Szegedi Tudományegyetem
Természettudományi és Informatikai Kar

Informatikai Intézet
Számítógépes Optimalizálás Tanszék (?)

Android oktatóalkalmazás Java programnyelvhez

Diplomamunka

Készítette: Gáspár Tamás
Programtervező Informatikus MSc hallgató

Témavezető: Dr. Csendes Tibor
Egyetemi tanár
Számítógépes Optimalizálás Tanszék

Szeged, 2021

Tartalomjegyzék

1. Bevezető

...

2. A diplomamunka célja

A cél egy olyan Android eszközökön futó alkalmazás, ami segítséget nyújt a *Java* programozási nyelv megtanulásához. Ehhez két komponensre van szükség: a konkrét alkalmazásra, mely képes valamilyen standard formában megadott tananyagot megjeleníteni, és magára a tananyagra.

2.1. Követelmények a tananyaggal kapcsolatban

A tananyagnak strukturáltnak kell lennie, hogy a felhasználó könnyen tudjon navigálni benne. A programozás alapjaitól kell kezdődnie, hogy teljesen kezdők számára is használható lehessen. Fontos, hogy a tananyag csak olyan ismeretekre hivatkozzon, amik már a korábbi fejezetekben be lettek mutatva. Lenniük kell számonkéréseknek is, amelyekkel a felhasználó megbizonyosodhat róla, hogy megértette-e az anyagot.

2.2. Az alkalmazás követelményei

Az alkalmazásnak képesnek kell lennie, hogy a tananyagot a struktúráját megőrizve megjelenítse. A programozás oktatásánál elengedhetetlenek a kódminták, ezeket formázottan és könnyen olvashatóan kell mutatni (tabulálás, színezés). A felhasználó által kitöltött számonkéréseket ki kell tudni értékelni, a helyes és helytelen válaszokat pedig jelölni.

3. A tananyag struktúrája

A tananyagot kurzusokra, fejezetekre, feladatokra és vizsgákra osztottam. A következő alfejezetekben részletesen leírom, hogy ezek mit takarnak.

3.1. XML

Az *XML* (*Extensible Markup Language*) egy jelölőnyelv, ami alkalmas strukturált adattárolásra. Az *XML* fájlok egymásba ágyazott *tag*ekbből állnak.

Egy példa *XML*-ben megadott adatra, ami egy azonosító és egy név *tag*et tartalmaz:

```
<root>
  <id>33</id>
  <name>XML</name>
</root>
```

A beépített *Android* erőforrástípusok *XML*-t használnak, ezért számomra is természetes volt, hogy ebben kódolom el a tananyag erőforrásfájljait.

A továbbiakban az *XML* dokumentumokban azokat a részeket, ahol nem konkrét adatot kell érteni [és] jelek közé fogom írni, ezáltal általánosabban adhatom meg a struktúrákat. A fenti *XML* így átírva:

```
<root>
  <id>[Azonosító]</id>
  <name>[Név]</name>
</root>
```

3.2. Kurzus

A kurzus a legnagyobb egység, a *Java* nyelv egy nagy területéhez kapcsolódó ismereteket tartalmazza. Ezek az ismeretek lehetnek egyszerűek (a tananyag elején), vagy magas szintűek, amelyek már a korábbi kurzusokra épülnek (jellemzően a későbbi kurzusok).

Külön kurzusba vettem például az adatszerkezeteket, vagy a generikus programozást. Természetesen ezek a területek önmagukban is rengeteg ismeretet tartalmaznak, ezért további felosztásra van szükség, ezért bevezettem a fejezeteket.

Minden kurzus tartalmaz egy nevet és egy egyedi azonosítószámot, méghozzá úgy, hogy ezek a számok növekvő sorrendbe állítva meghatározzák a kurzusok sorrendjét.

Ezen kívül minden kurzus tartalmazza, hogy mely fejezetek (3.3), feladatok (3.4) és vizsga (3.5) tartozik hozzájuk.

A fentiek alapján egy kurzust leíró *XML* fájl a következőképpen néz ki:

```
<resources>
  <coursedata>
    <id>[Kurzus azonosító]</id>
    <name>[Kurzusnév]</name>
```

```
    <finished>[Befejezettség jelölő]</finished>
  </coursedata>

  <chapter>[Fejezet azonosító]</chapter>
  ...
  <chapter>[Fejezet azonosító]</chapter>

  <task>[Feladat azonosító]</task>
  ...
  <task>[Feladat azonosító]</task>

  <exam>[Vizsga azonosító]</exam>
</resources>
```

Arról, hogy mit takar a befejezettség jelölő és a *<finished>* tag, a 3.7 alfejezetben írok.

3.3. Fejezet

A kurzus nem osztatlan egység, hanem fejezetekből épül fel. Ezek további kisebb, összefüggő részekre bontják a kurzus tartalmát, hogy az átláthatóbb és könnyebben elsajátítható legyen.

Például az adatszerkezetek kurzus fejezetei a következők:

1. Tömbök.
2. A tömbök hátrányai.
3. Listák.
4. Verem és sor.
5. Szótár.
6. Hasznos osztályok adatszerkezetek kezelésére.

A listából látszik, hogy vannak átkötő fejezetek is, melyek elmagyarázzák, hogy miért fontosak a korábbi, vagy későbbi fejezetek ismeretei. Például a *Tömbök* fejezet után a felhasználóban felmerülhet, hogy miért kell további adatszerkezetekkel foglalkozni. Ezért közbeiktattam egy plusz fejezetet, ami

részletesen megmagyarázza, hogy milyen hátrányai vannak a tömböknek és hogy mely esetekben érdemes más adatstruktúrát választani.

A fejezetek is rendelkeznek azonosítóval és névvel. Ezen kívül tartalmazzák a fejezet törzsét, ami majd konkrétan megjelenik a felhasználónak, amikor megnyitja az adott fejezetet.

```
<resources>
  <chapterdata>
        <id>[Fejezet azonosító]</id>
    <name>[Fejezetnév]</name>
  </chapterdata>

  [Fejezet törzse]

</resources>
```

Arról, hogy mi tartozhat a törzsbe, és ezek hogyan vannak elkódolva, a 4. részben írok részletesen.

3.4. Feladat

A feladat egy önálló programozási munkát takar, ami mindig egy kurzushoz kapcsolódik. Ahhoz, hogy a felhasználó meg tudja oldani, szüksége lesz az adott kurzus és az azt megelőző kurzusok ismereteire is.

Egy kurzushoz több feladat is tartozhat, de mindegyikhez adott legalább egy. Továbbra is maradva az adatszerkezetek kurzusnál, ehhez például két feladatot mellékeltem:

- Tömb alapú lista implementálása.
- Láncolt lista implementálása.

Megjegyzem, hogy ezen feladatok megoldásának nem kell olyan hatékonynak vagy általánosnak lennie, mint egy valódi implementáció, a cél csak a két lista alapelveinek megértése. Mivel a generikus programozás kurzus az adatszerkezetek után következik, ezért itt természetesen nem elvárás generikus listák programozása.

Egy-egy feladatnak olyan sokféle helyes implementációja van, hogy lehetetlen ellenőrizni, hogy a felhasználó megoldása megfelelő-e. Ennek ellenére

szerettem volna biztosítani, hogy ha a felhasználó elakad, vagy csak összevetné a megoldását egy másikkal, akkor erre lehetősége legyen. Minden feladat mellé referencia implementációt mellékeltem, ami az adott feladat alatt megtekinthető.

A feladatok *XML* elkódolása így néz ki:

```
<resources>
  <taskdata>
    <id>[Feladat azonosító]</id>
    <name>[Feladatnév]</name>
  </taskdata>

  [Feladat törzs]

  <solution>

  [Referencia implementáció törzs]

  </solution>
</resources>
```

Arról, hogy mi tartozhat a törzsbe, és ezek hogyan vannak elkódolva, a 4. részben írok részletesen.

3.5. Vizsga

A vizsgák a tananyag valódi számonkérései, itt ugyanis a feladatokkal ellentétben lehetséges a megoldások pontos ellenőrzése. Minden kurzushoz egy vizsga tartozik, amelyben benne lehet bármi az adott kurzus fejezeteiből.

Azért, hogy az alkalmazás a válaszokat ellenőrizni tudja, megkötéseket kell tenni a kérdések típusára. Nem lehet például esszékérdés. Ezt figyelembe véve a következő kérdéstípusokat implementáltam:

- Feleletválasztós kérdés, egy választási lehetőséggel (*single choice*).
- Feleletválasztós kérdés, több választási lehetőséggel (*multi choice*).
- Szöveges kérdés, ahol a válasz egy szó vagy rövid kifejezés lehet.
- Igaz-hamis kérdés.

A vizsgákhoz kérdéshalmaz tartozik, melyből az alkalmazás véletlenszerűen választ annyi, amennyi kérdésből az adott vizsga áll (jellemzően 25-30 darabot). Időlimit is van, ami alatt a felhasználónak be kell fejeznie a kitöltést. Az idő lejárta esetén a kitöltés akkori állapota kerül kiértékelésre.

A vizsgákat definiáló *XML* fájlok a következőképpen néznek ki:

```
<resources>
  <examdata>
        <id>[Vizsga azonosító]</id>
        <questionAmount>[Kérdések mennyisége]</questionAmount>
        <timeLimit>[Időkorlát, perceben]</timeLimit>
    <finished>[Befejezettség jelölő]</finished>
  </examdata>

  [Vizsgakérdések]

</resources>
```

Arról, hogy mit takar a befejezettség jelölő és a *<finished>* tag, a 3.7. alfejezetben írok. A kérdéstípusok részletes definiálása a 4.5. részben történik.

3.6. Előrehaladás a tananyagban

Az alkalmazás kezdeti indításakor a felhasználó csak az első kurzus fejezeteihez és feladataihoz fér hozzá. A vizsga zárolva van. El kell olvasnia az összes fejezetet ahhoz, hogy a vizsga kitölthető legyen. A feladatok elkészítése opcionális, de ajánlott.

A vizsga sikeres teljesítésével nyitható meg a következő kurzus, majd a folyamat ismétlődik addig, amíg van további kurzus.

Korábbi fejezetek, feladatok, és vizsgák bármikor újra megtekinthetőek és kitöltetőek.

3.7. Részben befejezett kurzusok

4. Törzstartalom, kódminták és kérdések

Már láttuk a tananyag struktúráját és azt, hogy ez hogyan van *XML*-ben elkódolva. Most bemutatom, hogy miként vannak definiálva azok a komponen-

sek, amelyeket a felhasználó a fejezetek, feladatok és vizsgák megnyitásakor lát és amelyek a tananyag törzsét alkotják.

4.1. Szöveges tartalmak

Egyszerű szöveg és cím tag leírása ide.

4.1.1. Bekeretezett szöveg

...

4.1.2. Magas szintű tartalom

...

4.2. Grafikus tartalmak

Kép beillesztés leírása ide.

4.3. Kódminták

...

4.4. Interaktív kódminták

...

4.5. Kérdések

...

4.5.1. feleletválaszós kérdés egy válaszlehetőséggel

...

4.5.2. feleletválaszós kérdés több válaszlehetőséggel

...

4.5.3. Szöveges kérdés

...

4.5.4. Igaz-hamis kérdés

...

5. A tananyag tárolása

A tananyagot a forráskódtól el kell különíteni, viszont futásidőben elérhetőnek kell lennie, hogy a felhasználó kéréseit ki lehessen szolgálni. Fontos az egységes elkódolás. Például minden kurzust leíró erőforrásfájlnak azonos felépítésűnek kell lennie: meg kell mondaniuk mely fejezetek, feladatok és vizsga tartozik hozzájuk.

5.1. Android erőforráskezelés

Az *Android* alapelve, hogy az erőforrásfájlok legyenek elkülönítve a kódtól, és erre több eszközt is kínál. Az első a *Resource System*. Ebben olyan erőforrásfájlok tárolhatóak, melyekre a legtöbb alkalmazásnak szüksége van:

- Grafikus elemek (*drawable*): lehetnek ikonok, képek, hátterek, stb.
- Szövegek (*string*): a felhasználói felületen megjelenő feliratok.
- Elrendezések (*layout*): a felhasználói felület elrendezését tárolják.
- Még sok további előre definiált kategória: stílusok, színek, stb.

A *Resource System* nem a legmegfelelőbb mód a tananyag tárolására, mivel az egyetlen gyakori, előre definiált kategóriába sem esik.

Biztosított továbbá az *Asset System*, mely annyiban tér el az előbb bemutatott rendszertől, hogy itt nincsenek kategóriák, be tud fogadni bármilyen erőforrásfájlt, amelyre egy alkalmazásnak szüksége lehet. Ez a legalkalmasabb módszer a tananyag tárolására.

6. Adatbázis

A 5 fejezetben említett *Resource System* és *Asset System* futásidőben csakis fájl olvasást tesznek lehetővé, írást nem. Ezért az alkalmazás egy kisebb relációs adatbázist használ arra, hogy a felhasználó előrehaladását tárolja.

6.1. Struktúra

...

6.2. Megvalósítás: SQLite és Room

...