Programozási technológia

Adatbáziskezelés (JDBC, Swing)

Dr. Szendrei Rudolf

ELTE Informatikai Kar

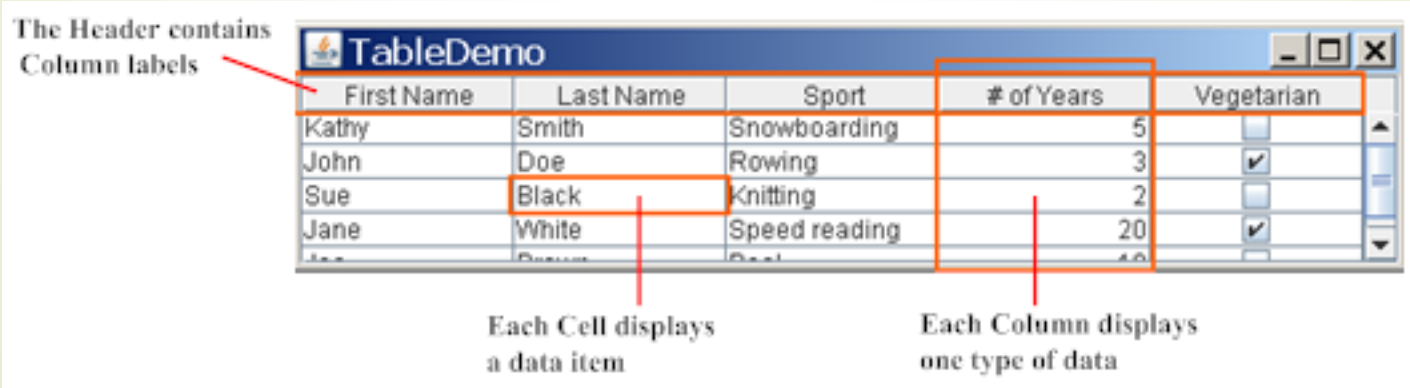
2018.

További szükséges komponensek

- JTable
- JComboBox
- JScrollPanel
- JSlider
- JPasswordField

JTable

- Adatok táblázatos formában való megjelenítésére alkalmas.
- Opcionálisan editálható.
- A JTable nem tartalmazza a megjelenített adatokat, az adatoknak csak egy nézete.



JTable feltöltése adatokkal

- A táblázat létrehozása történhet közvetlenül az adatok és oszlopnevek megadásával, melynek hátrányai, hogy
  - Ilyenkor a táblázat minden cellája editálható
  - Minden adattípus String-ként kezelt.
  - A tömböt össze kell állítani...

```
String[] columnNames = {"column1", "column2",...};
Object[][] data = {{value1,value2,...},
                   {value1,value2,...}};

JTable table = new JTable(data, columnNames);
```

JTable feltöltése adatokkal

- Minden táblázathoz tartozik, egy a tényleges adatokat tartalmazó objektum, amely implementálja a `TableModel` interfészt. Ez alapesetben egy `DefaultTableModel` példány.
- Az adatokat rugalmasabban tudjuk kezelni, ha ezt a modelt mi magunk definiáljuk. Ehhez származtatnunk kell az `AbstractTableModel` osztályból, és megvalósítanunk annak metódusait.
- A mi esetünkben az adatok egy `ArrayList`-ben érkeznek, aminek minden eleme egy `HighScore` típusú objektum lesz (ez tárolja a pálya nehézségi fokozatát/szintjét és a lépésszámot).

TableModel metódusai

`int getRowCount()`	sorok száma*
`int getColumnCount()`	oszlopok száma*
`getColumnName(int col)`	oszlop neve
`Class getColumnClass(int col)`	oszlop típusa
`boolean isCellEditable(int row, int col)`	szerkeszthető-e a cella.
`Object getValueAt(int row, int col)`	a cella értéke*
`setValueAt(Object val, int row, int col)`	cellaérték beállítása

* A pirossal jelölt metódusok implementálása kötelező.

Változások kezelése

- A model-ben történt adatváltozásokról a JTable-t értesíteni kell az AbstractTableModel megfelelő metódusának hívásával.
  (A mi esetünkben erre most nem lesz szükségünk…)

Metódus	Változás
`fireTableCellUpdated`	Cella frissítése
`fireTableRowsUpdated`	Sorok frissítése
`fireTableDataChanged`	Teljes tábla frissítése
`fireTableRowsInserted`	Új sor került beszúrásra
`fireTableRowsDeleted`	Sor törlés történt
`fireTableStructureChanged`	Teljes tábla érvénytelenítése (adatok és struktúra is)

Renderers

- Az azonos típusú adatok megjelenítéséhez ugyanaz a cell render komponens lesz felhasználva.
- Ha nincs explicit megadott renderer, a táblázat a `getColumnClass` alapján választ egy alapértelmezettet.
  - Boolean → Jelölő négyzet
  - Number → jobbra igazított címke
  - Date → címke
  - ImageIcon, Icon → középre igazított címke
  - Object → a string értéket megjelenítő címke

JTable – Rendezés, szűrés

- A rendezés legegyszerűbb módja, ha a táblázat `autoCreateRowSorter` tulajdonságát igazra állítjuk.
- Készíthetünk azonban saját rendező objektumot is:

```
TableRowSorter<TableModel> sorter = new
TableRowSorter<TableModel>(table.getModel());
table.setRowSorter(sorter);
```

- A `TableRowSorter` egy `Comparator` objektumot használ a sorok rendezéséhez. Az oszlophoz tartozó `Comparator` kiválasztása az az alábbi sorrend szerint történik:
  - `setComparator` metódussal megadott `Comparator`, ha van.
  - `String` oszloptípusnál `String` comparator.
  - `Comparable` osztálynál, a `compareTo` metódus használatos.
  - Megadott `StringConverter` esetén rendezés az objektumok `toString` reprezentációin.
  - Minden más esetben a `toString` eredményeit használó `Comparator`.

JTable – Sorok szűrése

- A sorok rendezése mellett Sorter-el adható meg, mely sorok jelenjenek meg a táblázatban.
- A `TableRowSorter` a szűrést a `javax.swing.RowFilter` segítségével implementálja.

```
RowFilter<MyModel, Object> rf =
                                RowFilter.regexFilter("regexp", 0);
sorter.setRowFilter(rf);
```

- Szűrések és rendezések használatakor az adatok más sorrendben szerepelhetnek a megjelenített táblázatban, mint a modellben.
  - Az indexeket konvertálni kell a megjelenítés és a table model között a `JTable` által biztosított konvertáló metódusokkal:

    `convertRowIndexToModel`,
    `convertColumnIndexToView`...

JTable – Kijelölés (Selection Mode)

- Alapértelmezésként a táblázat minden sora kiválasztható.
- A `JTable.setSelectionMode` metódussal változtatható meg a táblázatban engedélyezett kijelölés módja.
- Ennek értéke a `javax.swing.ListSelectionModel` osztály konstansai lehetnek. (`MULTIPLE_INTERVAL_SELECTION`, `SINGLE_INTERVAL_SELECTION`, és `SINGLE_SELECTION`)

JTable – Kijelölés (Selection Option)

- `rowSelectionAllowed`: ha igaz (és a columnSelectionAllowed hamis) akkor a sorok kijelölhetőek.
- `columnSelectionAllowed`: ha igaz (és a `rowSelectionAllowed` hamis) akkor az oszlopok kijelölhetőek.
- `cellSelectionEnabled`: ha igaz, cellák jelölhetőek ki.
- Kijelölés lekérdezése: `JTable.getSelectedRows` és `JTable.getSelectedColumns`. A kiválasztott indexek tömbjét adják meg.

JTable tartalmának görgetése

```
JScrollPane scrollPane = new JScrollPane(table);
table.setFillsViewportHeight(true);
```

- ScrollPane létrehozása a táblázat konténereként, a táblázat automatikusan hozzáadásra kerül.
- setViewportHeight: felhasználja-e a táblázat a konténer teljes magasságát akkor is, ha a táblának nincs elegendő sora.
- A ScrollPane a táblázat fejlécét automatikusan a viewport tetejére helyezi, az oszlopnevek görgetés közben is láthatóak maradnak.

Oszlopok szélsessége

- Alapértelmezetten minden oszlop egyforma széles, a táblázat teljes szélességét kitöltik.
- Egy oszlop szélességének megváltoztatása:

```
column = table.getColumnModel().getColumn(0);
column.setPreferredWidth(100);
```

JComboBox

- A komponens lehetőséget biztosít arra, hogy kiválasszunk egy elemet több lehetőség közül egy lenyíló lista segítségével.

```
String[] petStrings = {"Bird","Cat","Dog","Rabbit","Pig"};

//Create the combo box, select item at index 4.
//Indices start at 0, so 4 specifies the pig.

JComboBox petList = new JComboBox(petStrings);
petList.setSelectedIndex(4);
petList.addActionListener(this);
```

JSlider

- A JSlider komponens célja numerikus adatok megadása egy minimum és egy maximum érték között.

```
JSlider slider = new JSlider(  JSlider.HORIZONTAL,
                               MIN, MAX, INIT);
slider.addChangeListener(this);
slider.setMajorTickSpacing(10);
slider.setMinorTickSpacing(1);
slider.setPaintTicks(true);
slider.setPaintLabels(true);
```

- A slider mutatójának mozgatása esetén a changeListener StateChanged metódusa hívódik meg.

JSlider cimkék módosítása

```
Hashtable labelTable = new Hashtable();

labelTable.put(    new Integer( 0 ),
                   new JLabel("Stop") );
labelTable.put(    new Integer( FPS_MAX / 10 ),
                   new JLabel("Slow") );
labelTable.put(    new Integer( FPS_MAX ),
                   new JLabel("Fast") );

slider.setLabelTable( labelTable );
slider.setPaintLabels(true);
```

JPasswordField

- A JTextField komponens leszármazottja, jelszavak megadásához szükséges speciális beviteli mező.
- Biztonsági megfontolásokból az értékét karakter tömbben tárolja String helyett.
- A komponens alapértelmezésként egy „pont”-ot ír minden karakter helyére, megváltoztatása: `setEchoChar` metódussal.
- A begépelt jelszó a `getPassword` metódussal érhető el. Ha az érték már nem szükséges, a visszakapott tömböt ki kell törölni.

```
char[] input = asswordField.getPassword();
...
Arrays.fill(input, '0');
```

Feladat

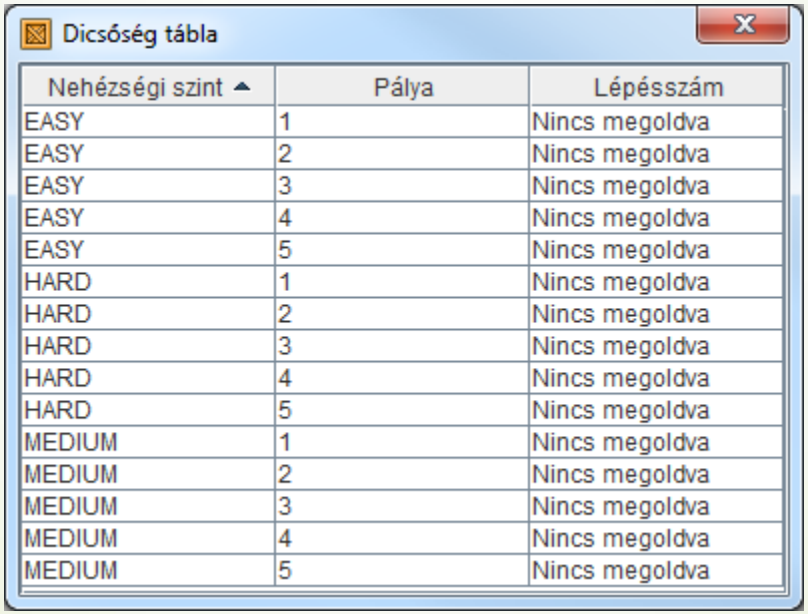
- Egészítsük ki a korábban már implementált Sokoban játékot úgy, hogy egy külön ablakban meg tudjuk nézni melyik pályát hány lépésben sikerült megoldanunk.
- Az eredményeket tároljuk adatbázisban, és ha a játék során jobb eredményt érünk el, akkor ezt frissítsük az adatbázisban.
- A megjelenített táblázatban azt is mutassuk meg, ha egy pályát még nem oldottunk meg.
- Az eredményeket lehessen rendezni lépésszám, nehézségi szint vagy pályaszám szerint.

Feladat – megoldása

- A feladatot két részfeladat megoldásaként értelmezhetjük, melyeket külön rétegekben oldunk meg.
  - Létre kell hoznunk az eredmények megjelenítését (view réteg)
  - Tárolnunk és frissítenünk kell az eredményeket egy adatbázisban (persistence réteg). Ehhez majd készítenünk kell egy adatbázist, amely egyetlen táblából fog állni.

Eredmények megjelenítése

- Az eredményeket egy a játék menüjéből megnyitható dialógusablak JTable komponensében jelenítjük meg



Nehézségi szint ▲	Pálya	Lépésszám
EASY	1	Nincs megoldva
EASY	2	Nincs megoldva
EASY	3	Nincs megoldva
EASY	4	Nincs megoldva
EASY	5	Nincs megoldva
HARD	1	Nincs megoldva
HARD	2	Nincs megoldva
HARD	3	Nincs megoldva
HARD	4	Nincs megoldva
HARD	5	Nincs megoldva
MEDIUM	1	Nincs megoldva
MEDIUM	2	Nincs megoldva
MEDIUM	3	Nincs megoldva
MEDIUM	4	Nincs megoldva
MEDIUM	5	Nincs megoldva

HighScore osztály

```
package persistence;


public class HighScore {
  public final String difficulty;
  public final int    level;
  public final int    steps;
  ...
  // konstruktorok
  public HighScore(GameID gameID, int steps){...}
  public HighScore(String difficulty, int level, int steps){...}
  ...
  // (difficulty, level) páros on generált hashCode és equals metódusok
}
```

HighScoreTableModel osztály

```
package view;
public class HighScoreTableModel extends AbstractTableModel{
  private final ArrayList<HighScore> highScores;
  private final String[] colName = new String[]{"Nehézségi szint","Pálya","Lépésszám"};

  public HighScoreTableModel(ArrayList<HighScore> highScores){
    this.highScores = highScores;
  }
  @Override public int getRowCount() {    return highScores.size(); }
  @Override public int getColumnCount() { return 3; }

  @Override public Object getValueAt(int r, int c) {
      HighScore h = highScores.get(r);
      if      (c == 0) return h.difficulty;
      else if (c == 1) return h.level;
      return (h.steps == 0) ? "Nincs megoldva" : h.steps;
  }
  @Override public String getColumnName(int i) { return colName[i]; }
}
```

Dialógusablak az eredményekhez

- Korábban már láttuk, hogy az adatok görgetéséhez a JScrollPane-t használtuk, tegyük ezt most is.
  - A táblázat konténereként létrehozva, és a táblázatot hozzáadva, utóbbit automatikusan görgethetjük.
  - A táblázat oszlopnevei a viewport tetejére kerülnek, és ott görgetés közben is láthatóak maradnak.
  - `setFillsViewportHeight`: a JTable ezen metódusával mondhatjuk meg, hogy felhasználjuk-e a konténer teljes magasságát akkor is, ha nincs elegendő táblasor.
- Alapértelmezetten minden oszlop egyforma széles, a táblázat teljes szélességét kitöltik. Oszlopszélesség megváltoztatása:

```
column = table.getColumnModel().getColumn(0);
column.setPreferredWidth(100);
```

HighScoreWindow osztály

```
package view;
public class HighScoreWindow extends JDialog{
  private final JTable table;
  public HighScoreWindow(ArrayList<HighScore> highScores,
                         JFrame parent) {
      super(parent, true);
      table = new JTable(new HighScoreTableModel(highScores));
      table.setFillsViewportHeight(true);
      // rendezés megvalósításának helye...
      add(new JScrollPane(table));
      setSize(400,400);
      setTitle("Dicsőség tábla");
      setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
      setLocationRelativeTo(null);
      setVisible(true);
  }
}
```

Eredmények rendezése

- A rendezés sorrendjét és irányát a `setSortKeys` metódussal adhatjuk meg. Egészítsük ki a HighScoreWindow osztályunkat a rendezéssel.

```
public class HighScoreWindow extends JDialog{
public HighScoreWindow(ArrayList<HighScore> highScores,
                       JFrame parent){
  ...
  TableRowSorter<TableModel> sorter =
                    new TableRowSorter<TableModel>(table.getModel());
  List<RowSorter.SortKey> sortKeys = new ArrayList<>();
  sortKeys.add(new RowSorter.SortKey(0, SortOrder.ASCENDING));
  sortKeys.add(new RowSorter.SortKey(1, SortOrder.ASCENDING));
  sorter.setSortKeys(sortKeys);
  table.setRowSorter(sorter);
  ...
  }
}
```

Sokoban adatbázis

- Hozzuk létre a játékunk adatbázisát, mely a HighScore típusú objektumok attribútumait tárolja.
- Ezen belül is a nehézségi fokozat és szint együttesen alkosson egy egyedi kulcsot, ami a pályát egyértelműen képes azonosítani.

```
CREATE DATABASE IF NOT EXISTS sokoban;
USE sokoban;
CREATE TABLE IF NOT EXISTS HighScore (
  Difficulty VARCHAR(50) NOT NULL,
  GameLevel  INT NOT NULL,
  Steps      INT,
  PRIMARY KEY(Difficulty, GameLevel)
);
```

Database osztály

Tárolja a pályákon elért legjobb eredményt (0, ha megoldatlan).

```
public class Database {
  private final String                    tableName = "highscore";
  private final Connection                conn;
  private final HashMap<GameID, Integer> highScores;
  public Database(){
    Connection c = null;
    try { c = ConnectionFactory.getConnection(); }
    catch (Exception e) { System.out.println("No connection"); }
    conn = c;
    highScores = new HashMap<>();
    loadHighScores(); // betölti az adatbázisból az eredményeket
  }
  ...
}
```

Database osztály

Eredmények betöltése az adatbázisból

```
private void loadHighScores(){
  try (Statement stmt = conn.createStatement()) {
    ResultSet rs = stmt.executeQuery("SELECT * FROM "+tableName);
    while (rs.next()){
      String diff = rs.getString("Difficulty");
      int level   = rs.getInt("GameLevel");
      int steps   = rs.getInt("Steps");
      GameID id   = new GameID(diff, level);
      mergeHighScores(id, steps, false);
    }
  } catch (Exception e){
    System.out.println("loadHighScores error"); }
}
```

Database osztály

A betöltött eredményt „összefésüljük” a már ismerttel.

```
private boolean mergeHighScores(GameID id, int score, boolean store){
  boolean doUpdate = true;
  if (highScores.containsKey(id)){
    int oldScore = highScores.get(id);
    doUpdate = ((score < oldScore && score != 0) || oldScore == 0);
  }
  if (doUpdate){
    highScores.remove(id);
    highScores.put(id, score);
    if (store) return storeToDatabase(id, score) > 0;
    return true;
  }
  return false;
}
```

Database osztály

Frissítsük az adatbázist, ha jobb eredményt érünk el, és mondjuk meg hány sort érintett a változás.

```
private int storeToDatabase(GameID id, int score){
  try (Statement stmt = conn.createStatement()){
    String s = "INSERT INTO " + tableName +
               " (Difficulty, GameLevel, Steps) " +
               "VALUES('" + id.difficulty + "'," +
               id.level + "," + score +
               ") ON DUPLICATE KEY UPDATE Steps=" + score;
    return stmt.executeUpdate(s);
  } catch (Exception e){
    System.out.println("storeToDatabase error"); }
  return 0;
}
```

Database osztály

Adjuk meg HighScore listaként a pályákon elért legjobb eredményeinket.

```
public ArrayList<HighScore> getHighScores(){
  ArrayList<HighScore> scores = new ArrayList<>();
  for (GameID id : highScores.keySet()){
    HighScore h = new HighScore(id, highScores.get(id));
    scores.add(h);
  }
  return scores;
}
```

További feladatok

- MainWindow osztály
  - A menüben létre kell hozni egy új menüpontot, a „dicsőség tábla” megtekintéséhez.
- Game osztály
  - Létre kell hozni egy adatbázis példányt a konstruktorból.
  - A pálya megoldásakor meg kell hívni az adatbázis `storeHighScore` metódusát, és el kell tárolni adattagként, hogy ügyesebbek voltunk-e most (`isBetterHighScore`).
  - Készítsünk egy `isBetterHighScore` gettert.
  - Készítsünk egy `isGameEnded` metódust.
  - A `addNewGameLevel` metódus tegye ismertté a pályát a Database osztály számára a `storeHighScore` hívással.