStream(file);
     Scanner scanner = new Scanner(fileInputStream)) {
     while (scanner.hasNextLine()) {
       String stringFromFile = scanner. nextLine();
       System.out.println(stringFromFile);
      }
}
```

Посмотреть как использовать BufferedReader для чтения из файла

Запись в файл

```
File file = new File("data.txt");
try (PrintWriter printWriter = new PrintWriter(file)){
    printWriter.println(“Hello, Ylab!”);
    printWriter.flush();
}
```

Live Coding Section

Продемонстрировать запись строк в файл и чтение строк из файла

Обзор домашнего задания