6. Az alábbi kódot valaki a Kotlin nyelv alapos megismerése nélkül készítette el. Jelöljön és javítson ki a kódban minél több hibát! (6 pont)

```
TODO("Hibás kódos feladat példájának kitalálása a minta ZH-hoz is...")
```

Kotlin alapú szoftverfejlesztés

Zárthelyi Dolgozat **MINTA**

2019. ősz

1. feladat (10 pont)	
2. feladat (5 pont)	
3. feladat (5 pont)	
4. feladat (9 pont)	
5. feladat (15 pont)	
6. feladat (6 pont)	
∑ (50 pont)	

Név:……

Neptun kód: ……

A feladatok megoldására 90 perc áll rendelkezésre. Csak ezen a feladatlapon dolgozhat, ha elfogyna a megoldásra fenntartott hely, akkor kérjen még feladatlapot! A magyarázatoknál kerülje a sablonos kifejezéseket, fogalmazzon pontosan.

1. feladat A következőkben állapítsa meg az állítás igaz voltát. Minden helytelen válasz 1 pont levonással jár. A feladatban elért összes pontszáma nem lehet negatív. (10 pont)

Kotlin és Java kód között kétirányú interoperáció létezik, mindkét nyelvből át lehet hívni a másikba.	
A `TODO()` használata mindig kivétel dobásához vezet.	
A Java kódból érkező értékek mindig *nullable*-nek számítanak, mivel bármikor lehet `null` az értékük.	
A Kotlin egy statikusan típusos nyelv.	
A `Sequence`-ek használata mindig hatékonyabb, mint az egyszerű `Collection`-ök (például `List`) használata.	
Extension function-ök segítségével csak a saját típusainknak adhatunk új funkcionalitást.	
Az alapértelmett láthatóság a `public`.	
A függvényeknek kötelező megjelölniük, hogy milyen kivételeket dobhatnak.	
Ha a JVM (Java Virtual Machine) felett használjuk, a Kotlin kód Java-ra fordul.	
A `Unit` típusnak nem léteznek értékei.	

2. feladat Karikázza be a helyes válasz betűjelét! Minden kérdésre csak egy helyes válasz van. (5 pont)

1. Melyik cég hozta létre a Kotlin nyelvet?
   A) Microsoft B) Google C) JetBrains D) Oracle

2. Melyik kulcsszó jelzi, hogy egy függvényt kötelező implementálni a leszármazottnak?
   A) `open` B) `final` C) `override` D) `abstract`

3. Hány `List` létrehozás történik a következő kódrészetben?
```
listOf(1, 2).map{ ... }.filter{ ... }
```
   A) 0 B) 1 C) 2 D) 3

4. Melyik nem *expression* Kotlinban, azaz melyiknek nincs visszatérési értéke?
   A) `if-else` B) `return` C) értékadás D) `try-catch`

5. Melyik művelet nem vezethet kivételhez?
   A) `as?` B) `!!` C) `string.toInt()` D) `as`

3. feladat Mit írnak ki az alábbi kódrészletek? A kód kimenetét írja a kód mellé.

1. Első részfeladat (2 pont)

```
data class Person(val name: String, val age: Int)
val carla = Person("Carla", 36)
val (name, age) = carla
println("$name is $age years old")
```

2. Második részfeladat (3 pont)

```
val names = listOf("Sam", "Sue", "Jim")
names.filter { it.startsWith("S") }
      .asSequence()
      .map { it.length }
      .first()
      .let {
            print(it)
      }
```

4. feladat Válaszoljon az alábbi elméleti kérdésekre!

1. A `Sequence` és a `List` közül melyik lehet végtelen hosszú, és miért? (2 pont)

2. Milyen költsége van egy lambda átadásának paraméterként, és hogyan lehet ezt elkerülni? (2 pont)

3. Mi a `Unit` típus? Hány értéke van, mi a jelentése? Mire használjuk? (3 pont)

4. Mi a különbség az `as` és az `as?` operátorok között? Melyik mit csinál? (2 pont)

5. feladat Implementálja az alább specifikált függvényeket és kódrészleteket.

1. Írjon egy `lastChar` nevű *extension function*-t, ami egy `String` utolsó karakterét adja vissza.
   Használata: `"foo".lastChar()`, amire a kimenet: `'o'`. A hibakezeléssel nem kell foglalkoznia. (3 pont)

2. Hozzon létre egy `Map<String, Int>` példányt, amiben a `"Jim"` kulcshoz a `7`, a `"Katie"` kulcshoz pedig a `10` érték tartozik. (2 pont)

3. Implementálja a `let` függvényt. Ez a függvény tetszőleges generikus típuson meghívható *extension*-ként. Paramétere egy függvény típus, ami ugyanezt a generikus típust várja saját paramétereként, és visszatér egy másik generikus típussal. A `let` függvény lefuttatja a paramétereként kapott függvényt, átadva neki a *receiver*-ét, és a függvény által adott értékkel tér vissza. Alább látható egy példa a függvény használatára. (4 pont)

```
23.let { (it * 2).toString() } // "46"
"hello".let { str -> str + "world" } // "hello world"
```

4. Írja át az alábbi kódrészletet úgy, hogy az `apply` függvényt használja az inicializálás elvégzésére. (2 pont)

```
val square = Square()
square.x = 10
square.y = 20
square.side = 50
```

5. Implementáljon egy `Int` típusú property-t, ami mindig kétszer annyit ad vissza, mint amennyit ténylegesen eltároltunk benne. (2 pont)

6. Írjon egy *teljes Kotlin programot*, ami kiírja a standard kimenetre a `"Hello world"` stringet. (2 pont)