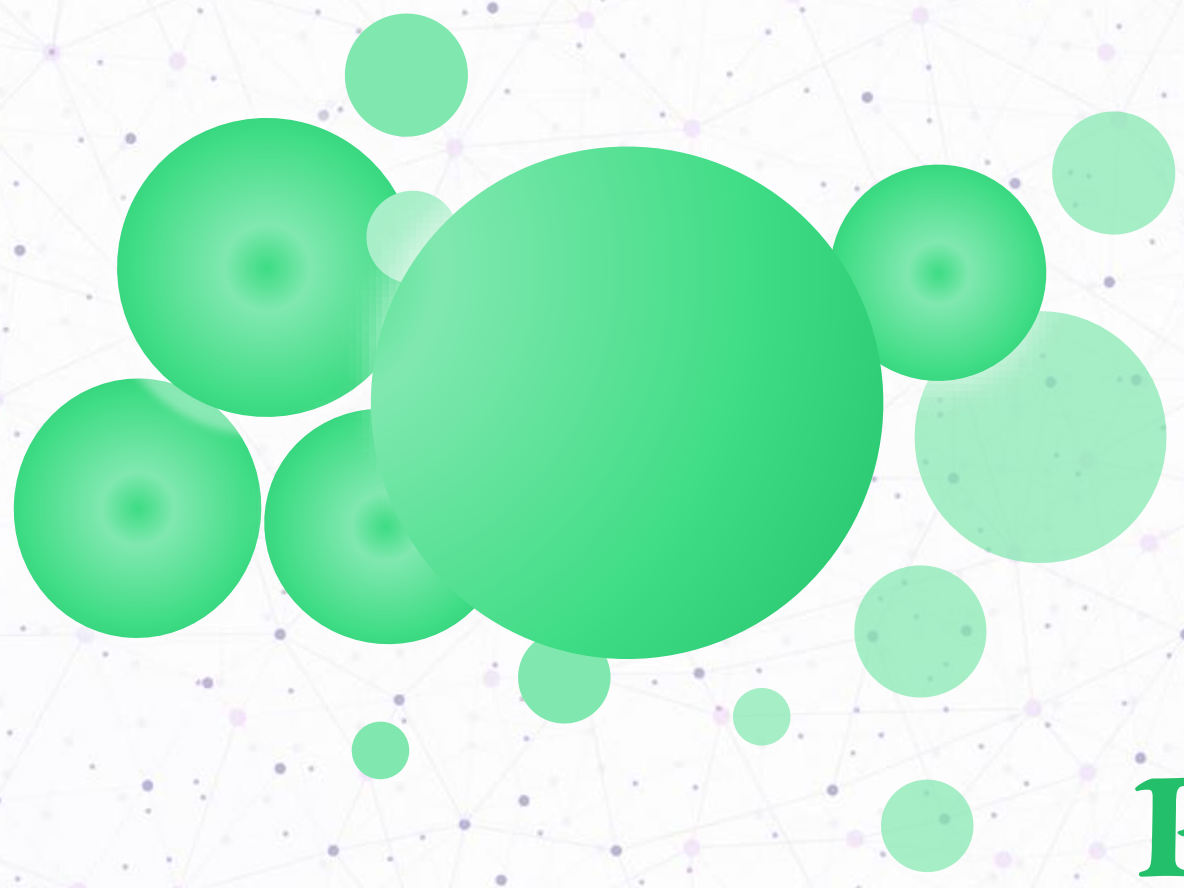

PV256
VÝVOJ NA ANDROID

Ing. Štefan Krajanec

1. prednáška – 20. 2. 2024

Ciele prednášky

- História Androidu
- Java × Kotlin
- Gradle
- Pozadie

História a vývoj Androidu

Začiatky Androidu

Významné verzie

Míľniky a inovácie

- **Prvá verzia:**

Prvá komerčná verzia Androidu, Android 1.0, vydaná v septembri 2008

Ďalej pravidelne vydané nové verzie, každá s novými funkciami a vylepšeniami.

- **Názvy verzií:**

Tradícia pomenovávať verzie po sladkostiach a dezertoch v abecednom poradí, ako sú Cupcake (1.5), Donut (1.6), Eclair (2.0),... až po Pie (9.0). Táto tradícia zmenená pri Android 10, kde Google prešiel na číselné označenie verzií.

História a vývoj Androidu

Začiatky Androidu

Významné verzie

Míľniky a inovácie

1. **Android 1.0 (API 1):** Základný systém s podporou webového prehliadača a Google Mapy.
2. **Android 1.5 Cupcake (API 3):** Prvé použitie názvu dezertu, widgety, video nahrávanie.
3. **Android 2.2 Froyo (API 8):** Zlepšená rýchlosť, podpora Flash, Wi-Fi hotspot.
4. **Android 4.0 Ice Cream Sandwich (API 14-15):** Holo dizajn, zlepšené multitaskingové funkcie.
5. **Android 5.0 Lollipop (API 21-22):** Material Design, zlepšené notifikácie, viacero užívateľských profilov.
6. **Android 6.0 Marshmallow (API 23):** Oprávnenia aplikácií na vyžiadanie, Google Now on Tap.

História a vývoj Androidu

Začiatky Androidu

Významné verzie

Míľniky a inovácie

7. **Android 7.0 Nougat (API 24-25):** Rozdelená obrazovka, zlepšené notifikácie, rýchlejšie prepínanie aplikácií.
8. **Android 8.0 Oreo (API 26-27):** Limitácia pozadia pre aplikácie, picture-in-picture mód.
9. **Android 9 Pie (API 28):** Adaptívna batéria a jas, gestové navigácie.
10. **Android 10 (API 29):** Tmavý režim, nové gestá, živé titulky.
11. **Android 11 (API 30):** Konverzácie a bubliny pre chat, jednorazové oprávnenia.
12. **Android 12 (API 31):** Material You, zlepšené súkromie a bezpečnosť.

História a vývoj Androidu

Začiatky Androidu

Významné verzie

Míľniky a inovácie

- **Akvizícia Google** (2005)
- **Uvedenie Androidu 1.0** (2008)
- **Spustenie Google Play Store** (2008)
- **Prvý telefón Nexus** (2010)
- **Uvedenie Material Designu** (2014)
- **Podpora pre rôzne zariadenia**
- **Zameranie na Bezpečnosť a Súkromie**
- **Uvedenie Android Go** (2017)
- **Uvedenie Kotlinu ako oficiálneho programovacieho jazyka** (2017)
- **Príchod Androidu 10 a novších verzií** (2019+)

1. **Rozšírenosť a komunita:**

Java je jeden z najpoužívanejších programovacích jazykov na svete s veľkou a aktívnou komunitou. To znamená obrovské množstvo knižníc, rámcov a nástrojov, ktoré sú k dispozícii pre vývojárov.

1. **Výkonnosť:**

V niektorých prípadoch môže byť Java rýchlejšia ako Kotlin, najmä pri použití optimalizovaných knižníc a kódu.

1. **Univerzálnosť:**

Java sa používa nielen pre vývoj Android aplikácií, ale aj v širokej škále iných oblastí, ako sú podnikové aplikácie, webové servery, middleware produkty, a ďalšie.

Kotlin

- **Bezpečnosť pred nulovými odkazmi:**

  Kotlin sa snaží vyhnúť problémom s nulovými odkazmi (NullPointerExceptions) prostredníctvom svojho systému typov.

- **Koncíznejší kód:**

  Kotlin umožňuje písať menej kódu na dosiahnutie rovnakých funkcií, čo znižuje šancu na chyby a zlepšuje čitateľnosť.

- **Interoperabilita s Javou:**

  Kotlin je navrhnutý tak, aby bol plne interoperabilný s Javou, čo umožňuje používať existujúce Javovské knižnice a rámce bez problémov.

- **Podpora moderných programovacích paradigiem:**

  Kotlin podporuje funkcionálne programovanie a má množstvo funkcií, ktoré Java nemá, ako sú rozšírené funkcie, delegáty, a **lambda** výrazy.

Java × Kotlin – zhrnutie

VLASTNOSŤ	JAVA	KOTLIN
Bezpečnosť pred nulovými odkazmi	Menej prísna, vyžaduje manuálne kontroly	Vstavaná podpora na úrovni jazyka
Koncíznosť kódu	Ukecaný kód	Menej kódu na dosiahnutie rovnakých funkcií
Interoperabilita s Javou	N/A (je to Java)	Plná interoperabilita
Funkcionálne programovanie	Obmedzená podpora	Plná podpora
Výkonnosť	Môže byť v niektorých prípadoch rýchlejšia	Porovnateľná, s možnými miernymi penalizáciami pri kompilácii
Univerzálnosť	Široko používaná v rôznych oblastiach	Hlavne zameraná na Android a webový vývoj

Java × Kotlin – príklady

1.

```java
String message = null;
if (message != null) {
    int length = message.length();
}
```

```kotlin
val message: String? = null
val length = message?.length
```

2.

```java
list.forEach(item -> System.out.println(item));
```

```kotlin
list.forEach { println(it) }
```

3.

```java
list.stream().filter(s -> s.startsWith("A")).forEach(System.out::println);
```

```kotlin
list.filter { it.startsWith("A") }.forEach { println(it) }
```

4.

```java
void display(String message) {
    display(message, false);
}

void display(String message, boolean important) {
    if (important) {
        System.out.println(message.toUpperCase());
    } else {
        System.out.println(message);
    }
}
```

```kotlin
fun display(message: String, important: Boolean = false) {
    if (important) {
        println(message.toUpperCase())
    } else {
        println(message)
    }
}
```

Java × Kotlin – príklady

```java
public class User {
    private String name;
    private int age;

    public User(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // Getters and setters
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
}
```

```java
public class StringUtil {
    public static String lastChar(String str) {
        return str.substring(str.length() - 1);
    }
}

// Použitie
String last = StringUtil.lastChar("Hello");
```

```kotlin
fun String.lastChar(): Char = this.get(this.length - 1)

// Použitie
val last = "Hello".lastChar()
```

```kotlin
data class User(val name: String, val age: Int)
```

```java
public String getUserName(User user) {
    if (user != null && user.getName() != null) {
        return user.getName();
    } else {
        return "Unknown"; // Predvolená hodnota
    }
}
```

```kotlin
fun getUserName(user: User?): String = user?.name ?: "Unknown"
```

Gradle × Gradlew

- správa závislostí
- konfigurácia zostavenia
- automatizácia úloh
- pluginy a rozšírenia

Gradle × Gradlew

- **Gradle**

Automatizácia zostavovania softwaru, ktorý sa široko používa v Java, Kotlin a Android projektoch.

Umožňuje vývojárom definovať konfiguráciu projektu pomocou skriptov, čo zjednodušuje a automatizuje procesy ako kompilácia kódu, balenie aplikácií, správa závislostí a nasadzovanie.

- **Gradle Wrapper**

Zabezpečuje, že všetci členovia tímu používajú rovnakú verziu Gradle, čím sa eliminujú problémy s kompatibilitou.

Gradle × Gradlew

Skript, ktorý umožňuje spustiť Gradle bez potreby jeho inštalácie na systéme.

Zabezpečuje, že všetci vývojári používajú rovnakú verziu Gradle.

- ./gradlew build
- ./gradlew clean
- ./gradlew app:dependencies

- Blok pre definíciu závislostí projektu **dependencies { ... }**
- Definuje repozitáre pre stiahnutie závislostí **repositories { ... }**
- Špecifické nastavenia pre Android projekty **android { ... }**

- **mavenCentral()**: Repozitár pre Java a Kotlin knižnice.
- **google()**: Repozitár pre Android knižnice od Google.
- iné

JVM (Java Virtual Machine)

Srdcom platformy Java a zabezpečuje jej kľúčovú vlastnosť: " Write Once, Run Anywhere " (napíš raz, spusti kdekoľvek). JVM je abstraktný výpočtový stroj, ktorý umožňuje počítaču spúšťať Java aplikácie (a aplikácie napísané v iných jazykoch, ktoré sú kompilované do Java bajtkódu).

Ako funguje?

- **Kompilácia:** Zdrojový kód Java (.java súbory) je najprv kompilovaný do bajtkódu (.class súbory) pomocou Java kompilátora.
- **Spustenie:** JVM následne načíta a spustí tento bajtkód, čím sa vykonáva aplikácia. JVM prekladá bajtkód na strojový kód príslušného operačného systému v reálnom čase pomocou procesu nazývaného Just-In-Time (JIT) kompilácia.

Výhody:

- **Platformová nezávislosť:** Aplikácie môžu bežať na akomkoľvek zariadení vybavenom JVM, bez ohľadu na jeho hardvér alebo operačný systém.
- **Bezpečnosť:** JVM poskytuje bezpečné spustenie aplikácií tým, že izoluje spustený kód od hostiteľského operačného systému.
- **Správa pamäte:** JVM automaticky spravuje pamäť aplikácií, najmä proces alokácie a uvoľňovania pamäte (garbage collection).

Dalvik a ART

Dalvik bol pôvodný virtuálny stroj používaný v Android OS na spustenie aplikácií.

- Od verzie Android 4.4 (KitKat) bol Dalvik postupne nahradený Android Runtime (ART).

Ako fungujú?

- **Dalvik:** Podobne ako JVM, Dalvik bol zodpovedný za spustenie aplikácií skompilovaných do formátu .dex (Dalvik Executable) bajtkódu. Dalvik používal Just-In-Time (JIT) kompiláciu na preklad bajtkódu na strojový kód.
- **ART:** ART zaviedol ahead-of-time (AOT) kompiláciu, pri ktorej sa aplikácie kompilujú do natívneho strojového kódu pri ich inštalácii. Tento prístup zlepšuje výkon aplikácií a efektivitu batérie, ale vyžaduje viac úložného priestoru.

Výhody ART oproti Dalvik:

- **Lepší výkon:** AOT kompilácia znižuje čas potrebný na spustenie aplikácií a zlepšuje ich plynulosť.
- **Efektívnejšia práca s pamäťou:** ART optimalizuje správu pamäte a garbage collection, čo znižuje spotrebu pamäte a zlepšuje výkon aplikácií.
- **Lepšia podpora pre novšie jazykové funkcie:** ART lepšie podporuje nové funkcie jazyka Java a Kotlin, čo umožňuje vývojárom využívať moderné programovacie paradigmy.

Ako to súvisí?

- **JVM**

Ako základ Java platformy položil základy pre koncept "Write Once, Run Anywhere", ktorý umožňuje aplikáciám bežať na rôznych platformách bez zmeny kódu. Tento koncept bol adaptovaný a špecificky prispôsobený pre mobilné prostredie prostredníctvom Dalvik a neskôr ART.

- **Dalvik**

Prvým krokom k prispôsobeniu Java virtuálneho stroja pre potreby mobilných zariadení s Androidom, zavádzajúc .dex formát a JIT kompiláciu pre efektívnejšie spustenie aplikácií.

- **ART**

Predstavuje ďalšiu evolúciu, ktorá priniesla významné zlepšenia vo výkone a efektivite aplikácií prostredníctvom AOT(ahead-of-time) kompilácie a lepšej správy pamäte. ART dnes umožňuje aplikáciám využívať najnovšie jazykové funkcie a poskytuje základ pre moderný vývoj Android aplikácií.

(Android 4.4-5.0 2014)

Prax

- **Android 4 (Dalvik):** Keďže Dalvik používa Just-In-Time (JIT) kompiláciu, váš kód sa kompiluje do natívneho strojového kódu až v momente spustenia. To znamená, že prvý štart aplikácie alebo spustenie určitých častí kódu môže byť pomalšie, pretože kompilácia prebieha v reálnom čase.
- **Android 6 (ART):** S ART a jeho Ahead-Of-Time (AOT) kompiláciou sa váš kód kompiluje do natívneho kódu už pri inštalácii aplikácie. To znamená rýchlejšie spustenie aplikácie a hladšie vykonávanie kódu, pretože kompilácia už bola vykonaná.

- **Android 4 (Dalvik):** Aplikácie môžu vyžadovať menej úložného priestoru, pretože kód sa ukladá v kompaktnejšom formáte bajtkódu a kompiluje sa len podľa potreby.
- **Android 6 (ART):** Aplikácie môžu vyžadovať viac úložného priestoru, pretože AOT kompilácia generuje viac natívneho kódu, ktorý sa musí uchovať na zariadení.

ĎAKUJEM ZA POZORNOSŤ