Android Studio. Extra

Położenie: (nie dotyczy)

Informacje

Android Studio to nowsze (w stosunku do słabo rozwijanego *Eclipse*) środowisko programistyczne firmy Google wydane na licencji *„Apache 2.0”* (bezpłatne z możliwością komercyjnego zastosowania, ale daje firmie *Google* możliwość zmian licencji w każdym momencie bez uzasadnienia). Istnieje zarówno wersja na *Linux*, jak i *Windows*.
Android Studio wymaga także środowiska JDK. Może to być:

- **oracle-jdk-bin** (dobrze działa z wersją 1.8, w wersji 11 są problemy z ustawieniem w *eselect*);
- **icedtea** (zastępuje JDK 1.8);
- **openjdk** (zastępuje płatne do użytku komercyjnego JDK 11; daje darmowe wsparcie przez 6 miesięcy, także do użytku komercyjnego);
- **adoptopenjdk** (zastąepuje JDK 11; daje wsparcie LTS przez 4 lata);

Jeśli *Android Studio* nie wykryje ścieżki do JDK, można ręcznie ją ustawić w następującej lokalizacji: *File / Project Structure / SDK Location / JDK Location*: */opt/android-studio/jre* (lub */usr/lib/jvm/oracle-jdk-bin-1.8*).

Uwaga na licencję! Firma *Oracle* poinformowała, że od lutego 2019 aktualizacja JDK 8 możliwa będzie tylko dla **firm**, którzy wykupią komercyjną licencję (płatne wsparcie LTS [*Long-Term Support*]). Indywidualni użytkownicy będą mogli pobierać aktualizacje jedynie na własne potrzeby i tylko do końca 2020 roku. Alternatywą, na którą należy jak najszybciej się zdecydować, jest *OpenJDK* (GPLv2+CE [*Classpath Exception* --> nie musimy naszej aplikacji udostępniać na GPL]; budowane z tych samych źródeł), które będzie miało wsparcie LTS firmy *RedHat*. Używanie JDK 11 możliwe jest tylko w celach testowych, a komercyjne wykorzystanie jest nielegalne.

Po instalacji

- Zdefiniować sobie domyślną ścieżkę do projektów: *File / Settings / Appearance & Behavior / System Settings / Default Project Directory: ...*
- Zainstalować lub uaktualnić dodatkowe pakiety → *File / Settings / Appearance & Behavior / System Settings / Android SDK / SDK Platforms...* lub *Tools / SDK Manager*;
- Utworzyć virtualny telefon w emulatorze → *Tools / Device Manager* *(Uwaga: Jeśli projekt nie jest poprawnie otwarty, pozycja nie pojawi się w sekcji Tools)*
- Uaktywnić opcję automatycznych aktualizacji edytora → *File / Settings / Editor / General / Auto Import / Add unambiguous imports on the fly*

Struktura aplikacji

- *nazwaProjektu/src/main/res/layout/**activity_main.xml*** (lub ***content_main.xml***) – definicje wyglądu aplikacji; plik taki posiada na dole dwie zakładki: *Design* (tryb WYSIWYG) oraz *Text* (tryb tekstowy);
- *nazwaProjektu/src/main/java/com/example/nazwaProjektu/MainActivity/**MainActivity.java*** – co aplikacja ma robić;
- *C:\Users\robert\AndroidStudioProjects\ZamianaTekstu3\app\build\outputs\apk\apk-debug.apk* – zbudowana aplikacja;
- aplikacja może składać się z wielu plików *activity* (wielu screenów); nowe activity tworzymy poprzez wciśnięcie prawego przycisku myszy na *nazwaAplikacji/app/New/Activity*.

Podłączanie telefonu

Aby włączyć na telefonie tryb programisty (*developer options*) należy kliknąć 7 razy w:
Ustawienia / Informacje o urządzeniu (About phone) / Numer wersji kompilacji (Build Number)
Pojawi się wtedy następująca sekcja, którą należy zaznaczyć:
Ustawienia (lub System) / Opcje programisty (Developer options) / Debugowanie USB
Od tego momentu smartfon powinien być wykrywany w *Android Studio*.
Uwaga: Niektóre typy kabli USB-C nie potrafią przesyłać danych (służą jedynie do ładowania smartfone’a).

W starszych wersjach *Android Studio* należy najpierw włączyć opcję *Tools/Android/Enable ADB Integration* (lub wyłączyć i włączyć ponownie), uruchomić *Run/Debug* '*app*', a potem *Run/Run* '*app*'. W telefonie należy wybrać połączenie USB typu PTP (anulować typ MTP jeśli zostanie zaproponowany).

Kompilatory

Kod Java'y wymaga kompilacji, najczęściej za pomocą takich składników jak *Ant*, *Maven*, *Gradle*.

Emulator (wirtualny smartfon) w Linux

W Windows należy zainstalować funkcję Hyper-V (jeśli jest nieaktywna, należy uaktywnić w UEFI opcję Virtual Technology VT). Jeśli emulator nadal nie uruchamia się, należy zmniejszyć mu ilość pamięci RAM do 512MB lub przełączyć w tryb „Store a *snapshot for faster startup*". Należy także sprawdzić, czy nie uruchomił się poza ekranem (sprawdzić czy jest uruchomiony na pasku zadań).

Do działania emulatora w Linux, wymaga także modułu KVM (maszyna wirtualna, która koliduje z *VirtualBox*):
[*] Virtualization --->
<M> Kernel-based Virtual Machine (KVM) support
<M> KVM for Intel processors support
<M> Host kernel accelerator for virtio net
Uwaga: Wcześniej należy sprawdzić, czy procesor wspiera wirtualizację (wsparcie musi być włączone w BIOS):
lscpu
oraz
egrep -c '(vmx|svm)' /proc/cpuinfo
Jeśli wynik będzie wynosił „0" - nie ma wsparcia wirtualizacji. Jeśli powyżej „0" - jest wsparcie.
Aby maszyna działała, należy na ten czas wyładować moduły *VirtualBox* (jeśli takie są załadowane), choć nie zawsze jest to konieczne:
modprobe -r vboxnetadp vboxnetflt
Należy także sprawdzić, czy zwykły użytkownik ma dostęp do urządzenia */dev/kvm*:
ls -al /dev/kvm
Moduły odpowiedzialne za działanie *kvm* to:
modprobe kvm
modprobe kvm_intel
Jeśli zwykły użytkownik nie ma dostępu do urządzenia */dev/kvm*, należy ustawić w */etc/udev/rules.d/80_kvm.rules*:
KERNEL=="kvm", GROUP="kvm", MODE="0666" (dostępne dla wszystkich użytkowników)
KERNEL=="kvm", GROUP="kvm", MODE="0660" (dostępne tylko dla będących w grupie *kvm*)
Uwaga: Zagadką jest, że emulator został uruchomiony po skompilowaniu jądra bez modułu KVM (brak urządzenia */dev/kvm*). Prawdopodobnie użyty jest pakiet *qemu* (bez akceleracji, więc działa bardzo powoli).

Słownik

ADB - *Android Debug Bridge*, pozwala zarządzać emulatorem.
SDK - zestaw narzędzi do tworzenia aplikacji; jednym ze składników jest JDK.

Google Play

Koszt umieszczenia swojej aplikacji w sklepie *Play* wynosi 25 dolarów (oraz 30% ceny sprzedawanej aplikacji). Do tego należy doliczyć koszty utrzymania konta obsługującego płatności.
Konta *Google Play* zakłada się na stronie: *https://play.google.com/apps/publish/signup/*
Następnie należy podać numer karty kredytowej, nazwę banku, kod bezpieczeństwa. Po założeniu konta, przesyłamy plik *.apk i określamy, na jakie urządzenia i wersje Androida jest przeznaczony. Musimy także ustalić sprawy podatkowe.

Programowanie

Typy plików

***.java** - pliki z kodem źródłowym w języku Java;
***.class** - pliki z kodem bajtowym (pośrednim) skompilowane przez kompilator **javac**;
***.dex** - plik binarny powstały w wyniku kompilacji **.class* przez kompilator **dx** (do wersji Android 4.4, w oparciu o maszynę wirtualną *Delvik*); aplikacja jest kompilowana za każdym razem, gdy jest uruchamiana (oszczędzało to miejsce na karcie pamięci w początkowym etapie rozwoju rynku smartfonów);
***.elf** - plik binarny powstały w wyniku kompilacji **.class* przez kompilator **dex2oat** pod konkretny procesor (podczas instalacji aplikacji na smartfonie), procedura obowiązująca od wersji Android 5.0 w oparciu o maszynę wirtualną ART (*Android RunTime*);
***.apk** - paczka zawierająca między innymi plik **.dex* lub binarny plik **.elf* oraz inne pomocnicze;

Struktura kodu

Nadrzędnym składnikiem kodu jest **klasa**, wewnątrz której znajdują się **metody** (odpowiednik funkcji z innych języków programowania). Jeśli metoda nie ma zwracać wyniku, jest typu ***void***. Jeśli ma zwracać jakiś wynik, musimy w miejscu słowa „*void*" wpisać typ zwracanej zmiennej i zastosować na koniec kodu ***return***. Istotę metody można ująć w zasadzie „*Code reuse*", która brzmi: „*Write once, run it many times*".
Klasy i metody mają modyfikatory:
public - wszystkie klasy mają dostęp do pól danych i metod *public*;
private - dostęp do metod i pól danych posiadają jedynie inne metody tej samej klasy;
protected - używana jedynie przez metody swojej klasy oraz metody wszystkich jej klas pochodnych;
static – nierozrywalnie połączona z klasą, w której znajduje się; wywołanie tak zdefiniowanej metody musi być poprzedzone zawsze nazwą klasy, w której znajduje się, np. *nazwaKlasy.nazwaMetody();*
@override – zastąpienie (nadpisanie) poprzedniego działania (poprzedniej *Activity*) obecnym. Także nadpisanie domyślnych (wbudowanych w Androida) metod i zachowań.
Przykład deklaracji metody:

dostęp	*deklaracja*	*typ wyniku*		*typ*	*parametr*	
public	**static**	**int**	**nazwaMetody**	**(int**	**number)**	**{…}**

Metoda z parametrem „*View*" oznacza, że będzie uruchomiona po kliknięciu w widoczny element. Aby ją wywołać (ang. „*method call*") w innym miejscu kodu, należy wprowadzić:
nazwaMetody(null);

Wartość null

Wartość *null* może występować tylko w polach typu *string* (nigdy nie może być typu *integer*). Wartości *null* nigdy nie parsujemy, w szczególności nie będzie działać „*Integer.parseInt()*". Aby sprawdzić, czy pole ma wartość null powinniśmy stosować ==, a nie „*equals*". Metody sprawdzania null:

```
int wartoscPola = 0;
if (!TextUtils.isEmpty(nazwaSprawdzanegoPola.getText())) {
   wartoscPola = Integer.parseInt(nazwaSprawdzanegoPola.getText());
}
```

Inny sposób:

```
public boolean czyPoleJestPuste (EditText editText) {
   boolean result = editText.getText().toString().length() <= 0;
   if (result)
      Toast.makeText(context, "Wprowadzać jakąś liczbę!", Toast.LENGTH_SHORT).show();
   return result;
}
```

Jeszcze inny sposób:

```
public static void nazwaMetody() {
    try{
        String nazwaPola=null;
        System.out.println (str.length());
    }catch(NullPointerException e){
```

```
        System.out.println("Wystapil wyjatek");
    }
}
```

Zamiana Integer na String

```
int liczbaJakoInteger = -10;
String liczbaJakoString = String.valueOf(liczbaJakoInteger);
// lub
int liczbaJakoInteger = -10;
String liczbaJakoString = Integer.toString(liczbaJakoInteger);
```

Wyświetlanie liczb w polu tekstowym

```
mojTekst.setText("Oto liczba: " + String.valueOf(liczbaInteger)); // lub
mojTekst.setText("Oto liczba: " + Integer.toString(liczbaInteger));
```

Zamiana String na Integer

```
String liczbaJakoString = "1234";
int liczbaJakoInteger = Integer.parseInt(liczbaJakoString);
// lub
String liczbaJakoString = "1234";
int liczba = Integer.valueOf(liczbaJakoString);
```

Zmiana koloru tekstu

```
mojTekst.setTextColor(Color.parseColor("#ffff00"));
```

SharedPreferences

Prosty rodzaj bazy danych w formacie XML (klucz = wartość).

// Sprawdzanie, czy dana wartość istnieje

Nie trzeba sprawdzać. Korzystając z *getString*, podajemy drugi **zastępczy** parametr (jakąś wartość), na wypadek, jeśli klucza w ogóle by nie było.Zwracana jest wtedy właśnie ta wartość.
Można także wykorzystać:

```
contains(String key);
```

```
// ZAPISYWANIE KLUCZY w SharedPreferences
public void save(Context context, String text) {
    SharedPreferences ustawienia;
    Editor edytor;
    ustawienia = context.getSharedPreferences(PREFS_NAME, Context.MODE_PRIVATE);
    edytor = ustawienia.edit();
    edytor.putString(PREFS_KEY, text);
    edytor.commit();
}

// Odczytywanie KLUCZY z SharedPreferences
public String getValue(Context context) {
    SharedPreferences ustawienia;
    String text;
     ustawienia = context.getSharedPreferences(PREFS_NAME, Context.MODE_PRIVATE); //1
    text = ustawienia.getString(PREFS_KEY, null); //2
    return text;
}

// Usuwanie pojedynczego klucza
public void removeValue(Context context) {
    SharedPreferences ustawienia;
    Editor edytor;
    ustawienia = context.getSharedPreferences(PREFS_NAME, Context.MODE_PRIVATE);
    edytor = ustawienia.edit();
    edytor.remove(PREFS_KEY);
    edytor.commit();
}
```

```
// Czyszczenie całego SharedPreferences
public void clearSharedPreference(Context context) {
    SharedPreferences ustawienia;
    Editor edytor;
    ustawienia = context.getSharedPreferences(PREFS_NAME, Context.MODE_PRIVATE);
    edytor = ustawienia.edit();
    ustawienia.clear();
    ustawienia.commit();
}

// Sprawdzanie, czy aplikacja jest uruchomiona pierwszy raz
SharedPreferences preferences = getSharedPreferences("myprefs", MODE_PRIVATE);
if (preferences.getBoolean("firstLogin", true)) {
        SharedPreferences.Editor editor = preferences.edit();
        editor.putString("firstLogin", false);
        editor.commit();
}
```

Gdzie jest AndroidManifest.xml?

Plik ten przetrzymuje ogólne definicje na temat *Activity* wchodzących w skład aplikacji.
Ścieżka: *NazwaProjektu / app / src / main / AndroidManifest.xml*
Można do niego dodać na przykład możliwość zapisywania na zewnętrznych nośnikach (np. zapisywania *SharedPreferences*):
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
lub pozwolenie na robienie fotek:
<uses-permission android:name="android.permission.CAMERA"/>
lub blokowanie orientacji ekranu:
<activity
android:name=".MainActivity"
android:screenOrientation="portrait">
lub blokowanie zmiany konfiguracji przy zmianie trybu na landscape i odwrotnie:
<activity
android:name=".MainActivity"
android:configChanges="keyboardHidden|orientation|screenSize">

Tworzenie ikon programu

Stary sposób (projekty Java+XML):
http://romannurik.github.io/AndroidAssetStudio/
Launcher Icons / Image / Download .ZIP
Rozpakowaną zawartość skopiować do: *nazwaProjektu / src / main / res / **
Istnieje także pakiet domyślnych ikon Androida do pobrania:
http://commondatastorage.googleapis.com/androiddevelopers/design/
Android_Design_Icons_20131106.zip

Nowy sposób (projekty Kotlin + JetPack Compose):
1. Usunąć pliki:
- *folderProjektu / app / src / main / res / drawable / **
- *folderProjektu / app / src / main / res / mipmap**

2. Poniżej należy wybrać albo opcję „a” albo „b”:
a. Wczytanie ikon z plików **.xml*: *Resource Manager / + Image Asset /*
- *Foreground Layer / Path: Browse... /* własny plik: *ic_launcher_foreground.xml*
- *Background Layer / Path: Browse... /* własny plik: *ic_launcher_background.xml*

b. Wczytanie ikon z zasobów *ClipArt*: *Resource Manager / + Image Asset /*
- *Asset Type: ClipArt* (wybrać ikonę)

Dodatkowe ikony:
- https://romannurik.github.io/AndroidAssetStudio/index.html

Debugowanie

Aby debugowanie zatrzymało się na konkretnej linii, należy ją zaznaczyć tzw. Breakpoint.
Aby zacząć debugować już uruchomioną aplikację, należy wydać polecenie *Run / Attach debugger*

to Android process.

Komentowanie

W pliku XML (np. w *activity_main.xml*):
<!-- To jest komentarz... -->
W pliku Java (np. w *MainActivity.java*):
// To jest komentarz...

Błędy i Tips'y

Cannot reload AVD list

Podczas uruchomienia projektu w emulatorze, pojawia się komunikat: *„Cannot reload AVD list: cvc-enumeration-valid: Value '280dpi' is not facet-valid...”*. Należy uruchomić *Tools/SDK Manager* i usunąć zbędne pakiety (wymienione w komunikacie), m.in. *Android Wear ARM EABI v7a System Image, Android Wear Intel x86 Atom System Image*.

Unresolved reference... Conflicting overloads...

Powodem może być wybór języka Kotlin zamiast Java. Jeśli zmiana nie pomoże, należy zamknąć Android Studio i ponownie otworzyć.
Inna możliwość:
1. *Usunąć {folderProjektu}/.idea*.
2. *File / Sync Project with Grade Files*.

Jeszcze inna możliwość:
1. *Build / Clean Project*.
2. *Build / Make Projects*.

lub też:
1. *File / Invalidate caches / Restart*.

W ostateczności: utworzyć nowy projekt.

Unresolved reference: VERSION_1_8

W pliku *app / build.gradle.kts* jest:

```
compileOptions {
        sourceCompatibility = VERSION_1_8
        targetCompatibility = VERSION_1_8
    }
```

powinno być:

```
compileOptions {
        sourceCompatibility = JavaVersion.VERSION_1_8
        targetCompatibility = JavaVersion.VERSION_1_8
    }
```

app:compileDebugKotlin vs Java version

W pliku *app / build.gradle.kts*:

```
kotlinOptions {
        jvmTarget = "1.8" // Tu musi być ta sama wersja Javy, co w compileOptions
    }
```

Incompatible version (AGP 8.*)

*File / Project structure / Gradle Version: 8.**
oraz:
folderProjektu / Gradle Script / build.gradle.kts → plugins{... version ... 8.}*

Importowanie projektów

Projekt stworzony w systemie *Windows* nie będzie działał w systemie Linux (inne ścieżki dostępu). Adaptacja projektu z innego systemu nie jest sprawą prostą, jeśli w ogóle możliwą.

Reset Androida

Aby przywrócić smartfona do ustawień fabrycznych w przypadku, gdy zapomnieliśmy hasła (na przykładzie *Samsung Galaxy Tab 4*), należy wyłączyć smartfon, a następnie:

- przytrzymać jednocześnie trzy przyciski: *POWER* + *VolumeUp*+*HOME*;
- za pomocą przycisku *Volume Up/Down* wybrać opcję "*wipe data / factory reset*" i zatwierdzić przyciskiem *POWER*;
- następnie wybieramy opcję "*Yes - delete all user data*".

Could not find com.android.support:appcompat-v*

Należy otworzyć w projekcie plik *Gradle script* / *build.gradle* (Project: nazwaProjektu) i dodać odnośniki do *maven*:

```
allprojects {
    repositories {
        jcenter()
        maven { url "https://jitpack.io" }
        maven { url "https://maven.google.com" }
        mavenLocal()
    }
}
```

Dependency 'androidx.appcompat:appcompat-resources:' (lub: androidx.core:core-ktx:*) requires libraries and applications that depend on it to compile against version 34 or later of the Android APIs…***

Rozwiązanie: Edytujemy plik „*nazwaProjektu / app / build.gradle.kts*” i wpisujemy odpowiednią wersję SDK:

```
android {
    namespace 'com.example.nazwaProjektu'
    compileSdk 34
```

Następnie, należy wejść do: *File / Project Structure / Dependencies / app*. Przeklikać wszystkie składniki i sprawdzić czy któryś nie potrzebuje update’u.
Ustawić także odpowiednią wersję SDK:
File / Project Structure / Modules / Compile SDK Version: 34

Jak wyłączyć sprawdzanie pisowni w kodzie?

Wyłączyć opcję w: *File / Settings / Editor / Inspections / Proofreading / Typo*

Jak zmienić nazwę aplikacji?

Wyedytować plik: *folderProjektu / app / src / main / res / values / strings.xml*

Jakie jest minimalne SDK dla obecnego projektu?

File / Project Strukture / Modules / Default Config / Min SDK Version

Jak zrobić update SDK Platform?

Menu: *Tools / SDK Manager* / zakładka *SDK Platforms* → należy odznaczyć pozycję, która posiada „*Update Available*” i zaznaczyć jeszcze raz. Po lewej stronie pojawi się ikona *Update’u*.

Ostatnia aktualizacja: 15 lutego 2024.