Regular Expressions

Classes

. (dot) any character

\w word characters / letters A-Z a-z 0-9

\W non-word characters

\s white space characters empty, CR, TAB, LF

\S non-white-space characters

\d digits [0-9]

\D non-digits

[ ] Own class e.g. [aeiou] [A-Za-z]

() Alternatives e.g. (de|com|org)

Special characters

* Previous pattern repeated 0 to n times

+ Previous pattern repeated 1 to n times

? Previous pattern repeated 0 or 1 time

{n,m} Previous pattern repeated min n and max m times

{n,} Previous pattern repeated min n times

{,m} Previous pattern repeated max m times

\ Escape Character

Working with JUnit

1. File > new > Source Folder (name of new folder "test")
2. File > new > JUnit Test Case (name of class = name of class under tests + "Test" (e.g. EmailCheckTest"))
3. Implement test code (in our case just copy from assignment)

Assignment: implement the method "checkValidAdress"

Offer: submit 10 exercises for final exam

StartRegExp.java EmailCheck.java EmailCheckTest.java

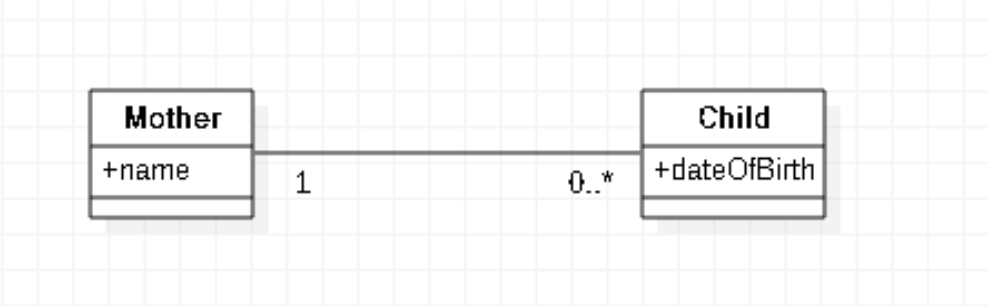
```java
package regexp;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

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

        String email = "ab@htwg-konstanz.de";

        //1. Naive approach
        System.out.println("The email is valid " + email.contains("@"));

        //2. Regular Expression
        System.out.println("The email is valid " + email.matches("[A-Za-z]{3,200}@.+\\.(de|com|org)"));

        //3. Using patterns (recommend if a huge amount of data needs to be processed)
        Pattern pattern = Pattern.compile("[A-Za-z]{3,200}@.+\\.(de|com|org)");
        Matcher matcher = pattern.matcher(email);
        System.out.println("The email is valid " + matcher.matches());

        //4. Search and replace. Example: +49-7531-206-xxxx needs be replaced by +49-7531-1234-xxx
        String data = "49-7531-206-206";
        data = data.replaceAll("(49-7531)-(206)-(\\d{3,4})", "$1-1234-$3");
        System.out.println("The new phone number is " + data);

    }
}
```

1 2 3

use replaceAll instead of replace()

$1 and $3 refer to the first and third part identified by the parentheses.

UML 2 Java Code



Different Options:

1. With arrays
    1. Disadvantage: fixed array size (that might be too little or to big)
    2. Disadvantage: search becomes slower with increasing size of the array
    3. Advantage: fast access to elements if the index (position) is known
2. With linked lists
    1. Advantage: Dynamic array size
    2. Disadvantage: search and inserts becomes slower with increasing size or length of the list
3. HashSets (s. next page)

HashSets

HashSets are Java classes that wrap Arrays (internally) and offer methods such as add, contains, delete.

Add-function (Insertion)

1. Ask object to be inserted for hashCode.
2. Check whether value at position of hashCode is null
3. If yes, insert object at this position (typical case)
4. If not, check whether the objects at this position is equals to the object to be inserted
    1. if yes: do nothing, object is already contained
    2. if no: Deal with colission. Two options
        1. Use a linear list ("sitting on the lap")
        2. Use a function to search for the next free spot ("Sondierungsfunktion", "exploratory function). Always check whether object there is equals

Search function

1. ask object to be searched for for its hashCode
2. Check whether value at position of hashCode is null
3. If yes, return false (object is not in the array / HashSet)
4. If no, check whether the object at this position is equals to the object to be searched for
    1. if yes: return true, object is found
    2. if no:
        1. Search in linear list
        2. Use exploratory function to search for object. Always check whether object there is equals. If exploratory function finds empty space return false, object is not in array / HashSet.

Mother.java StartClass.java Child.java

```java
package uml2codeWithHashSet;

import java.util.Date;

public class Child {
    private Date dateOfBirth;
    private String name;
    private Mother mother;

    public Child(String name, Mother m) {
        super();
        this.name = name;
        this.mother = m;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public Mother getMother() {
        return mother;
    }
    public void setMother(Mother mother) {
        this.mother = mother;
    }


}

```

Mother.java StartClass.java Child.java

```java
package uml2codeWithHashSet;

import java.util.HashSet;

public class Mother {
    private String name;
    private HashSet children = new HashSet();

    public void addChild(Child c) {
        children.add(c);
    }

    public boolean hasMotherThisChild(Child c) {
        return children.contains(c);
    }

    public int getNumberOfChildren(){
        return children.size();
    }

    public void abortion(Child c) {
        children.remove(c);
    }


}

```