文字列操作と正規表現

オブジェクト指向プログラミング特論

2018年度

只木進一：工学系研究科

文字列と文字列クラス

- 0個以上の長さの文字の列
  - JavaではStringクラス
- 操作
  - 文字列を作る・連結する
  - 文字列中に文字列を探す
  - 文字列中の文字列を置き換える
  - 部分文字列を得る

Stringクラス

- 文字列を保持するクラス
  - 文字列は定数であることに注意
- 比較に注意
  - “==“：オブジェクトとしての同等性
  - equals()：保持している文字列の同等性

```
String a = “abc”;
String b = “abc”;
if ( a == b ){//失敗
}
if ( a.equals(b) ){//成功
}
```

文字列を探す

- char charAt(int index)
  - indexの位置の文字
- int　indexOf(String str)
  - strが最初に現れる位置
- int indexOf(String str, int from)
  - from以降でstrが最初に現れる位置
- String substring(int begin, int end)
  - 部分文字列

文字列を作る・連結する

- Stringクラス中の文字列は定数
- Objectクラス
  - toString()メソッドで文字列化
- toString()メソッドを上書きすることで、各クラスに適切な文字列化を定義

StringBuilderクラス

- 文字になるものを連結するメソッド
- append()
  - 引数を文字列表現に変換(toString())して、末尾に連結
- delete()
  - 部分文字列を削除
- insert()
  - 指定位置の要素を挿入

StringBuilderの例

リストの要素をカンマで連結し、”[]”で囲む

```
public static <T> String list2String(List<T> list) {
  StringBuilder sb = new StringBuilder();
  sb.append("[");
  list.stream().forEachOrdered(
        p -> sb.append(p).append(",")
  );
  int k = sb.lastIndexOf(",");
  sb.deleteCharAt(k).append("]");
  return sb.toString();
}
```

正規表現

- 文字や文字列の繰り返しパターンを文字列として記述
- ^ : 文字列の先頭
  - ”^Java” : 文の先頭が”Java”である
- $: 文字列の終端
  - ”java$” : 文末が”java”である

- X? : Xが0または1回
- X+ : Xが1回以上
- X* : Xが0回以上
- X{n}：Xがn回
- X{n,}：Xがn回以上
- [abc] : a、b、またはc
- ¥s :空白文字(spaceやtabなど)
- ¥S : 空白文字以外
- ¥d :数字: [0-9]
- ¥D :数字以外

例：多様な区切り文字で文字列を分割

- space、タブ、カンマ、コロンなど、様々な区切り文字に対応
- String ss[] = s.split("¥¥s|,|:");
- “|”：パターンをorで連結
- 注意：javaでは”¥”は制御文字

文字列を見つける

- 正規表現の定義
  - Pattern p = Pattern.compile(String regex);
- matcherの生成
  - Matcher m = p.matcher(input);
- パターンの探索
  - boolean m.find() パターンの発見
  - int m.start() パターンの開始位置
  - String m.group() 一致した文字列

パターン探索の例

```
String input = "0010111010011";
//正規表現の定義
Pattern p = Pattern.compile("101+");
Matcher m = p.matcher(input);
while (m.find()) {//探索
  //一致した文字列
   String s = m.group();
   System.out.println("matches ");
}
```

```
String input = "0010111010011";
//正規表現の定義
Pattern p = Pattern.compile("101+");
Matcher m = p.matcher(input);
int c = 0;//探索開始位置
while (m.find(c)) {//位置を指定して探索
    c = m.start();//パターンを発見した位置
    String s = m.group();
    System.out.println("matches "+s + " at " + c);
    c++;//探索位置を一つ進める
}
```

正規表現グループ

- ((A)(B(C)))は以下のように番号付く
  1. ((A)(B(C)))
  2. (A)
  3. (B(C))
  4. (C)

文字列の置換

- 単純な置き換え
  - m.replaceFirst(置換文字列)
  - m.replaceAll(置換文字列)
- 一致した文字列の再利用
  - $0一致した全体
- 部分文字列の利用
  - $1、$2など

パターン置換の例

```
String input = "001011101001101";
//正規表現の定義
Pattern p = Pattern.compile("101+");
Matcher m = p.matcher(input);
//単純な置き換え
System.out.println(m.replaceFirst("121"));
System.out.println(m.replaceAll("121"));
//一致した文字列の利用
System.out.println(m.replaceAll("_$0_"));
//一致した部分の指定
p = Pattern.compile("(10)(1+)");
m = p.matcher(input);
System.out.println(m.replaceAll("12$2"));
```

BuilderExample.java

```java
package example;

import java.util.ArrayList;
import java.util.List;

/**
 *
 * @author tadaki
 */
public class BuilderExample {

    public static <T> String list2String(List<T> list) {
        StringBuilder sb = new StringBuilder();
        sb.append("[");
        list.stream().forEachOrdered(
                p -> sb.append(p).append(",")
        );
        int k = sb.lastIndexOf(",");
        sb.deleteCharAt(k).append("]");
        return sb.toString();
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        int n = 10;
        List<Integer> list = new ArrayList<>();
        for (int i = 0; i < n; i++) {
            list.add((int) (n * Math.random()));
        }
        System.out.println(list2String(list));
    }

}
```

SplitExample.java

```java
package example;

/**
 *
 * @author tadaki
 */
public class SplitExample {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        String input[] = {
            "a,b,c,d,e,f",
            "a b c d e f",
            "a¥tb¥tc¥td¥te¥tf",
            "a:b:c:d:e:f"
        };
        for (String s : input) {
            String ss[] = s.split("¥¥s|,|:");
            for (String e : ss) {
                System.out.print(e + " ");
            }
            System.out.println();
        }

    }

}
```

RegexSample.java

```java
package example;

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

/**
 *
 * @author tadaki
 */
public class RegexSample {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        String input = "0010111010011";
        //正規表現の定義
        Pattern p = Pattern.compile("101+");
        Matcher m = p.matcher(input);
        int c = 0;//探索開始位置
        while (m.find(c)) {//位置を指定して探索
            c = m.start();//パターンを発見した位置
            String s = m.group();
            System.out.println("matches "+s + " at " + c);
            c++;//探索位置を一つ進める
        }
    }

}
```

ReplaceExample.java

```java
package example;

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

/**
 *
 * @author tadaki
 */
public class ReplaceExample {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        String input = "001011101001101";
        //正規表現の定義
        Pattern p = Pattern.compile("101+");
        Matcher m = p.matcher(input);
        //単純な置き換え
        System.out.println(m.replaceFirst("121"));
        System.out.println(m.replaceAll("121"));
        //一致した文字列の利用
        System.out.println(m.replaceAll("_$0_"));
        //一致した部分の指定
        p = Pattern.compile("(10)(1+)");
        m = p.matcher(input);
        System.out.println(m.replaceAll("12$2"));
    }

}
```