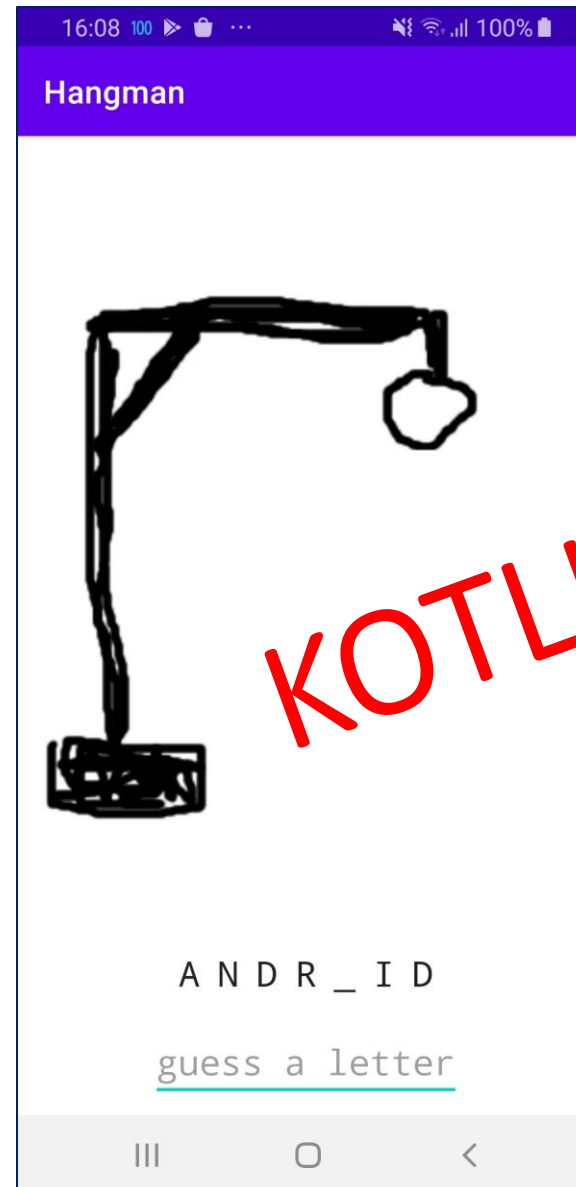




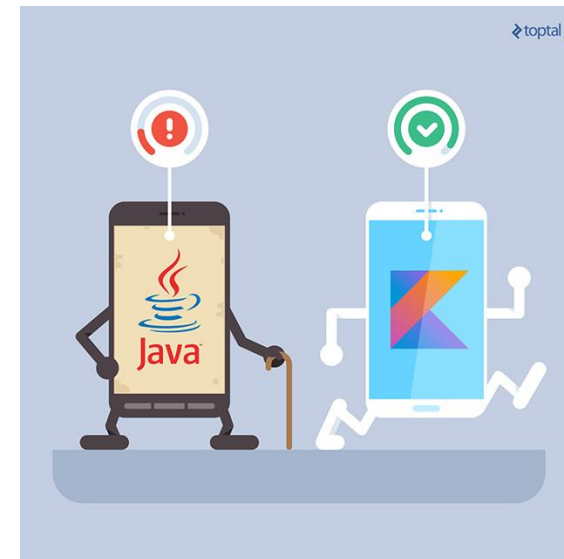
# Hangman

Aplikácia, ktorú sme už raz robili.  
Ale tentokrát v Kotlin.



# Kotlin

- **Android je Kotlin-first** od Google I/O 2019
- Kotlin od 2011, Java od 1995
- Funguje spolu s Javou
- Menej kódu, lepšia čitateľnosť, bezpečnejší kód
- Android, iOS backend, web apps



# Kotlin na prvý pohľad

- Menej ukecané (POJO, Data Class)
- Syntax - žiadna bodkočiarka
- Typy premenných – nie vždy explicitne povedané
- == vs === (v Java equals a ==)
- String literály (a mnoho iného neskôr)
- Nový objekt bez new

```
public class Student {
    private int rollNo;
    private String firstName;
    private String lastName;

    public int getRollNo() {
        return rollNo;
    }

    public void setRollNo(int rollNo) {
        this.rollNo = rollNo;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
}
```

```
data class Student(var rollNumber: Int, var firstName: String, val lastName: String)
```

# #1 - Preskúmanie existujúceho kódu

- Meníme iba Java kód za Kotlin. Resources ostávajú nezmenené.
- Rozširovanie tried
- Definovanie metód
- Null safety

# Rozširovanie tried

- Aj v Jave vznikali triedy **iba** rozširovaním
- Java: Object
- **Kotlin: Any (equals, hashCode, toString)**
- Rovnako sa implementuje aj interface (viac rozhraní oddelíme čiarkou) - podľa zátvoriek (volania konštruktora) sa dá rozpoznať rozdiel medzi rozširovaním a implementovaním.
- Súvisiace veci na neskôr: dedičnosť a keyword open, deklarovanie konštruktora

# Definovanie metód

- Funkcia **fun** keyword
- **Override** už nie iba ako anotácia
- Definovanie premenných
  - Názov: Typ
  - Default arguments (**a: Int = 0**)
  - Named arguments - pri zavolaní funkcie
- Návratový typ
  - Píše sa na konci
  - Java – void, Kotlin – Unit (**nemusí sa písať**)
  - Unit – typ s jedinou hodnotou Unit

# Null safety ?

- Menšia početnosť **NullPointerException** a viac pod kontrolou
- premenná **surprise** - nemôže byť null  
(kompilačná chyba ak by sme sa o to pokúsili)
- premenná **surprise?** - môže byť null
  - volanie `surprise.length` nepovolené
- **Safe call - ?.**  
**surprise?.length** vráti hodnotu alebo null – ale nevyhodí výnimku
- Budeme sa tomu venovať podrobnejšie neskôr ...

## #2 – Rozhranie v Kotline

- Prepíšeme interface
- Premenné (val, var) a ich typy
- Companion object



# premenné

- **var** - mutable
- **val** – read-only
- Typy premenných
  - Kotlin nepoužíva primitívne typy (iba interne po kompilácii)
- Int, Long, Double, Boolean, Char, String ...
- Nie vždy je potrebné písať typ premennej

# Companion object

- Kotlin **class** - pre triedy, ktoré môžu mať viac inštancií
- Kotlin **object** – anonymné triedy, singletony
- **companion object**
  - pridružený k nejakej triede
  - Funkcie a premenné - ako static v Java
  - Môže byť pomenovaný
- **Const** – pre hodnoty známe v čase kompilácie, ktoré sa nebudú meniť
  - Len pre Stringy a *primitívne typy*
- V kotline je možné písať kód aj mimo triedy

## #3 – Implementácia rozhrania

- Prepisujeme java kód do kotlinu. Konštruktor, premenné, metódy.
- Konštruktory
- Polia
- For cyklus
- Single expression funkcie
- Porovnávanie (== vs. ===)

# Konštruktory

- **1 primárny** v hlavičke triedy
- keyword **constructor** len pri zmene viditeľnosti  
`class Person(firstName: String)`
- Primárny konštruktor nemá kód - **init** blok (aj na viacerých miestach - vykonáva sa postupne)
- Dajú sa deklarovat' inštančné premenné (**val/var**)
- Ľubovoľne veľa **sekundárnych** (+ keyword)

# Kolekcie

- List – iba immutable operácie
- MutableList a Array poskytujú mutable operácie
- Prístup k prvkom v liste cez []
  
- **Array** (zodpovedá poľu v jave)
  - arrayOfNulls()
  - arrayOf()
  - IntArray, intArrayOf()
  
- Veľa užitočných funkcií v porovnaní s Javou
- Lambda a funkcionálny prístup

# cykly

- Keyword **in** (ako for-each)

```
for (item in collection)
```

```
for (i in 1..3)
```

```
for (i in array.indices)
```

```
for (i in 6 downTo 0 step 2)
```

# Single expression funkcie

- Ak funkcia **vráti iba výraz**, môže byť vrátený priamo cez = namiesto bloku kódu

```
fun double(x: Int): Int = x * 2
```

- Návratový typ je **voliteľný** - ak to vie kompilátor doplniť
- Funkcie (s blokom kódu) musia definovať návratový typ, inak sa doplní Unit

# Stringy a porovnavanie

== VS ===

- **Structural** equality  
== porovnáva objekty (ako equals v jave)
- **Referential** equality  
=== porovnáva referencie (ako == v jave)



## #4 – Kotlin nie je len prepísaná Java

- Android Studio umožňuje konvertovať Java kód do Kotlinu
- Deklarovanie premenných, gettre, settre
- Properties

# Interface s premennými

- Interface v Kotlin - veľa nových možností
  - mnoho podnetov pre OOP teoretikov (private pri premenných/metódach)
- Dajú sa deklarovať **premenné**
  - Namiesto funkcie, ktorá je reálne getter premennej
  - V implementácii **override**
- *Android Studio vie konvertovať Java->Kotlin*

# Gettre a settre

- zopakujme si: Java, inštančná premenná, zapúzdrenosť
- **Field** v Kotline je len časťou **Property** na uchovávanie hodnoty
- Val property - viditeľný getter
- Var property - viditeľný getter a setter
  - **private set** nastaví neviditeľnosť
- **Getter a setter vieme prispôbiť: `get()` =**
- private val nie je nutnosť kvôli zapúzdreniu
  - val/var definujú property, nie inštančnú premennú

# #5 – Aktivita v Kotlin

- Lateinit
- Null safety
- Pretypovanie
- Scope funkcie

# lateinit

- findViewById a widgety
  - Ak dáme `?` tak musíme riešiť `null` hodnotu – widgety sa inicializujú až v `onCreate()`
  - Ak nedáme `?` - akú hodnotu nastaviť?
- **lateinit** dovoľí inicializovať neskôr
- Mení sa hodnota - musí byť **var**
- Nefunguje pre *primitívne typy*
- `::textView.isInitialized` - Reflection

# Null safety (zase)

- Klasická podmienka ( $x \neq \text{null}$ ) - ok pre immutable
- Safe call ( $?.$ ) + chain
  - **surprise?.length** - vráti hodnotu alebo null
- Elvis operator ( $?:$ )
  - **surprise?.length ?: 0** - vráti hodnotu alebo 0
- Null check ( $!!$ ) + konvertuje na non-null typ
  - **surprise!! .length** - vráti hodnotu alebo výnimku
- Jeden (!) môžeme sa rozhodnúť
  - **Toast T!** Znamená **T** alebo **T?**



# random

- Inrange má random funkciu (a mnoho iných fun)
  - `(0..words.size-1).random()`
- **Random.Default**
  - companion object
  - defaultná implementácia
- `Random.asJavaRandom()` - vráti `java.util.Random` ale s kotlinovským generátorom

# Pretypovanie

- **is** – overenie typu (obj is String)
- **Smartcast** – v Android Studio zvýraznené zelenou  
`if (x is String && x.length > 0) ...`
- Smartcast iba pre **val**
- **Unsafe cast** - môže hodiť výnimku `x as String`
- **Safe cast** - môže hodiť null `x as? String`



# Scope funkcie

- Vykonajú kód v kontexte nejakého objektu - vytvorí dočasný scope

1. **let**

2. **run**

3. **with** - *with this object, do the following*

4. **apply**

5. **also**

- Viac v dokumentácii
- Vyhneme sa vyrábaniu novej premennej
- Context object: **this** alebo **it**