7. STYLOVÁNÍ A SDÍLENÍ DAT

BI-AND

Evropský sociální fond
Praha & EU: Investujeme do vaší budoucnosti

7. přednáška

- Stylování
- Themování
- 9-patch
- Sdílení dat
- Shared Preferences
- Android Interface Definition Language

Vzhled aplikace

- Každý výrobce má většinou vlastní uživatelské prostředí
  - Nahrazení „stock/nativního“ vzhledu widgetů a dalších prvků
    - TouchWiz, HTC Sense, Timescape, atd.
  - Není možné se spolehnout na přítomnost nativních prvků
  - Holo Theme přítomné ve všech zařízení s Androidem 4.0+
- Obtížné přizpůsobení vzhledu aplikace
  - Vytvořit vlastní vzhled widgetů, dialogů a dalších prvků
    - Nelze snadno poznat, jaká úprava UI je použita
  - Používat prvky z Holo Theme nebo starší nativní prvky
- Úprava vzhledu bývá většinou velmi náročná

Stylování a themování vzhledu aplikace

DRY – Don‘t repeat yourself

- Styl
  - Sbírka vlastností, která specifikuje vzhled `View` či `Window`

```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textColor="#00FF00"
    android:typeface="monospace"
    android:text="@string/hello" />
```

```
<TextView
    style="@style/CodeFont"
    android:text="@string/hello" />
```

Vlastnosti `layout_width`, `layout_height`, `textColor` a `typeface` je možné přesunout do vlastního stylu CodeFont

- Theme
  - Styl aplikovaný na activitu či celou aplikaci

Definování stylu

- Vytvoříme XML soubor ve složce res/values
- Kořenový tag musí být `<resources>`
- Zvolíme název a případně rodiče stylu, od kterého se má dědit
- Nadefinujeme vlastní hodnoty k příslušným názvům vlastností

- ADT plugin pro Eclipse podporuje „extrahování“ stylu

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="CodeFont"
                parent="@android:style/TextAppearance.Medium">
        <item name="android:layout_width">fill_parent</item>
        <item name="android:layout_height">wrap_content</item>
        <item name="android:textColor">#00FF00</item>
        <item name="android:typeface">monospace</item>
    </style>
</resources>
```

Vlastnosti stylů

- Styl může být použitý na
  - Widget
  - Container
- Vždy však ovlivní pouze prvek, na který je použit
  - Např. styl „RedText“ použitý na LinearLayout neovlivní potomka TextView
- Atributy je možné najít např. Android Javadoc pod
  - XML Attributes
  - Inherited XML Attributes
  - R.attr
- Android ignoruje neplatné styly
  - Není vyvolána žádná compile-time ani run-time výjimka

Dědičnost stylů

- Chceme konkrétního rodiče (např. Theme.Holo) s částečně jinými vlastnostmi
- Přidáme nebo změníme požadované vlastnosti
  - U zbývajících vlastností jsou použity vlastnosti rodiče
- Používá se atribut parent nebo „tečková notace“

```
<style name="GreenText" parent="@android:style/TextAppearance" >
    <item name="android:textColor">#00FF00</item>
</style>
```

Rozdíl?

```
<style name="MyText.Green" >
    <item name="android:textColor">#00FF00</item>
</style>
```

Themování příklad

```
<resources>
    <style name="Theme" parent="android:Theme"></style>

    <style name="Theme.Translucent">
        <item name="android:windowBackground">
        @drawable/translucent_background</item>
        <item name="android:windowNoTitle">true</item>
        <item name="android:colorForeground">#fff</item>
    </style>

    <style name="Theme.Transparent">
        <item name="android:windowBackground">
        @drawable/transparent_background</item>
        <item name="android:windowNoTitle">true</item>
        <item name="android:colorForeground">#fff</item>
    </style>

    <style name="TextAppearance.Theme.PlainText"
        parent="android:TextAppearance.Theme">
        <item name="android:textStyle">normal</item>
    </style>
</resources>
```

Theme použít u <activity> nebo <application>

Druhy atributů

- Typy atributů (format)
  - string
  - fraction
  - enum
  - flag
  - reference (na jiný resource)
  - color
  - boolean
  - dimension
  - float
  - integer
- Lze volit více druhů současně
- Vlastní jsou definovány pomocí <declare-styleable> (name + format)
- Důležité pro vytváření vlastních widgetů
- Více v 11. přednášce

Používání atributu reference

- Pokud použití stylů přímo v XML layoutu nechceme být vázáni na konkrétní styl, ale styl použitý až podle daného theme

Atribut se vyhodnotí až v run-time

- Ve složce values vytvořit soubor attrs.xml

```
<attr name="textTitle" format="reference">
```

- Danou referenci použít v XML layoutu pomocí „?název“

- Nastavit hodnoty (konkrétní styly) pro atributy v daných themes

Používání atributů

- Obdobně lze použít atributy i v jiném formátu (např. integer) a použít pro změnu drawable (např. src u Button)

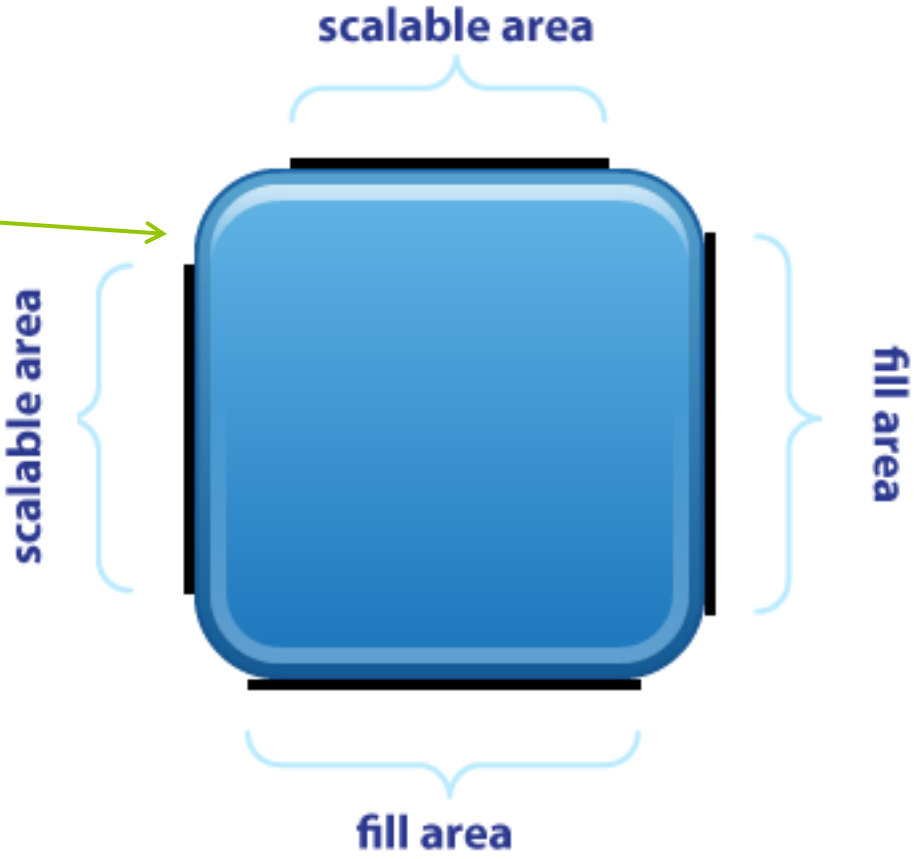
- Lze používat i atributy přímo z operačního systému

```
<ProgressBar
style="?android:attr/progressBarStyleHorizontal”>
```

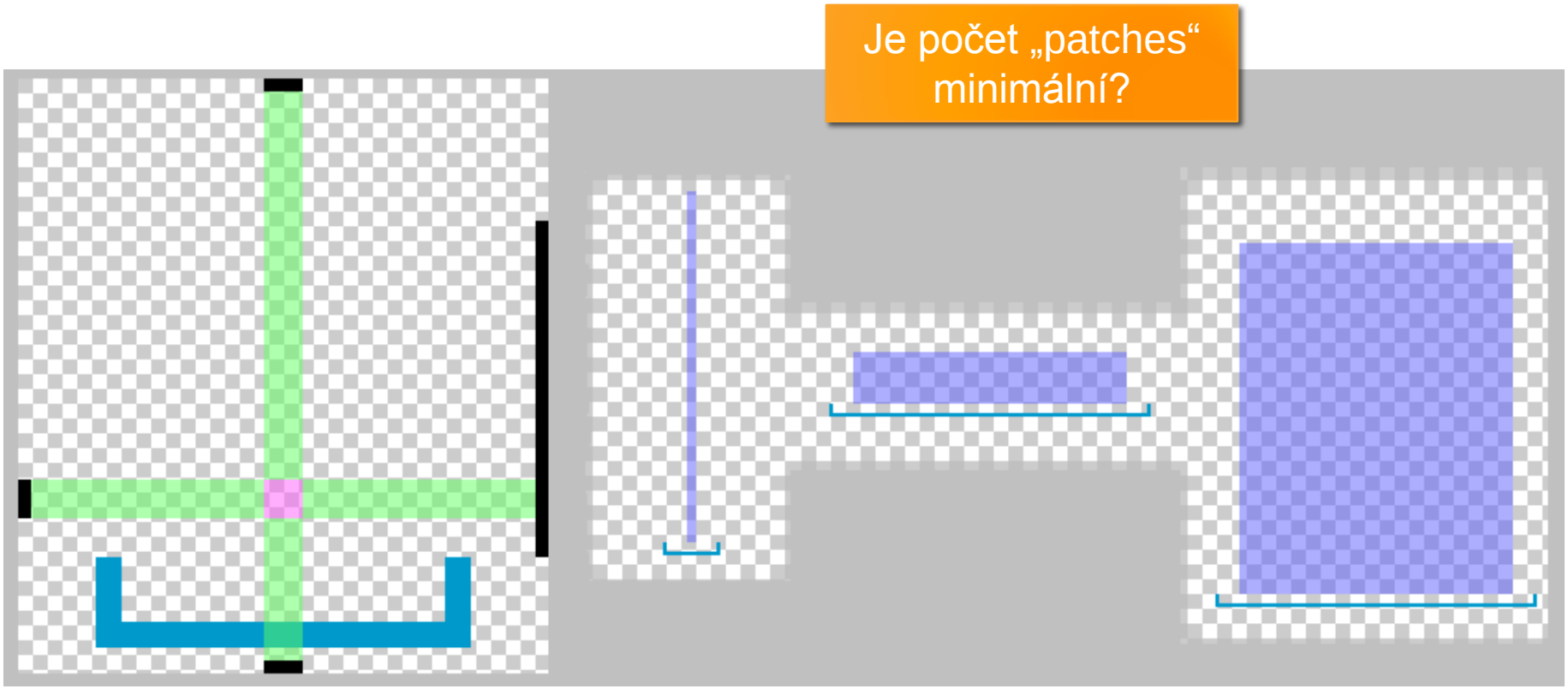
- Bude použit styl, který je vázán na aktuální theme dané verze OS

Draw 9-patch (draw9patch)

- WYSIWYG editor pro tvorbu 9-patch
- 9-patch drawable
  - PNG obrázek
  - Má tzv. ,,patche“ – v nich se obráze roztahuje a škáluje
  - Vhodné pro různé velikosti obrazovek pro
    - Pozadí tlačítek
    - Rámečky
    - Mnohé další prvky
- Další druhy Drawables na 11. přednášce



Ukázka 9-patch pro EditText



Data

- Persistentní (trvalá)
  - Data musejí být dostupná vždy (i po restartu aplikace)
- Nepersistentní (netrvalá)
  - Data potřebná v jedné instanci aplikace

Možnosti sdílení persistentních data

- Soubory
- Databáze
- Content Provider
- Shared Preferences

Shared Preferences

- `android.content.SharedPreferences`
- Třída pro ukládání a načítání trvalých párů klíč-hodnota
- Uložení jakéhokoli primitivního typu
  - `boolean, float, int, long, String`
- Data se uchovávají v XML
  - Umístění: `/data/data/<jméno aplikace>/shared_prefs/`

Příklad XML s uloženými SharedPreferences:

```
1. <?xml version='1.0' encoding='utf-8' standalone='yes' ?>
2. <map>
3.     <long name="longkey" value="1" />
4.     <boolean name="booleankey" value="false" />
5.     <string name="stringkey">text</string>
6. </map>
```

Shared Preferences

- Instanci můžeme získat několika metodami

- `PreferenceManager.getDefaultSharedPreferences(Context);`
  - Vrátí výchozí `SharedPreferences` společné pro všechny komponenty aplikace

- `Context.getSharedPreferences(name, mode);`
  - Vrátí `SharedPreferences` identifikované parametrem `name`

Shared Preferences

- `Activity.getPreferences(mode);`
  - Vrátí `SharedPreferences` unikátní pro danou aktivitu
  - Alternativa k volání metody `getSharedPreferences` se jménem třídy v prvním parametru
- Oprávnění můžeme určit parametrem `mode`
  - *`MODE_PRIVATE`*
  - *`MODE_WORLD_WRITEABLE`*
  - *`MODE_WORLD_READABLE`*
- Přístup z jiné aplikace pomocí

```
1. createPackageContext("cz.cvut.and", 0);
```

Shared Preferences – ukládání dat

- K editování a následnému uložení se používá `SharedPreferences.Editor`
- Instance se získá zavoláním `edit` na `SharedPreferences`
- Samotné uložení se provede příkazem `commit`

```
1. SharedPreferences sp =getPreferences(MODE_PRIVATE);
2. Editor ed = sp.edit();
3. ed.putString("klic", "hodnota");
4. ed.putFloat("lastFloat", 1f);
5. ed.putInt("wholeNumber", 2);
6. ed.putLong("aNumber", 3l);
7. ed.putBoolean("isTrue", true);
8. ed.commit();
```

apply() vs commit()

Shared Preferences – načítání dat

- Pro každý typ, který může být v `SharedPreferences` uložen, existuje getter, který přijímá dva parametry:
  - Klíč
  - Defaultní hodnotu, která se vrátí, pokud záznam s daným klíčem neexistuje
- Pro zjištění, jestli hodnota s daným klíčem existuje, se může použít metoda `SharedPreferences.contains(key);`

```
1.  SharedPreferences sp = getPreferences(MODE_PRIVATE);
2.  String text = sp.getString("text", "");
3.  int number = sp.getInt("wholeNumber", 0);
4.  float lastFloat = sp.getFloat("lastFloat", 0f);
5.  long aNumber = sp.getLong("aNumber", 0);
6.  boolean isTrue = sp.getBoolean("isTrue", false);
```

Možnosti sdílení nepersistentních dat

- Intent
- Třída `android.app.Application`
- Veřejná statická proměnná nebo metoda
- Singleton object
- Service `(android.app.Service)`
  - Systém ji může automaticky zabít pouze v extrémních situacích
  - Vhodná pouze pro zpracování déle trvajících operací
- Bundle
  - Data jsou ztracena po ukončení aplikace

Sdílení dat přes Intent

- `putExtra(klíč, hodnota)`
  - Klíč – `String`
  - Hodnota
    - Primitivní datové typy
    - Objekty implementující
      - `java.io.Serializable`
        - Většina objektů Java a Android knihovny mají již implementováno
        - Rychlé na implementaci
      - `android.os.Parcelable`
        - Rychlejší zpracování
        - Časové náročnější na implementaci

- Data je možné poslat activitě, která běžící activitu spustila
  - `startActivityForResult(Intent intent, int requestCode)`

Posílání dat přes Intent - příklad

ActivityA.java

```
Intent intent = new Intent(getApplicationContext(),
ActivityB.class);
intent.putExtra("cz.cvut.fit.secret", "hello world");
startActivity(intent);
```

ActivityB.java

```
Bundle extras = getIntent().getExtras();
        if (extras != null) {
            String string =
                extras.getString("cz.cvut.fit.secret");
}
```

Parcelable - příklad

```
public class ParcelableData implements Parcelable
```

```
private Date date;
private String message;
private double value;

public Date getDate() { return date; }
public String getDetails() { return message; }
public double getMagnitude() { return value; }

public ParcelableData(Date date, String message,
                      double value) {
  this.date = date;
  this.message = message;
  this.value = value;
}
```

Parcelable – pokračování příkladu

Data zapíšeme do kontejneru `Parcel`

```
public void writeToParcel(Parcel out, int flags) {
  out.writeLong(date.getTime());
  out.writeString(message);
  out.writeDouble(value);
}

private ParcelableData(Parcel in) {
  date = new Date();
  date.setTime(in.readLong());
  message = in.readString();
  value = in.readDouble();
}
```

Data vytvoříme z kontejneru `Parcel`

Parcelable – pokračování příkladu

Generátor instance `ParcelableData`

```
public static final Parcelable.Creator<ParcelableData>
CREATOR =
  new Parcelable.Creator<ParcelableData>() {
    public ParcelableData createFromParcel(Parcel in) {
      return new ParcelableData(in);
    }

    public ParcelableData[] newArray(int size) {
     return new ParcelableData[size];
    }
  };

public int describeContents() {
  return 0;
}
```

Použito např. pro potomky třídy

Sdílení pomocí `Application`

- Výhody
  - Komplexní kontrola nad správou životního cyklu dat
    - Přehledná inicializace a destrukce zdrojů
  - Centralizovaný přístup, kam může získat přístup jakákoliv `Activita` či `Service`
- Nevýhoda
  - Data jsou dostupná pouze v rámci aplikace

Sdílení pomocí `Application`

Do `AndroidManifest.xml` k elementu `<application>` je nutno přidat

```
android:name=".GlobalState"
```

Sdílený objekt s potřebnými proměnnými a metodami

```
public class GlobalState extends Application {
    private String text;
    public String getText() {
        return text;    }
    public void setText(String testMe) {
        this.text = testMe;
    }
}
```

Použití vlastní třídy Application

Přístup ke sdílenému objektu

```
GlobalState globalState = (GlobalState) getApplication();
```

Singleton

- Velmi podobné sdílení přes `Application`
- Vlastní a obtížnější kontrola životního cyklu dat
- Obtížnější pro testování
- Používat nejenom jako úschovnu dat

Singleton - příklad

```
public class SingletonObject {
    private static SingletonObject instance = null;
    private String text;

    public static synchronized SingletonObject getInstance() {
        if (instance == null) {
            instance = new SingletonObject();
        }
        return instance;
    }

    public void setText(String text) {
        this.text = text;
    }

    public String getText() {
        return text;
    }
}
```

```
SingletonObject.getInstance().setText("hello world");
```

Android Interface Definition Language (AIDL)

- Umožňuje definovat programové rozhraní, přes které komunikuje klient a služba za použití meziprocesové komunikace (Interprocess Communication - IPC)
- Nástavba IPC je právě AIDL
  - Usnadňuje práci s nízko úrovňovým protokolem
- V Androidu proces nemůže sdílet paměť s jiným procesem
  - Objekty musí být rozloženy na „jednoduché“ datové typy (příp. Parcelable), s nimiž systém umí pracovat
  - Nutné vytvořit interface (.aidl), který bude určovat dostupné služby a zde popsat signaturu metod

Android Interface Definition Language (AIDL)

- AIDL umí pracovat s:
  - Všemi primitivními datovými typy v Javě
  - String a CharSequence
  - List – vždy použit/vrácen ArrayList
  - Map - vždy použit/vrácen HashMap
- List a Map může obsahovat pouze ostatní datové typy podporované AIDL

- Postup implementace:
  - 1. Vytvořit .aidl s definicí interface
  - 2. Implementovat daný interface v Javě
  - 3. Implementovat Service pro předání Binderu

Další zdroje

- http://ofps.oreilly.com/titles/9781449390501/AIDL.html
- http://android10.org/index.php/articlesother/279-draw-9-patch-tutorial