

java.lang (Java Language)
Package

Goals

- Get to know “*java.lang*” package and classes
- Get to know mechanism of exception handling

java.lang package

Основные классы пакета java.lang:

- *Object*
- *String*
- Wrapper Classes
- *Math,*
- *System,*
- *Runtime,*
- *Throwable/Exception/Error*
- . . .

java.lang.Object

В Java “почти” все данные является объектами и наследуют *Object*.

"Основные" методы класса *Object*:

- **boolean** `equals (Object)`
- **int** `hashCode ()`
- **String** `toString ()`

Д.з.: Изучить как реализованы эти методы в классах `java.lang.String`, `java.lang.Integer` и `java.util.ArrayList`.

Оператор *instanceof*

```
Object obj = new Object();
```

```
if (obj instanceof Object) {...}
```

```
...
```

```
String[] strArray = {};
```

```
if (strArray instanceof String[]) {...}
```

```
...
```

```
int[] intArray = {1, 2, 3};
```

```
if (intArray[0] instanceof Object) {...}
```

java.lang.String

String - контейнер для 16-bit Unicode-символов.

Все строковые литералы в Java (например "abc") являются объектами класса *String*.

Определение: `String str;`

Создание: `str = new String();`

```
str = new String("hi");
```

```
str = "hi";
```

```
str = "hi" + " there";
```

java.lang.String (продолжение)

Некоторые методы String:

- **int** `length()`
- **boolean** `equals(String)`
- **boolean** `startsWith(String)`
- **String** `toUpperCase()`
- **String** `toLowerCase()`
- **int** `indexOf(String)`
- **String** `substring(int)`

Wrapper Classes

- **Boolean**
- **Character**
- **Number**
- **Byte**
- **Short**
- **Integer**
- **Long**
- **Double**
- **Float**

Используются там, где нужны объекты. Содержат "полезные" поля и методы. Например:

- **MAX_VALUE, MIN_VALUE**
- **SIZE**
- **byte** `byteValue()`, **int** `intValue()`, ...
- **int** `parseInt(String)`, **long** `parseLong(String)`
- ...

String и Wrapper classes.

Особенности

- Перегруженные операции
- Передача в метод по значению

```
public static void summ(Integer i) {  
    i = i + new Integer(2);  
}
```

```
public static void main(String[] args) {  
    Integer i1 = 10;  
    summ(i1);  
    int i2 = 10;  
    summ(i2);  
    System.out.println("i1="+i1+";i2="+i2);  
}
```

java.lang.Math

Статические методы для операций над числами.

- **int** `abs(int)`, **long** `abs(long)`, ...
- **double** `cos(double)`, **double** `acos(double)`, ...
- **double** `exp(double)`, **double** `log(double)`, ...
- **int** `max(int, int)`, **long** `min(long, long)`, ...
- **double** `pow(double, double)`
- **double** `random()`
- ...

java.lang.System

Содержит "полезные" поля и методы.

Конструктор недоступен. Все поля и методы статические.

Класс предоставляет доступ к потокам *in*, *out*, *err*, методы доступа к *properties* и другие.

```
System.out.println("Hello World!");
```

```
System.currentTimeMillis();
```

```
System.exit(1);
```

```
System.getProperty("path.separator");
```

...

java.lang.Runtime

Позволяет получить доступ к "окружению" и запускать независимые процессы.

```
Process process = runtime.exec(...);
```

Во время выполнения в JVM есть только один экземпляр *Runtime*. Конструктор этого класса недоступен, но (в отличие от *System*) здесь нестатические методы.

Как можно получить доступ к ним?

java.lang.Runtime (продолжение)

Объект *Runtime* получают с помощью “static” метода `getRuntime()`.

* Это паттерн Синглтон о котором ещё будет отдельный разговор.

```
try {  
    Runtime.getRuntime().exec("notepad.exe");  
} catch (IOException e) {  
    e.printStackTrace();  
}
```

Что такое *try/catch*?

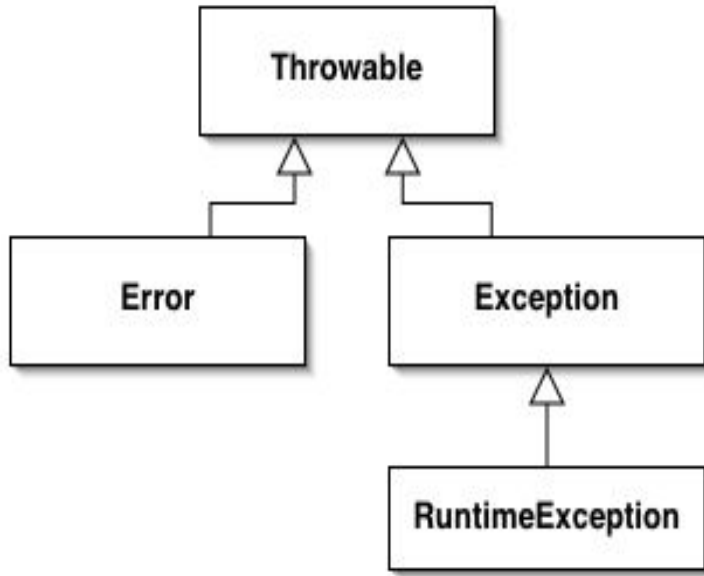
Exceptions: try catch final

Exception handling - способ обработки

"ИСКЛЮЧИТЕЛЬНЫХ" СИТУАЦИЙ ВО ВРЕМЯ ВЫПОЛНЕНИЯ программы.

```
try {  
    doSomething(); // code that may throw  
exception/error  
} catch (AnyThrowable1 e) {  
    ... // exception handler 1  
} catch (AnyThrowable2 | AnyThrowable3 e) {  
    ... // exception handler 2 and 3  
} finally {  
    ... // clean-up code  
}
```

Throwable, Exception, Error



RuntimeException - наследник *Exception*, зарезервированный для исключений связанных с некорректным использованием "стандартных" API.

Исключения, которые не являются наследниками *RuntimeException*, должны "отлавливаться".

Exceptions: throw throws

```
public class ThrowDemo {  
  
    static void demo1() {  
        throw new NullPointerException(); // RuntimeExceptions- OK  
    }  
  
    static void demo2() {  
        throw new IOException(); // compile error  
        // IOException must be caught or declared to be thrown  
    }  
  
    static void demo3() throws IOException {  
        throw new IOException();  
        // IOException is declared to be thrown - OK  
    }  
}
```


Домашнее задание

1. Реализовать программу для изменения регистра (если он в верхнем, то перевести в нижний, и наоборот) первого символа строки (параметр).
2. Реализовать класс *Cat* с атрибутами *int[3] rgbColor* и *int age*. Реализовать для него методы *toString* и *equals* (два кота идентичны, если у них одинаковый цвет и возраст). Подсказка: обратить внимание на метод *hashCode*.
3. Реализовать два класса-наследника *Exception* и *RuntimeException* и класс с двумя вызываемыми методами, каждый из которых будет бросать одно из этих исключений.