

1 Eclipse 简介和使用

IDE (Integrated Development Environment)：集成开发环境，集合开发、运行、调试于一体的一个软件

Eclipse 是一个开放源代码的、基于 Java 的可扩展开发平台。

下载地址：<http://www.eclipse.org/>

1.1 管理：

工作空间(workspace)、工程(project)

workspace

工作空间:代码保存在硬盘的空间建议按照班级号(20121224)

注意:工作空间命名不能选择带中文或者空格的例如(Program Files 就不可以)

Java Project

java 工程 (Java Project):Java Project 管理所有的 java 源程序和 class 文件

File -> new Java Project ->指定 Java 工程名 在学习中习惯按照时间命名(例如 day01)

1.2 使用

1. 创建 Java 工程

File --> new ----> java Project

2. 编程

可以使用使用快捷键,提高开发效率.

3. 运行

右键单击要运行的 java 文件----> run as ----> java application

或者是 ctrl+F11 快捷键。

1.3 常用快捷键

快捷键的配置，常用快捷键：

内容提示：

Alt + /

例如：System.out.println(); 语句 ,syso 再按住 alt 和/ 就会补全。

忘记某个类如何书写,可以写出一部分,按住 alt 和/ 就会有提示。

快速修复：

Ctrl + 1

例如,程序有编译期异常,或者需要导包.使用该快捷键.会有相关提示.

导包:

Ctrl + shift + O

如果需要导入的包比较多,可以一次性全部导入,也会将多余的包清理掉.

格式化代码块:

Ctrl + Shift + F

代码位置调换:

Alt+上下键

添加/除去单行注释

Ctrl+/
Ctrl+Shift+/
Ctrl+Shift+\

添加/除去多行注释

Ctrl+Shift+/
Ctrl+Shift+\

重置透视图: window->reset perspective

当 eclipse 的 Java 视图变的很乱的时候,就可以重置透视图,还原为最初的界面.

大小写转换

更改为大写 Ctrl+Shift+X

更改为小写 Ctrl+Shift+Y

复制行

Ctrl+Alt+向下键

查看源代码

1、Ctrl+单击 需要查看源码的类

2、Ctrl+Shift+T

删除:

1.Ctrl + D 删除当前行

1.4 自定义快捷键

注意: 快捷键的冲突:

例如: 输入法的简繁体切换 就是 ctrl+shift+f.

Windows Preferences 输入 key 点击 keys

例如设置 alt+/ 此时弹出对话框中输入 alt+/ 此时显示已被 Eclipse 默认设置,但是不是我们需要的.可以点击 Remove binding 来解除绑定

重新设置 输入 Content Assist (内容助理) binding 内输入 alt+/ 然后点击 apply

1.5 类的创建

(方法、构造函数、封装)

在类中鼠标右键点击 Source .可以选择创建 hashCode 方法,equals 方法, toString 方法.可以创建无参数构造,有参数构造.根据成员变量的 get set 方法.

方法的抽取

选中需要抽取为方法的代码块,鼠标右键 Refactor ->Extract method ->输入方法名.

变量的重命名

如果一个类中该变量重复较多,需要一个个修改,可以选中该变量名,然后 Refactor ->Rename ,同名的变量都会修改.

2 JDK5 特性

JDK5 中新增了很多新的 java 特性,利用这些新语法可以帮助开发人员编写出更加清晰,安全,高效的代码。

静态导入

自动装箱/拆箱

增强 for 循环

可变参数

枚举

泛型

2.1 静态导入（了解）

JDK 1.5 增加的静态导入语法用于导入类的某个静态属性或方法。使用静态导入可以简化程序对类静态属性和方法的调用。

语法:

import static 包名.类名.静态属性|静态方法|*

例如:

```
import static java.lang.System.out
```

```
import static java.lang.Math.*
```

```
import static java.lang.System.out;
import static java.lang.Math.*;

public class Demo {

    public static void main(String[] args) {
        // 普通写法
        System.out.println("hello world");
        int max = Math.max(100, 200);
        System.out.println(max);

        // 静态导入
        out.println("hello world");
    }
}
```

```
int max2 = max(100, 200);  
System.out.println(max2);  
}  
  
}
```

2.2 增强 for 循环

引入增强 for 循环的原因：在 JDK5 以前的版本中，遍历数组或集合中的元素，需先获得数组的长度或集合的迭代器，比较麻烦！

因此 JDK5 中定义了一种新的语法——增强 for 循环，以简化此类操作。**增强 for 循环只能用在数组、或实现 Iterable 接口的集合类上**

语法格式：

```
for (变量类型 变量 : 需迭代的数组或集合) {}
```

For each 是为了让你的代码变得简捷、和容易维护。

增强 for 循环要注意的细节：

1. 迭代器可以对遍历的元素进行操作，使用增强 for 循环时，不能对集合中的元素进行操作的。
2. 增加 for 循环与普通的 for 循环区别。
3. map 的遍历。

2.3 可变参数

JDK 中具有可变参数的类 Arrays.asList() 方法。

分别传多个参、传数组，传数组又传参的情况。

注意：传入基本数据类型数组的问题。

从 JDK 5 开始，Java 允许为方法定义长度可变的参数。

语法：数据类型...变量名。

可变长参数是 Object[] 数组。（可变参数里存的是对象数组）

JDK 中的典型应用：

Arrays.asList(T...a) 是 jdk 中的典型应用。

需求：对若干个整数进行求和

```
public static int sum1(int a,int b ) {  
    return a+b;  
}
```

若干个整数求和如何解决？

可以使用数组接收整数。

```
public static int sum1(int[] numbers) {  
    if (numbers == null) {  
        return 0;  
    }  
  
    if (numbers.length == 0) {  
        return 0;  
    }  
  
    int sum = 0;  
    for (int num : numbers) {  
        sum += num;  
    }  
    return sum;  
}
```

可以使用可变参数

```
public static int sum2(int... numbers) {  
    if (numbers == null) {  
        System.out.println("可变参数的值为null");  
        return 0;  
    }  
  
    if (numbers.length == 0) {  
        System.out.println("可变参数的值的长度为0");  
        return 0;  
    }  
  
    int sum = 0;  
    for (int num : numbers) {  
        sum += num;  
    }  
    return sum;  
}
```

可变参数的使用

```
public static void main(String[] args) {  
    // int result = sum1(new int[] { 1, 3, 5, 7, 9 });  
    // System.out.println(result);  
  
    // // 使用了可变参数, 传一个数组进去  
    // int result = sum2(new int[] { 1, 3, 5, 7, 9 });
```

```
// System.out.println(result);

// 使用了可变参数，不必声明数组，简化书写
// int result = sum2(2, 4, 6, 8, 10);
// int result = sum2(1);
int result = sum2();
System.out.println(result);
}
```

可变参数的细节

声明：

在一个方法中，最多只能有一个可变参数。
可变参数只能放在参数列表的最后面。

调用：

当使用可变参数时，可以传 0 或多个参数。
当使用可变参数时，也可以传一个数组进去，就表示多个参数。

使用：

在方法内部使用时，就是在使用一个数组。
当调用时没有传参数时（传了 0 个），这时在方法内部的参数数组是有值的（不为 null），但长度为 0。

2.4 自动装箱/拆箱

自动装箱：指开发人员可以把一个基本数据类型直接赋给对应的包装类。

自动拆箱：指开发人员可以把一个包装类对象直接赋给对应的基本数据类型。

典型应用：

```
List list = new ArrayList();
list.add(1);
//list.add(new Integer(1));
int i=list.get(0);
//int j = (Integer)list.get(0);
```

2.4.1 基本数据类型包装类

包装类

基本数据类型

Byte	byte
Short	short

Integer	int
Long	long
Boolean	boolean
Float	float
Double	double
Character	char
对象变基本数据类型:拆箱	基本数据类型包装为对象:装箱

为了使得 java 的基本类型有更多的功能, java 为其所有的基本类型提供了包装类来封装常见的功能。如: 最大值、数值转换等。

将基本数据类型封装成对象的好处在于可以在对象中定义更多的功能方法操作该数据所属的包: java.lang.*

常见应用一:

获取最大最小值 MAX_VALUE / MIN_VALUE

整数类型最大值

Integer.MAX_VALUE

```
System.out.println(Integer.MIN_VALUE); // -2147483648
System.out.println(Integer.MAX_VALUE); // 2147483647
```

应用二:

基本数据类型和字符串之间的转换

例: Integer 的 parseInt 方法, intValue 方法

基本数据类型转换成字符串:

1:基本数据类型+""

2:基本数据类型.toString(基本数据类型值);

例如 Integer.toString(34); //将 34 变成了"34"

基本数据类型转字符串

```
int i=100;
String str=100+"";
String string = Integer.toString(100);
```

字符串变基本数据类型

基本数据类型 a=基本数据类型包装类.parse 基本数据类型(String str);

```
str="123";
int parseInt = Integer.parseInt(str);
System.out.println(parseInt);
```

注意:

```
public static int parseInt(String s)
Integer 类中的 parseInt 方法是静态的 参数必须是数字格式
```

Double

```
str = "3.14";  
  
double parseInt2 = Double.parseDouble(str);  
System.out.println(parseInt2);  
  
boolean b = Boolean.parseBoolean("true");
```

应用三:

进制转换:

十进制转成其他进制.

toBinaryString(int i)

以二进制（基数 2）无符号整数形式返回一个整数参数的字符串表示形式。

toHexString(int i)

以十六进制（基数 16）无符号整数形式返回一个整数参数的字符串表示形式。

toOctalString(int i)

以八进制（基数 8）无符号整数形式返回一个整数参数的字符串表示形式。

那么其他进制转成十进制

parseInt(String radix);

parseInt(String s, int radix)

使用第二个参数指定的基数，将字符串参数解析为有符号的整数。

十进制转其他进制

```
// 十进制转二进制  
String binaryString = Integer.toBinaryString(100);  
System.out.println(binaryString); // 1100100  
// 十进制转十六进制  
String hexString = Integer.toHexString(100);  
System.out.println(hexString); // 64  
  
// 十进制转八进制  
String octalString = Integer.toOctalString(100);  
System.out.println(octalString); // 144
```

其他进制转十进制

```
// 字符串转对应的进制  
int parseInt3 = Integer.parseInt(octalString);  
System.out.println(parseInt3);  
// 二进制转十进制  
int parseInt4 = Integer.parseInt(binaryString, 2);  
System.out.println(parseInt4);  
// 十六进制转十进制
```



```
int parseInt5 = Integer.parseInt(hexString, 16);
System.out.println(parseInt5);
// 八进制转十进制

int parseInt6 = Integer.parseInt(octalString, 8);
System.out.println(parseInt6);
```

JDK5.0 后出现了自动装箱和拆箱

JDK5.0 以后，简化了定义方式。

```
Integer x = new Integer(5); // 装箱
int intValue = x.intValue(); // 拆箱

// 5.0 简化书写
// 自动装箱。new Integer(5);
Integer y = 5;
// 对象加整数, x 进行了自动拆箱, 变成了 int 型 和 5 进行加法运算后再将和进行
// 装箱赋给 x。
y = y + 5; // 是通过 Integer.intValue() 方法进行拆箱
```

练习：

```
public static void main(String[] args) {
    Integer a = 127;
    Integer b = 127;
    System.out.println(a == b);

    Integer c = 128;
    Integer d = 128;
    System.out.println(c == d);
}
```

请问结果？

a==b 为 true 因为 a 和 b 指向了同一个 Integer 对象。

Integer 的缓存大小 -128 ~ 127 之间也就是 byte 的范围。

2.5 枚举类

一些方法在运行时，它需要的数据不能是任意的，而必须是一定范围内的值，此类问题在 JDK5 以前采用自定义带有枚举功能的类解决，Java5 以后可以直接使用枚举予以解决。

例如：交通灯（红、黄、绿） 性别（男、女） 星期（星期一、二、三.....）

分数等级（A、B、C、D、E）

JDK 5 新增的 enum 关键字用于定义一个枚举类。

枚举的实现

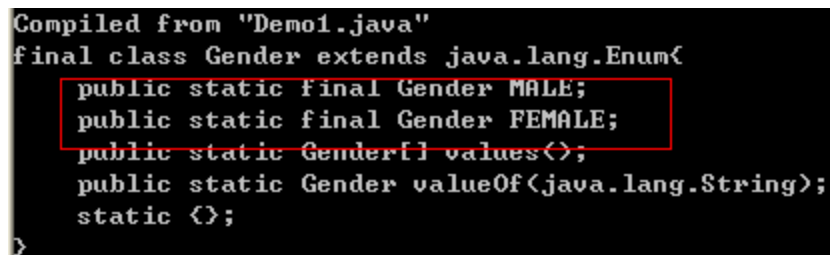
使用 enum 定义枚举类

在枚举类中定义枚举值（大写）

```
enum Gender {
```

```
    MALE, FEMALE;  
}
```

使用 javap 命令



```
Compiled from "Demo1.java"  
final class Gender extends java.lang.Enum<  
    public static final Gender MALE;  
    public static final Gender FEMALE;  
    public static Gender[] values();  
    public static Gender valueOf(java.lang.String);  
    static {};  
>
```

发现其中每一个枚举值都是枚举类的具体实例对象.只不过是静态常量.

枚举类具有如下特性:

枚举类也是一种特殊形式的 Java 类。

枚举类中声明的每一个枚举值代表枚举类的一个实例对象。

与 java 中的普通类一样,在声明枚举类时,也可以声明属性、方法和构造函数。

```
public class Demo1 {  
    public static void main(String[] args) {  
        Gender male = Gender.MALE;  
        System.out.println(male.getInfo());  
    }  
}  
  
enum Gender {  
    MALE("男"), FEMALE;  
  
    // 成员变量  
    private String info;  
  
    // 构造函数  
    private Gender() {  
  
    }  
  
    private Gender(String info) {  
        this.info = info;  
    }  
  
    // 成员方法  
    public String getInfo() {  
        return info;  
    }  
}
```

枚举类可以声明抽象方法,但是要有具体的枚举值去实现。

```
public class Demo1 {  
    public static void main(String[] args) {  
        Gender male = Gender.MALE;  
        System.out.println(male.getInfo());  
        male.speak();  
    }  
}  
  
enum Gender {  
    MALE("男") {  
        @Override  
        public void speak() {  
            System.out.println("是男人");  
        }  
    },  
    FEMALE {  
        @Override  
        public void speak() {  
            System.out.println("是女人");  
        }  
    };  
  
    // 成员变量  
    private String info;  
  
    // 构造函数  
    private Gender() {  
  
    }  
  
    private Gender(String info) {  
        this.info = info;  
    }  
  
    // 成员方法  
    public String getInfo() {  
        return info;  
    }  
  
    public abstract void speak();  
}
```

枚举类也可以实现接口(序列化)、或继承抽象类。

JDK5 中扩展了 switch 语句，它除了可以接收 int, byte, char, short 外，还可以接收一个枚举类型(enum)。

```
public class Demo2 {  
    public static void main(String[] args) {  
        WeekDay mon = WeekDay.MON;  
        switch (mon) {  
            case MON:  
                System.out.println("星期一要上班...");  
                break;  
            case TUE:  
                System.out.println("星期二,继续上班...");  
                break;  
        }  
    }  
}  
  
enum WeekDay {  
    MON, TUE, WED, THU, FRI, SAT, SUN;  
}
```

若枚举类只有一个枚举值，则可以当作单态设计模式使用。

练习：

请编写一个关于星期几的枚举 WeekDay，要求：枚举值：Mon, Tue, Wed, Thu, Fri, Sat, Sun 该枚举要有一个方法，调用该方法返回中文格式的星期。

```
enum WeekDay {  
    MON {  
        @Override  
        public String getInfo() {  
            return "星期一";  
        }  
    },  
    TUE {  
        @Override  
        public String getInfo() {  
            return "星期二";  
        }  
    },  
}
```

```
WED {
    @Override
    public String getInfo() {

        return "星期三";
    }
},
THU {
    @Override
    public String getInfo() {

        return "星期四";
    }
},
FRI {
    @Override
    public String getInfo() {

        return "星期五";
    }
},
SAT {
    @Override
    public String getInfo() {

        return "星期六";
    }
},
SUN {
    @Override
    public String getInfo() {

        return "星期天";
    }
};

public abstract String getInfo();
}
```

3 正则表达式

正则表达式：其实一种规则，有自己特殊的应用,其作用就是针对于字符串进行操作。

正则：就是用于操作字符串的规则,其中这些规则使用了一些字符表示。

3.1 快速体验正则表达式

需求：只能输入数字

```
public class Demo2{

    public static void main(String[] args) {
        //只能输入数字
        String str = "124354232";
        char[] arr = str.toCharArray();
        boolean flag = true;
        for(int i = 0 ; i< arr.length ; i++){
            if(!(arr[i]>=48&&arr[i]<=57)){
                flag = false;
            }
        }
        System.out.println(flag?"输入正确":"输出只能是数字");
    }

}
```

使用正则表达式：

```
public class Demo2{

    public static void main(String[] args) {
        //只能输入数字
        String str = "12435423a2";
        boolean flag = str.matches("[0-9]+");
        System.out.println(flag?"输入正确":"只能输入数字");
    }

}
```

3.2 正则表达式的符号

预定义字符类

.	任何字符（与行结束符可能匹配也可能不匹配）
\d	数字：[0-9]
\D	非数字：[^\d]
\s	空白字符：[\t\n\x0B\f\r]
\S	非空白字符：[^\s]
\w	单词字符：[a-zA-Z_0-9]
\W	非单词字符：[^\w]

```

System.out.println("a".matches("."));
System.out.println("1".matches("\\d"));
System.out.println("%".matches("\\D"));
System.out.println("\r".matches("\\s"));
System.out.println("^".matches("\\S"));
System.out.println("a".matches("\\w"));

```

Greedy 数量词

X?	X，一次或一次也没有
X*	X，零次或多次
X+	X，一次或多次
X{n}	X，恰好 n 次
X{n,}	X，至少 n 次
X{n,m}	X，至少 n 次，但是不超过 m 次

```

System.out.println("a".matches("."));
System.out.println("a".matches("a"));
System.out.println("a".matches("a?"));
System.out.println("aaa".matches("a*"));
System.out.println("".matches("a+"));
System.out.println("aaaaa".matches("a{5}"));
System.out.println("aaaaaaaaa".matches("a{5,8}"));
System.out.println("aaa".matches("a{5,}"));
System.out.println("aaaaab".matches("a{5,}"));

```

范围表示

[abc]	a、b 或 c（简单类）
[^abc]	任何字符，除了 a、b 或 c（否定）
[a-zA-Z]	a 到 z 或 A 到 Z，两头的字母包括在内（范围）
[a-d[m-p]]	a 到 d 或 m 到 p：[a-dm-p]（并集）
[a-z&&[def]]	d、e 或 f（交集）
[a-z&&[^bc]]	a 到 z，除了 b 和 c：[ad-z]（减去）

[a-z&&[^m-p]]	a 到 z, 而非 m 到 p: [a-lq-z] (减去)
<pre> System.out.println("a".matches("[a]")); System.out.println("aa".matches("[a]+")); System.out.println("abc".matches("[abc]{3,}")); System.out.println("abc".matches("[abc]+")); System.out.println("dshfshfu1".matches("[^abc]+")); System.out.println("abcdsaA".matches("[a-z]{5,}")); System.out.println("abcdsaA12".matches("[a-zA-Z]{5,}")); System.out.println("abcdsaA12".matches("[a-zA-Z0-9]{5,}")); System.out.println("abdxzyz".matches("[a-c[x-z]]+")); System.out.println("bcbcbc".matches("[a-z&&[b-c]]{5,}")); System.out.println("tretret".matches("[a-z&&[^b-c]]{5,}")); </pre>	

3.3 匹配功能

需求: 校验 QQ 号, 要求: 必须是 5~15 位数字, 0 不能开头。没有正则表达式之前

```

public static void checkQQ(String qq)
{
    int len = qq.length();
    if(len>=5 && len <=15)
    {
        if(!qq.startsWith("0"))
        {
            try
            {
                long l = Long.parseLong(qq);
                System.out.println("qq:"+l);
            }
            catch (NumberFormatException e)
            {
                System.out.println("出现非法字符");
            }
        }
        else
        {
            System.out.println("不可以0开头");
        }
    }
    else
    {
        System.out.println("QQ号长度错误");
    }
}

```


有了正则表达式之后：

`[1-9][0-9]{4,14}` `[1-9]`表示是第一位数字是会出现 1-9 范围之间的其中一个，下来的数字范围会出现在 0-9 之间，至少出现 4 次，最多出现 14 次。

```
public static void checkQQ2 ()
{
    String qq = "12345";
    String reg = "[1-9][0-9]{4,14}";
    boolean b = qq.matches(reg);
    System.out.println("b="+b);
}
```

需求：匹配是否为一个合法的手机号码。

```
public static void checkTel ()
{
    String tel = "25800001111";
    String reg = "1[35]\\d{9}"; //在字符串中，定义正则出现\ 要一对出现。
    boolean b = tel.matches(reg);
    System.out.println(tel+": "+b);
}
```

3.4 切割功能

需求 1：根据空格对一段字符串进行切割。

```
public static void splitDemo ()
{
    String str = "aa.bb.cc";
    str = "-1    99    4    23";
    String[] arr = str.split(" ");
    for (String s : arr)
    {
        System.out.println(s);
    }
}
```

需求 2：根据重叠词进行切割。

```
public static void splitDemo2()
{
    String str = "sdqqfgkkkhjppppkl";
    String[] arr = str.split("(.)\\1+");
    for(String s : arr)
    {
        System.out.println(s);
    }
}
```

注意：为了提高规则复用，用()进行封装，每一个括号都有一个编号，从1开始，为了复用这个规则。可以通过编号来完成该规则的调用。需要对编号数字进行转义。\\1 就代表获取1组规则。

3.5 替换功能

需求：把手机号替换成“*”号。

```
String str = "联系我：13567012119联系我：13567012119联系我：13567012119联系我：13567012119联系我：13567012119";
String reg= "1[34578]\\d{9}";
str = str.replaceAll(reg, "*****");
System.out.println("替换后的帖子: "+ str);
```

练习二：我我...我...我,要...要要...要学...学学..学.编..编编.编.程.程.程..程
将字符串还原成 我要学编程。

```
public static void showStr()
{
    String str = "我我....我...我,要...要要...要学....学学..学.编..编编.编.程.程.程..程";

    /*
    先将字符串中的 . 都去掉。
    将没有 . 的字符串进行叠词的处理。多个重叠的字保留一个。
    */
    // 将点去掉。
    str = str.replaceAll("\\.", "");
    System.out.println(str);

    str = str.replaceAll("(.)\\1+", "$1");
    System.out.println(str);
}
```

3.6 获取

获取需要使用到正则的两个对象：使用的是用正则对象 `Pattern` 和匹配器 `Matcher`。

用法：

范例：

```
Pattern p = Pattern.compile("a*b");
Matcher m = p.matcher("aaaaab");
boolean b = m.matches();
```

步骤：

- 1, 先将正则表达式编译成正则对象。使用的是 `Pattern` 类一个静态的方法 `compile(regex)`;
- 2, 让正则对象和要操作的字符串相关联, 通过 `matcher` 方法完成, 并返回匹配器对象。
- 3, 通过匹配器对象的方法将正则模式作用到字符串上对字符串进行针对性的功能操作

需求：获取由 3 个字母组成的单词。

```
public static void getDemo()
{
    String str = "da jia zhu yi le,ming tian bu fang jia,xie xie!";
    //想要获取由3个字母组成的单词。
    //刚才的功能返回的都是一个结果，只有split返回的是数组，但是它是把规则作为
    //分隔符，不会获取符合规则的内容。
    //这时我们要用到一些正则对象。
    String reg = "\\b[a-z]{3}\\b";
    Pattern p = Pattern.compile(reg);
    Matcher m = p.matcher(str);
    while(m.find())
    {
        System.out.println(m.start()+"...."+m.end());

        System.out.println("sub:"+str.substring(m.start(),m.end()));
        System.out.println(m.group());
    }
    //    System.out.println(m.find()); //将规则对字符串进行匹配查找。
    //    System.out.println(m.find()); //将规则对字符串进行匹配查找。
    //    System.out.println(m.group()); //在使用group方法之前，必须要先找，找
    //到了才可以取。
}
```

练习 3：校验邮件

```
public static void checkMail()
{
    String mail = "abc123@sina.com.cn";
    mail = "1@1.1";
    String reg = "[a-zA-Z_0-9]+@[a-zA-Z0-9]+(\\. [a-zA-Z]+)+";
    reg = "\\w+@\\w+(\\.\\w+)+"; //简化的规则。笼统的匹配。
    boolean b = mail.matches(reg);
    System.out.println(mail+":"+b);
}
```

练习 4：网络爬虫

```
class GetMailList
{
    public static void main(String[] args) throws Exception
    {
        String reg = "\\w+@[a-zA-Z]+(\\. [a-zA-Z]+)+";
        getMailsByWeb(reg);
    }

    public static void getMailsByWeb(String regex) throws Exception
    {
        URL url = new URL("http://localhost:8080/myweb/mail.html");

        URLConnection conn = url.openConnection();
        BufferedReader bufIn = new BufferedReader(new
InputStreamReader(conn.getInputStream()));
        String line = null;
        Pattern p = Pattern.compile(regex);
        while((line=bufIn.readLine())!=null)
        {
            //System.out.println(line);
            Matcher m = p.matcher(line);
            while(m.find())
            {
                System.out.println(m.group());
            }
        }
        bufIn.close();
    }

    public static void getMails(String regex) throws Exception
    {
        BufferedReader bufr =
            new BufferedReader(new FileReader("mail.txt"));
        String line = null;
    }
}
```

```
Pattern p = Pattern.compile(regex);
while((line=bufr.readLine())!=null)
{
    //System.out.println(line);
    Matcher m = p.matcher(line);
    while(m.find())
    {
        System.out.println(m.group());
    }
}
bufr.close();
}
```