Android Logger

Logujemy przy pomocy **android.util.Log**.
Korzystamy z niego następująco:

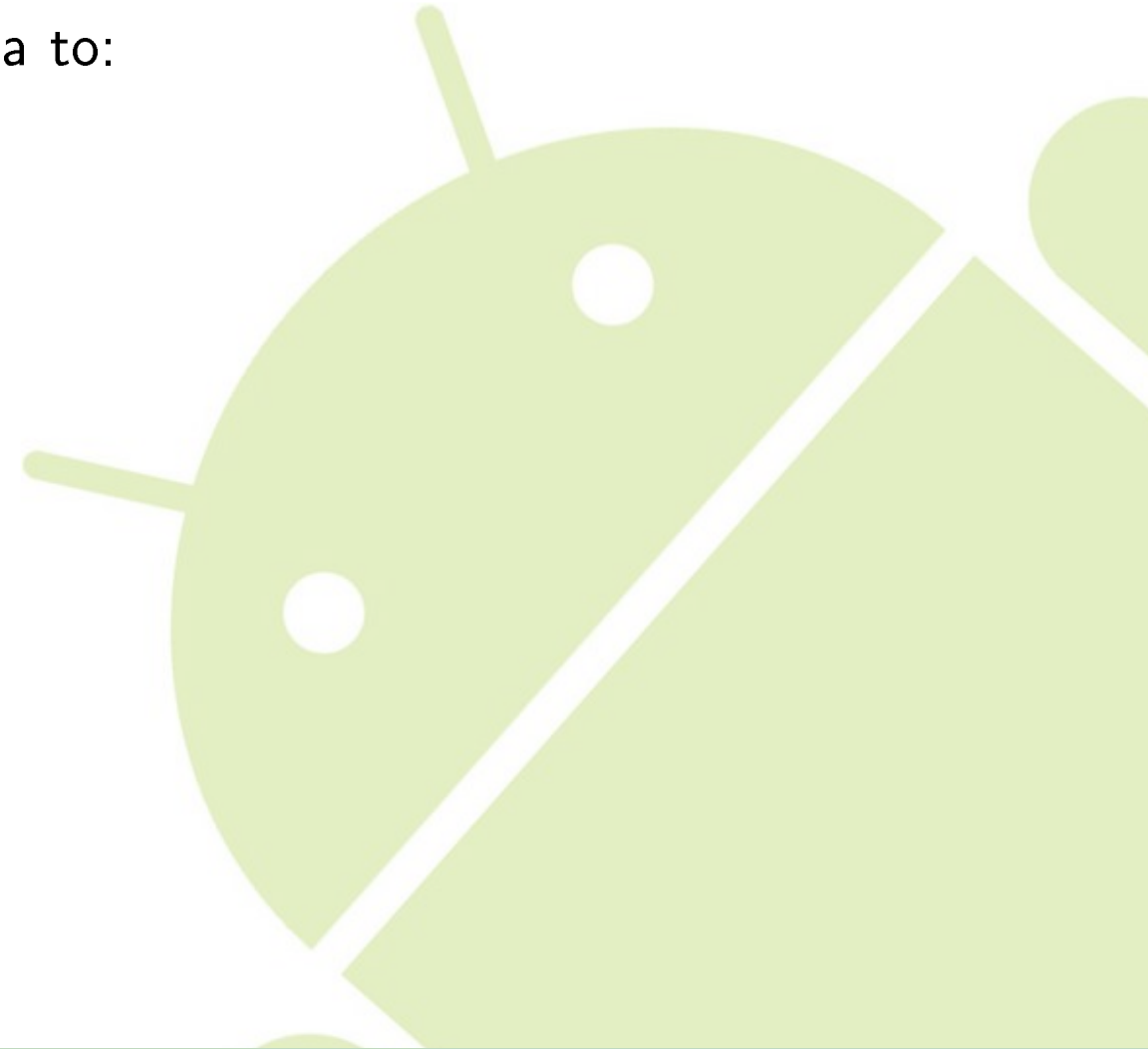
```
class MyClass {
  private final String TAG = getClass()
                               .getSimpleName();

  void doStuff() {
    Log.d(TAG, "doing␣stuff...");  // debug
  }
}
```

Android Log levels

Dostępne poziomy logowania to:

- `Log.v()` - VERBOSE
- `Log.d()` - DEBUG



Android Log levels

Dostępne poziomy logowania to:

- `Log.v()` - VERBOSE
- `Log.d()` - DEBUG
- `Log.i()` - INFO

Android Log levels

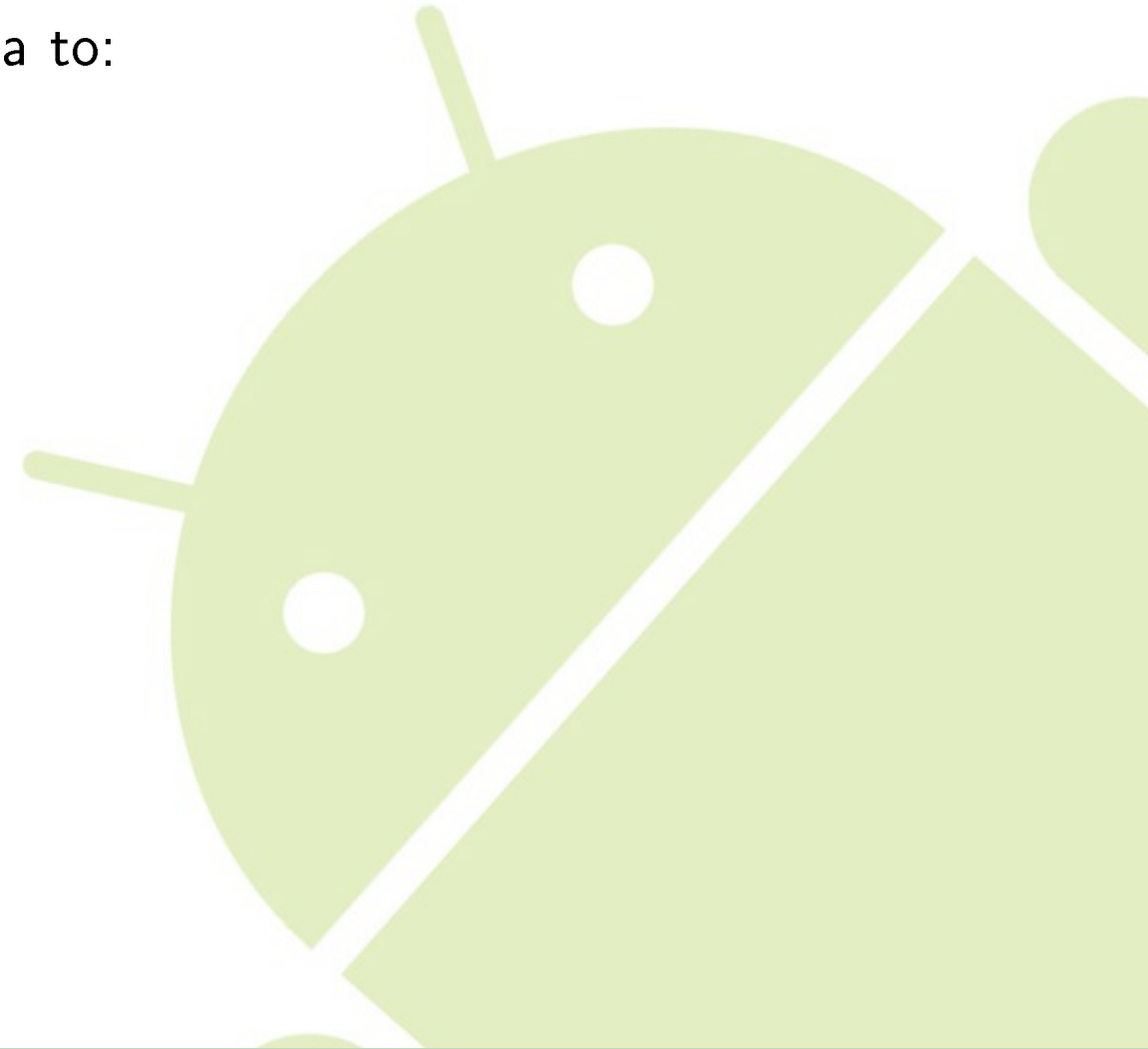
Dostępne poziomy logowania to:

- `Log.v()` - VERBOSE
- `Log.d()` - DEBUG
- `Log.i()` - INFO
- `Log.w()` - WARN

Android Log levels

Dostępne poziomy logowania to:

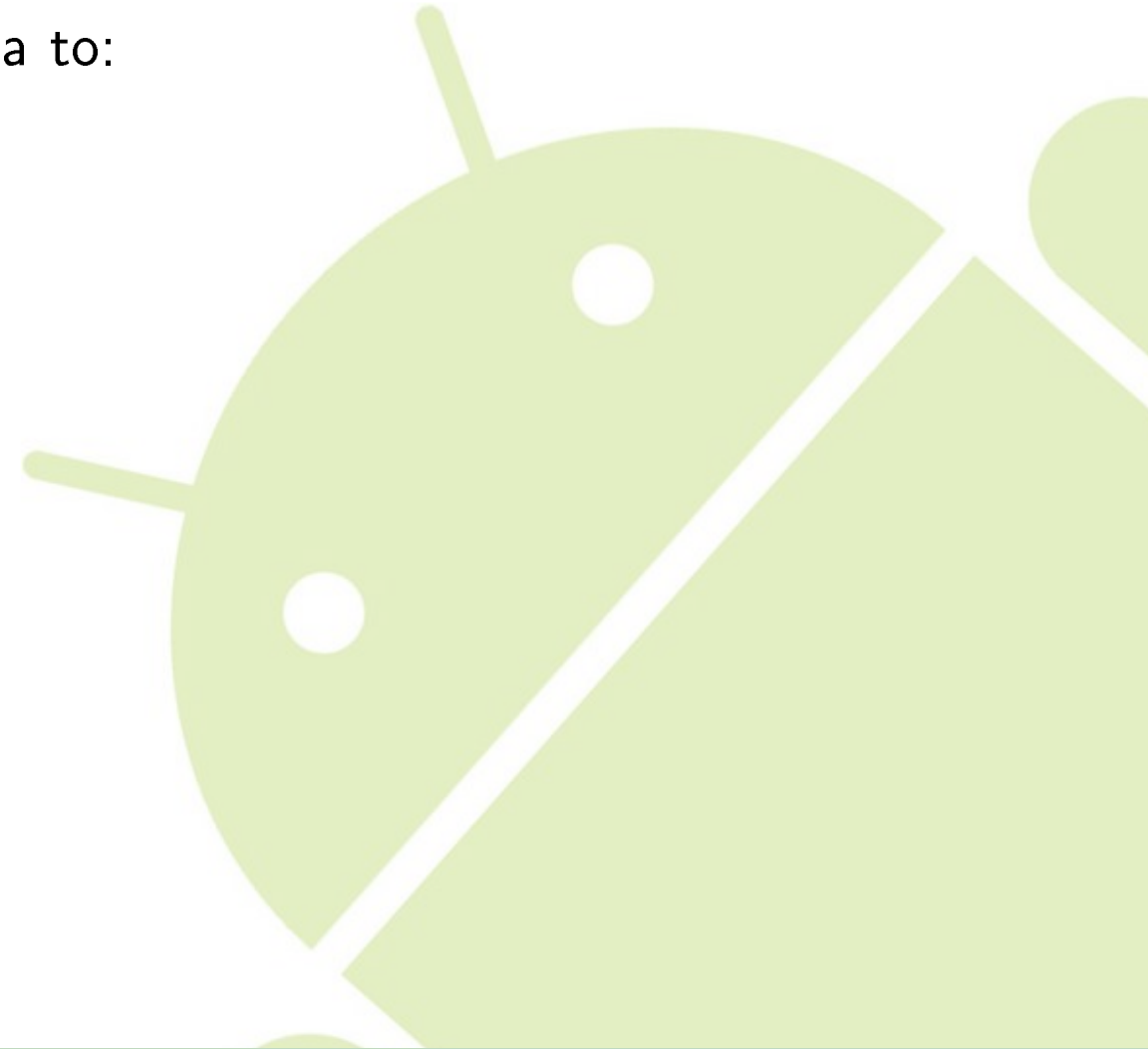
- `Log.v()` - VERBOSE
- `Log.d()` - DEBUG
- `Log.i()` - INFO
- `Log.w()` - WARN
- `Log.e()` - ERROR



Android Log levels

Dostępne poziomy logowania to:

- `Log.v()` - VERBOSE
- `Log.d()` - DEBUG
- `Log.i()` - INFO
- `Log.w()` - WARN
- `Log.e()` - ERROR
- `Log.wtf()`



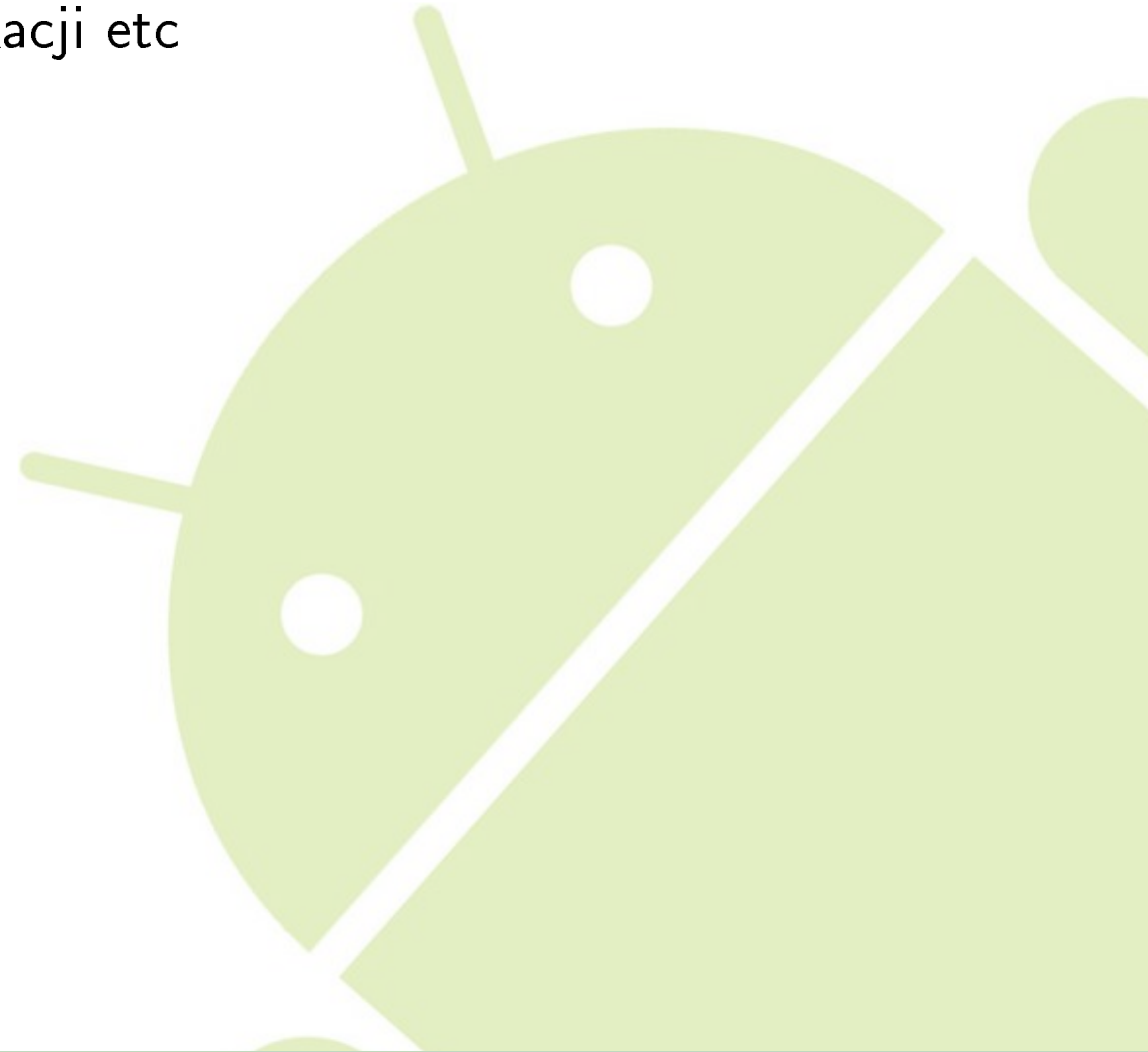
Android Log levels

Dostępne poziomy logowania to:

- `Log.v()` - VERBOSE
- `Log.d()` - DEBUG
- `Log.i()` - INFO
- `Log.w()` - WARN
- `Log.e()` - ERROR
- `Log.wtf()` - **W**hat a **T**errible **F**ailure

Sposoby przechowywania danych

- **Shared Preferences** - Prosty KeyValue store, idealny dla prostych ustawień aplikacji etc



Sposoby przechowywania danych

- **Shared Preferences** - Prosty KeyValue store, idealny dla prostych ustawień aplikacji etc
- **Internal Storage** - Zapisywanie plików w swoim formacie na **wewnątrznej pamięci** urządzenia (dobre dla cache obrazków etc)

Sposoby przechowywania danych

- **Shared Preferences** - Prosty KeyValue store, idealny dla prostych ustawień aplikacji etc
- **Internal Storage** - Zapisywanie plików w swoim formacie na **wewnątrznej pamięci** urządzenia (dobre dla cache obrazków etc)
- **External Storage** - Zapisywanie plików w swoim formacie na np. **karcie SD** (dobre dla cache obrazków etc)

Sposoby przechowywania danych

- **Shared Preferences** - Prosty KeyValue store, idealny dla prostych ustawień aplikacji etc
- **Internal Storage** - Zapisywanie plików w swoim formacie na **wewnątrznej pamięci** urządzenia (dobre dla cache obrazków etc)
- **External Storage** - Zapisywanie plików w swoim formacie na np. **karcie SD** (dobre dla cache obrazków etc)
- **SQLite Database** - Zwyczajna instancja bazy danych SQLite - m.in. tym sposobem dostajemy informacje o kontaktach

Sposoby przechowywania danych

- **Shared Preferences** - Prosty KeyValue store, idealny dla prostych ustawień aplikacji etc
- **Internal Storage** - Zapisywanie plików w swoim formacie na **wewnątrznej pamięci** urządzenia (dobre dla cache obrazków etc)
- **External Storage** - Zapisywanie plików w swoim formacie na np. **karcie SD** (dobre dla cache obrazków etc)
- **SQLite Database** - Zwyczajna instancja bazy danych SQLite - m.in. tym sposobem dostajemy informacje o kontaktach
- **Cloud Storage** - Nie trzymamy danych lokalnie, pchamy i pobieramy wszystko z chmurki

Shared Preferences

SharedPreferences - API

Uzyskanie instancji:

```
// zawsze zadziala:
Context ctx = getApplicationContext();

// w Activity lub Service jest latwiej:
// ctx = this;

SharedPreferences prefs = PreferenceManager
                    .getDefaultSharedPreferences(ctx);
```

SharedPreferences - Read API

Przykład **odczytania** zmiennej:

```
// oto jak korzystac z @strings/ w Activity
String key = getString(R.string.key_sound_notif);

String s = preferences.getString(key, "undefined");

// analogicznie dla Integer/Long/Double/StringSet
```

SharedPreferences - Write API

Przykład **zapisania** zmiennej:

```
SharedPreferences.Editor editor = preferences.edit();

editor.putString(key, "new value");
// analogicznie dla Integer/StringSet/Double ...

editor.commit();
```

Shared Preferences

Shared w sensie „wewnątrz **naszej** aplikacji”, nie między wieloma.
SharedPreferences zapisywane są w:

```
/data/data/pl.project13.myapp/shared_prefs
```

Shared Preferences

Shared w sensie „wewnątrz **naszej** aplikacji”, nie między wieloma.
SharedPreferences zapisywane są w:

```
/data/data/pl.project13.myapp/shared_prefs
```

Można się tam dobrać gdy się jest **root**:

```
$ abd shell
> cat /data/data/pl.project13.myapp/shared_prefs

<map>
 <string name="pass">zomg_its_plain_text</string>
 <!-- ... -->
</map>
```

Security a SharedPreferences

Wniosek jest prosty:
Szyfrujemy ważne rzeczy trzymane gdziekolwiek na komórce.

Robo Guice - Google Guice for Android

Google Guice = **JSR-330** Dependency Injection for Java

RoboGuice - co zyskujemy?

Przed:

```
class Act extends Activity {
 SharedPreferences prefs;
 EditText mLogin;
 public void onCreate(Bundle savedState) {
  prefs = PreferenceManager
          .getDefaultSharedPreferences(this);
  mLogin = (EditText) findById(R.id.login);
 }
}
```

RoboGuice - co zyskujemy?

Przed:

```
class Act extends Activity {
 SharedPreferences prefs;
 EditText mLogin;
 public void onCreate(Bundle savedState) {
  prefs = PreferenceManager
          .getDefaultSharedPreferences(this);
  mLogin = (EditText) findById(R.id.login);
 }
}
```

Po:

```
class Act extends RoboActivity {
 @Inject SharedPreferences prefs;
 @InjectView(R.id.login) EditText mLogin;
 public void onCreate(Bundle savedState){ /**/ }
}
```

Zapięcie RoboGuice w 4 krokach:

1. własny **App extends RoboApplication** gdzie nadpisujemy **#addApplicationModules**

Zapięcie RoboGuice w 4 krokach:

1. własny **App extends RoboApplication** gdzie nadpisujemy **#addApplicationModules**
2. własny **SzczecinModule extends AbstractAndroidModule**, który dodajemy powyżej

Zapięcie RoboGuice w 4 krokach:

1. własny **App extends RoboApplication** gdzie nadpisujemy **#addApplicationModules**
2. własny **SzczecinModule extends AbstractAndroidModule**, który dodajemy powyżej
3. dodanie `<application android:name=".App"` dla naszej aplikacji w **AndroidManifest.xml**

Zapięcie RoboGuice w 4 krokach:

1. własny **App extends RoboApplication** gdzie nadpisujemy **#addApplicationModules**
2. własny **SzczecinModule extends AbstractAndroidModule**, który dodajemy powyżej
3. dodanie `<application android:name=".App"` dla naszej aplikacji w **AndroidManifest.xml**
4. zamiana `MyActivity extends Activity` na: `MyActivity extends RoboActivity`

Zadanie: Kroczki do przodu

- podpinamy **RoboGuice**
- zapisujemy **imię użytkownika** w **SharedPreferences**
- podpinamy **res/menu/menu.xml** (1 element menu, o nazwie 'settings') (tip: robi się to w **onCreateOptionsMenu()**)

tip:

```
bindConstant()
.annotatedWith(SharedPreferencesName.class)
.to("pl.project13");
```