Expressions régulières pour Android

1

Classes de caractères

- Une expression régulière est composée de une ou plusieurs classes de caractères.
- Une classe de caractères:
  - [abc] : les caractères entre [ ] sont utilisés pour remplacer un caractère. Ainsi, [abc] signifie un caractère parmi a, b ou c.
  - [^abc] : tous les caractères sauf ceux énumérés.
  - [a-z] : tous les caractères compris entre a et z inclusivement.
  - \w est équivalent à [a-zA-Z0-9_]
  - \W est équivalent à [^ a-zA-Z0-9_]
  - \d est équivalent à [0-9]
  - \D est équivalent à [^0-9]
  - \s est équivalent à [\t\r\n\v\f]
  - \S est équivalent à [^ \t\r\n\v\f]

•2

CARACTÈRES SPÉCIAUX

- $ indique la fin d'une ligne
- ^ indique le début d'une ligne
- . indique n'importe quel caractère sauf le retour à la ligne
- Exemple:

  ^[a-z][0-9]$: indique deux caractères dont le premier est une lettre minuscule et le second un chiffre. Comme le mot doit commencer par la lettre (tout de suite après le ^) et se terminer par un chiffre (il y a le $ tout de suite après le chiffre), nous n'acceptons que les mots de 2 caractères.

"^.{3}$": chaîne qui contient 3 caractères

".*": Tous les caractères

RÉPÉTITIONS

- Permettent de préciser combien de fois un élément d'une expression régulière peut (doit) être répété.
- {n, m} au moins n fois et au plus m fois.
- {n, } au moins n fois.
- {n} exactement n fois.
- ? 0 ou 1 fois.
- + équivalent à {1, }
- * équivalent à {0, }
- ex: [a-z]+ // au moins une lettre minuscule
  [0-9]{5} // exactement 5 chiffres

"abc+": chaîne qui contient "ab" suivie de un ou plusieurs "c" ("abc", "abcc", etc.)

"abc*": chaîne qui contient "ab" suivie de zéro ou plusieurs "c" ("ab", "abc", etc.)

"abc?": chaîne qui contient "ab" suivie de zéro ou un "c" ("ab" ou "abc" uniquement)

"^abc+": chaîne commençant par "ab" suivie de un ou plusieurs "c" ("abc", "abcc", etc.)

Les accolades {X,Y} permettent de donner des limites précises de nombre d'occurences.

"abc{2}": chaîne qui contient "ab" suivie de deux "c" ("abcc")

"abc{2,}": chaîne qui contient "ab" suivie de deux "c" ou plus ("abcc" etc..)

"abc{2,4}": chaîne qui contient "ab" suivie 2, 3 ou 4 "c" ("abcc" .. "abcccc")

Il est à noter que le premier nombre de la limite est obligatoire ("{0,2}", mais pas "{,2}"). Les symboles vu précedemment ('*', '+', et '?') sont équivalents à "{0,}", "{1,}", et "{0,1}".

•5

GROUPEMENT

- | permet de spécifier des alternatives comme dans a | b qui signifie a ou b.
- ( ) permet de regrouper comme dans (ab | cd)+ | ef qui signifie ef ou encore une ou plusieurs occurences de ab ou cd.

•6

GROUPEMENT

- | permet de spécifier des alternatives comme dans / a | b / qui signifie a ou b.
- ( ) permet de regrouper comme dans / (ab | cd)+ | ef / qui signifie ef ou encore une ou plusieurs occurences de ab ou cd.

Quelques exemples :

Code postal

^[A-Za-z][0-9][A-Za-z] ?[0-9][A-Za-z][0-9]$

Courriel

^[a-zA-Z0-9_-]+@[a-zA-Z0-9-]{2,}[.][a-zA-Z]{2,3}$

NAS

^[0-9]{3}[\-]{1}[0-9]{3}[\-]{1}[0-9]{3}$

Meilleur courriel (voir explications à la fin des exemples)

^(\w+[\-\.])*\w+@(\w+\.)+[A-Za-z]+$

Mot de passe

^[A-Za-z\d]{6,8}$

9

EXPLICATIONS

^(\w+\.)*\w+@(\w+\.)+[A-Za-z]+$

L'accent circonflexe (^) dit de commencer au début. Cela empêche l'utilisateur d'entrer des caractères invalides au début de l'adresse courriel.

(\ w + [\ - \.]) * permet une séquence de caractères suivi d'un point ou un tiret. Le * indique que le motif peut être répété zéro ou plusieurs fois. Modèles réussis incluent "ndunn. ", "Ndunn-", "nat.s.", et "nat-s-".

\ w + permet un ou plusieurs caractères .

@ Permet un symbole @ unique.

(\ w + \.) + permet une séquence de caractères suivi d'un point. Le + indique que le modèle peut être répétée une ou plusieurs fois. C'est le nom de domaine sans la dernière partie (par exemple, sans la «com» ou «org»).

[A-Za-z] + permet à une ou plusieurs lettres. Il s'agit de la "com" ou « org" partie de l'adresse courriel.

Le signe dollar ($) indique la fin. Cela empêche l'utilisateur d'entrer des caractères invalides à la fin de l'adresse mail.

10

Class Pattern et Matcher

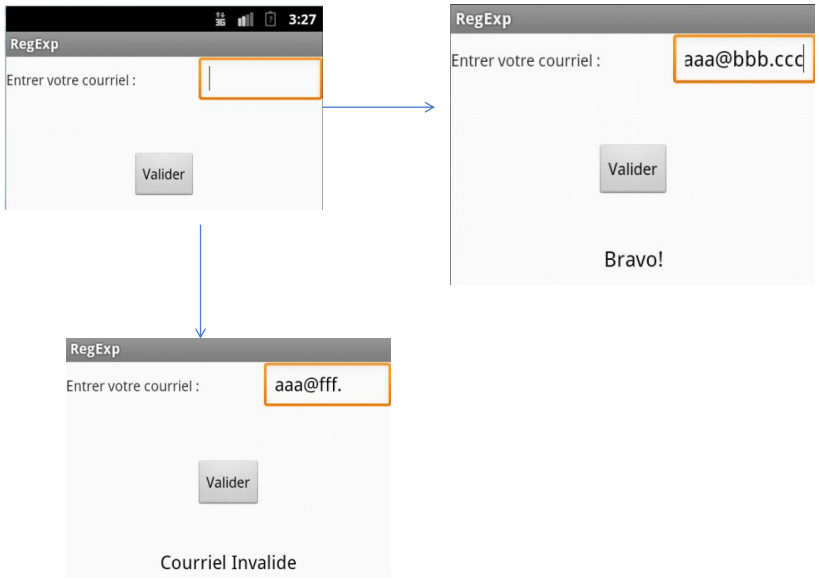
```
String loginTxt = login.getText().toString();

String passTxt = pass.getText().toString();


if (loginTxt.equals("") || passTxt.equals("")) {

      Toast.makeText(MainActivity.this,

      R.string.erreur_courriel_pass,

      Toast.LENGTH_SHORT).show();
return; //sortir de la fonction
}
```

11

```
// On déclare le pattern que l'on doit suivre
Pattern p = Pattern.compile(".+@.+\.[a-z]+");
// On déclare un matcher, qui comparera le
pattern avec la
// string passée en argument
Matcher m = p.matcher(loginTxt);
// Si l'adresse mail saisie ne correspond au
format d'une
// adresse mail
if (!m.matches()) {
Toast.makeText(MainActivity.this,
R.string.erreur_courriel,
Toast.LENGTH_SHORT).show();
```

12

Exemple : (projet RegExp)



13

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >

    <EditText
        android:id="@+id/courriel"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="63dp"
        android:layout_toRightOf="@+id/textView1"
        android:inputType="textEmailAddress"
        android:ems="10" >

        <requestFocus />
    </EditText>
```

14

```
<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/courriel"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="48dp"
    android:text="@string/texte_bouton" />

<TextView
    android:id="@+id/msg"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/courriel"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="136dp"
    android:text=""
    android:textAppearance="?android:attr/textAppearanceMedium"
/>
```

15

```
<TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/courriel"
    android:layout_alignBottom="@+id/courriel"
    android:layout_alignParentLeft="true"
    android:text="@string/courriel" />

</RelativeLayout>
```

16

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import android.os.Bundle;
import android.app.Activity;
import android.widget.*;
import android.view.*;
import android.view.View.OnClickListener;


public class MainActivity extends Activity {
Button b;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);

      b= (Button)findViewById(R.id.button1);
b.setOnClickListener(new OnClickListener() {
@Override
public void onClick(View v) {
validerCourriel();
}
});
}
```

17

```
public void validerCourriel() {
EditText codeUsager = (EditText) findViewById(R.id.courriel);
String courriel = codeUsager.getText().toString();
TextView msg = (TextView) findViewById(R.id.msg);
Pattern p = Pattern.compile(".+@.+\\.[a-z]+");

Matcher m = p.matcher(courriel);

if (!m.matches()) {
        msg.setText("Courriel Invalide");

}
else
        msg.setText("Bravo!");

//Toast.makeText(MainActivity.this,R.string.msg,
//Toast.LENGTH_SHORT).show();
}

}
```

18