SZAKDOLGOZAT

Jánosi Tímea

Debrecen

2011

Debreceni Egyetem

Informatika Kar

Kezdő japán nyelvi oktatóprogram készítése Java nyelven

Témavezető:

Dr. Csernoch Mária

Egyetemi docens

Készítette:

Jánosi Tímea

Programtervező Informatikus

(Bsc)

Debrecen
2011

Tartalomjegyzék

1. Bevezetés

A ma világában akármerre nézünk, azt látjuk, hogy az anyanyelvünk mellett az idegen nyelvek használata és ismerete egyre fontosabbá válik. Akármelyik országot vagy iskolát nézzük, a munkahelyekről és állásajánlatokról nem is beszélve, mindenhol elvárás ma már, hogy legalább egy, de inkább több idegen nyelven tudjon az ember, lehetőleg folyékonyan.

A fentebb említett okok miatt pedig mindenhol egyre több igény van a nyelvoktatásra. Az iskolák már egyre alacsonyabb osztályoktól indítják az egyes nyelvi képzéseket és egyes óvodák is foglalkoznak a kisgyerekek alapszintű oktatásával. Ha a médiát nézzük, ott is láthatunk nem egy olyan műsort, amely célközönségnek valamilyen nyelv tanulására vágyó csoportot jelöl ki. A hagyományos nyomdai háttér pedig sorozatban ontja magából az ilyen témájú könyveket, hang-és képanyagokat.

Ha belegondolunk, hogy melyek a legnépszerűbb nyelvek, akkor az angol és a német kerül az első helyre. Emellett szerepel még több olyan nyelv a népszerűségi listán, amelyek elsajátítása egyszerűbb, hiszen csak a nyelvtan és szavak elsajátítását igénylik. Azonban vannak olyan nyelvek is, amelyek tanulásához egészen az ábécéhez kell visszamennünk.

Az egyik ilyen nyelv a japán, amely napjainkban szintén kezd népszerűvé válni. Az egyetemen is találkozhatunk vele, hiszen itt is lehetőségünk van a japán nyelv tanulására. Szintén találkozhatunk a kultúra terjesztésére és ismertetésére, valamint barátságok kötésére is lehetőséget adó, évenként megrendezett „Japán napok” név alatt futó rendezvénysorozattal.

Ennek a nyelvnek az elsajátításához azonban szükséges az, hogy rengeteget gyakoroljunk. Ehhez igyekeztünk, csoporttársammal, Ponyi Ildikóval, kitalálni és megvalósítani egy nyelvi oktató programot. A program a japán nyelv nyelvi alapjait mutatja be és lehetőséget biztosít a gyakorlásra. Igyekeztünk egy olyan keretrendszer létrehozni, amely később más nyelvekre is adaptálható, például koreai nyelvre. A program a megvalósítása Java nyelven történt, és a NetBeans IDE segítségével állítottuk össze a felhasználói felületet.

A szakdolgozatom témája tehát egy oktatóprogram készítése, amely egy olyan idegen nyelvet ismertet meg a tanulni vágyóval, amihez szükséges a nyelv alap írásjeleinek elsajátítása is. Ezt a feladatot egy csoporttársammal – Ponyi Ildikóval – igyekeztünk

megvalósítani, a program egyes funkcióinak elkészítését egymás között felosztva. Az általunk készített program hiánypótló, amely elsődleges célja az, hogy a japán nyelv alapjainak elsajátításához adjon segítséget. Az ötlet onnan származik, hogy jómagam is elkezdtem a japán nyelv tanulását, és tudom, hogy rengeteg gyakorlás szükséges hozzá. A tapasztalatok azonban azt mutatják, hogy a kereskedelmi forgalomban elérhető tankönyvek azonban nem nyújtanak elég gyakorlási lehetőséget. Ez igaz a legtöbb japánhoz hasonló nyelvre is, akár kínai, akár koreai nyelvet akarunk tanulni. A nyelvtan tanulása ugyanis lehet könnyű – itt megjegyezném, hogy a japán nyelvtan nem számít olyan bonyolultnak, mint mondjuk a francia vagy akár a német nyelvé – , viszont az előbbi esetek nagyszámú idegen karakter elsajátítását igénylik, ami több ezer karaktert is jelenthet nyelvenként. Ez tulajdonképpen azt jelenti, hogy a nyelvtanuló úgy érezheti magát, mint egy első osztályos, akinek mindent elölről kell kezdeni az ábécé megismerésénél, és majd csak ezt követheti a lexikai és nyelvtani elemek elsajátítása.

A szakdolgozat célja tehát az, hogy a kezdők számára segítséget nyújtson az írásrendszer és a nyelvtan elsajátításához. Olyan funkciókat igyekeztem kitalálni (és szeretnék még a jövőben) a társammal, amelyek mind az új ismeretek átadására, mind pedig a gyakorlásra lehetőséget adnak. A szakdolgozataink ugyan egy programmá állnak össze, de külön-külön is működő egységet alkotnak, így könnyebben megvalósítható volt, hogy amennyiben nem egy időben tudunk dolgozni rajta, illetve nem ugyanolyan határidők szükségesek, mégis tudjunk együtt is és külön is dolgozni rajta.

Az oktatóprogram nem csupán a kezdők számára segítség, hanem az újrakezdőknek is, ugyanis a rengeteg új karaktert és szót rendkívül könnyű folyamatos gyakorlás és használat nélkül elfelejteni.

Az általam készített részekről a 9.2., 9.3., 9.4., 9.5., 9.6. fejezetekben szó esik részletesen. A jelen fejezetben a program második feléről teszek említést, amelyet Ponyi Ildikó valósított meg. A programnak ez a része tartalmazza a nyelvtani leckéket, illetve az ezekhez tartozó gyakorló feladatokat. A második rész nagyobb egységeit a szituációs feladatok és a fordításhoz kapcsolódó tesztek alkotják. A digitális oktatási anyagok előnyeit kihasználva a program lehetőség biztosít a nehézségi szint, valamint a nyelvtani anyag kiválasztására is a felhasználó igényeinek megfelelően.

Bár a nyelvtanok és a hozzá kapcsolódó tesztek még a tervezési fázisban vannak, jó pár ötlet felmerült, hogy mivel lehetne ötletesen és hasznosan segíteni a tanulást.. Mivel például a számozás, vagy akár a külön formájú és nagyságú tárgyak számolása is különböző módon zajlik, így ezekhez tartozó példákat meg lehet jeleníteni szavakkal vagy képekkel és a hozzá kapcsolódó megoldást várni a felhasználótól. Ezt úgy képzelhetjük el, mint mikor az első osztályban, a munkafüzetben egy kiskutya képe szerepelt és nekünk le kellett írni, hogy mi is az. Ez ugyan elég egyszerűnek hangzik, de nagyon sok ember könnyebben megjegyzi az új dolgokat, ha valamilyen vizuális dologhoz tudja őket kapcsolni [21].

A programhoz nem egyszerű szürke hátteret készítettünk, amin fekete színű betűkkel szerepel a tanulnivaló. Próbáltunk olyan felületet készíteni, amivel érdekesebbé tehetjük a tanulást.. Az erről szóló rész a design (10. fejezet) és a felhasználók (11. fejezet) későbbiekben lesznek bővebben kifejtve, illetve a függelékben szerepel majd néhány screenshot a programról működés közben.

Ezek után következzen egy kis ismerkedés a japán nyelvvel, valamint a Java és egyéb felhasznált témakörök rövid bemutatása. Igyekeztem mindenről, ami szóba jött a készítés során szólni valamit. Az egyes fejezetek elején kis ikonokat használtam, amiket próbáltam hasonló stílusban, az adott témakörökre utalva kiválasztani, ezzel is segítve azok elkülönülését.

2. A japán nyelvről

A japán a Kelet egyik legfontosabb nyelve. A világon a hatodik legelterjedtebb nyelv, melyet mintegy 130 millió ember beszél anyanyelveként. Japán rendkívül gazdag kultúrával rendelkezik, melyhez a mai napig ápolt hagyományok társulnak. Emellett korunk világgazdaságának egyik legfontosabb tényezője: a japán gazdaság a világ egyik legnagyobb és legfejlettebb gazdasága. A japán nyelvismeret ezzel nem csupán a sokszínű japán kultúra megismeréséhez és megértéséhez segíthet hozzá, hanem jól hasznosítható a gazdasági életben is [11].

2.1. Dialektusok

A japán nyelvnek létezik egy standard, hivatalos változata, amely a tokiói nyelvjáráson alapszik. Emellett vannak különböző dialektusai, azonban ezek nem térnek el oly mértékben egymástól, hogy az komolyabb értelmezési problémákat okozna [11].

2.2. A japán nyelv genealógiájáról

A japán nyelv eredete máig vitatott. A kutatók egy része az altaji nyelvekkel rokonítja. Laikusok gyakorta feltételezik a kínai nyelvvel fennálló rokonságát, ez a feltételezés azonban minden alapot nélkülöz. Bár a japán nyelv számos írásjegyet és szót vett át a kínai nyelvből, ezektől eltekintve a két nyelv nem sok hasonlóságot mutat [11].

2.3. Nyelvi udvariasság

A japán nyelv jellegzetességei közé tartozik, hogy igen változatos nyelvi eszköztárral rendelkezik a beszélgetőtárs vagy akár a beszédtéma iránt érzett tisztelet, megbecsülés kifejezésére. Ennek megfelelően több fajta beszédmódot is el kell sajátítanunk. Attól függően, hogy hozzánk képest milyen rangú vagy korú emberrel beszélünk, más lesz a megszólítás és a beszéd teljes stílusa is [11].

2.4. Nyelvtan és írás

2.4.1. A japán nyelv tipológiailag

A japán ragozó nyelv, de nyelvtana viszonylag egyszerű, könnyen rendszerezhető, egységes szabályrendszere kevés kivételt ismer. Kötött szórendű – téma-alany-idő-hely-tárgy-állítmány – mondatok jellemzik, számos nyelvtani partikulát (pl.: wa (téma jelölésére), ga (alany jelölésére), o (tárgy jelölője), ni (hely jelölése), no (jelzők)) alkalmaz.

A személyes névmások gyakran kiesnek, ha referenciájuk a kontextusból kiderül. Az első személyben az udvariasság foka szerint többféle alak közül választhatunk, második személyben gyakran ugyancsak udvarias formula helyettesíti a névmást (például megszólított neve, tiszteleti –san/-sama szócskával toldva).

A japánban az igék nem vesznek fel személyragokat – személyragok nincsenek, a személyes névmások vagy a szövegkörnyezet helyettesítik őket –, viszont igeidő és mód tekintetében van igeragozás. Nincs főnévi igenév. Az igék formái a mondatban a különféle udvariassági fokot fejezik ki, pl.: csináld meg; légy szíves csináld meg; tisztelettel kérlek, hogy csináld meg; stb.…)

A magyarral azonos a nevek használata (vezetéknév, keresztnév), a dátum és cím használata, a birtokos eset használata (először jön a birtokos, utána a birtok, például a ház ajtaja). A helymeghatározáshoz szükséges címeket is hasonló módon írják – nagyobb egységtől a kisebb felé haladva –, város, városrész, háztömb stb. [1].

2.4.2. A japán írásról

A japán írásrendszer háromféle írást használ, ezek: a kanji, a hiragana és a katakana. A kanjik a kínaiaktól kisebb-nagyobb módosításokkal átvett írásjegyek. Ezek alapvetően ideografikus jelek, tehát fogalmakat, és nem hangzást jelölnek. A kanjik száma több ezerre tehető, azonban ezt a listát 1981-ben 1945 írásjegyre korlátozták. A hiragana és a katakana írás ezzel szemben egyaránt 48-48 jelből álló, hangjelölő írás (1. ábra). A japán nyelvű szövegekben az igei és névszói töveket rendszerint kanjikkal írják le, a toldalékokat, a partikulákat hiraganával, míg a nem kínai (elsősorban az igen jelentős mennyiségben használt angol) jövevényszavakat katakanával. [1]

2.4.3. Központozási jelek

A régi japán írásrendszerben nem található egyetlen központozási jel sem, az összefüggéseket a szövegkörnyezet és a szövegek elhelyezkedése adja. A szöveg megértéséhez a napjainkban kiegészítő jelek segítenek. Megjegyzendő, hogy a japán írásban nincsenek szóközök, az írásjegyek egymás után következnek, valamint, hogy két fajta írásmódszer van. Az egyik a vízszintesen balról jobbra, a másik a függőlegesen fentről lefelé való írás. Függőleges írásmód esetén az olvasás jobbról balra történik. Az olvasást segítő jelek például:

- 。 kicsi, karika alakú mondatvégi írásjel, használatában megfelel a magyar pontnak.
- 、 vessző, használata a magyaréhoz hasonló, bár szabadabb szintaktikai szabályok alapján.
- ・ a sorok szélességének/magasságának közepén (az írás irányától függően) levő pontot az idegen nyelvű szavak elválasztására használják.

ん	わ	ら	や	ま	は	な	た	さ	か	あ
		り		み	ひ	に	ち	し	き	い
		る	ゆ	む	ふ	ぬ	つ	す	く	う
		れ		め	へ	ね	て	せ	け	え
	を	ろ	よ	も	ほ	の	と	そ	こ	お

ン	ワ	ラ	ヤ	マ	ハ	ナ	タ	サ	カ	ア
		リ		ミ	ヒ	ニ	チ	シ	キ	イ
		ル	ユ	ム	フ	ヌ	ツ	ス	ク	ウ
		レ		メ	ヘ	ネ	テ	セ	ケ	エ
	ヲ	ロ	ヨ	モ	ホ	ノ	ト	ソ	コ	オ

a i u e o

1. ábra
Alap írásjelek táblázatai, a felső a hiragana, az alsó a katakana tábla
A jobb oldali magánhangzók a velük azonos sorban levő kanák magánhangzóit jelölik.

3. Objektum-orientált programozás

Az objektum-orientált programozás (angolul object-oriented programming, röviden OOP) egy programozási módszer.

Ellentétben a korábbi programozási nyelvekkel, nem a műveletek megalkotása áll a középpontban, hanem az egymással kapcsolatban álló programegységek hierarchiájának megtervezése. Az objektum-orientált gondolkodásmód lényegében a valós világ modellezésén alapul – például egy hétköznapi fogalom, a *„kutya”* felfogható egy osztály (a kutyák osztálya) tagjaként, annak egyik objektumaként. Minden kutya objektum rendelkezik a kutyákra jellemző tulajdonságokkal (például szőrszín, méret stb.) és cselekvési képességekkel (például futás, ugatás).

Az objektum-orientált programozásban fontos szerep jut az úgynevezett *öröklődésnek*, ami az osztályok egymásból való származtatását teszi lehetővé: a kutyák osztálya származhat az állatok osztályából, így megörökli az állatok tulajdonságait és képességeit, valamint kibővítheti, vagy felülírhatja azokat a kutyák egyedi tulajdonságaival, képességeivel. [4]

3.1. Az OO-paradigma

Az objektum-orientált paradigma az absztrakt adattípus fogalmára épül. Az objektum-orientált nyelvek legfontosabb alapeszköze az absztrakt adattípust megvalósító osztály. Az osztály maga egy absztrakt nyelvi eszköz, ezen nyelvek implementációi gyakran egy osztályegyüttesként jönnek létre.

Az osztályok rendelkeznek attribútumokkal és metódusokkal. Az attribútumok tetszőleges bonyolultságú adatstruktúrát írhatnak le. A metódusok szolgálnak a viselkedés megadására. Ezek fogalmilag (és általában ténylegesen is) megfelelnek az eljárásorientált nyelvek alprogramjainak.

A másik alapeszköz az objektum, ami szintén egy nyelvi eszköz. Egy objektum mindig egy osztály példányaként jön létre a példányosítás során. Egy adott osztály minden példánya azonos adatstruktúrával és azonos viselkedésmóddal rendelkezik. Minden objektumnak van

címe, amely azt a memóriaterületet jelenti, ahol az adott adatstruktúra elemei elhelyezkednek. Az adott címen elhelyezkedő értékegyüttest az objektum állapotának hívjuk.

A példányosítás folyamán az objektum a kezdőállapotába kerül. Az objektum-orientált szemléletben az objektumok egymással párhuzamosan, egymással kölcsönhatásban léteznek. Az objektumok kommunikációja üzenetküldés formájában valósul meg. Minden objektum példányosító osztálya meghatározza azt az interfészt, amely definiálja, hogy más objektumok számára az ő példányainak mely attribútumai és metódusspecifikációi látszanak. (Ez az úgynevezett láthatósági beállításokkal valósítható meg.) Tehát egy objektum küld egy üzenetet egy másik objektumnak (ez általában egy számára látható metódus meghívásával és az üzenetet fogadó objektum megnevezésével történik), ez pedig, ha tudja, megválaszolja azt (a metódus visszatérési értéke, vagy output paraméter segítségével). Az üzenet hatásának következményeként, az objektum megváltoztathatja az állapotát.

Az objektumnak van öntudata, minden objektum csak önmagával azonos és az összes többi objektumtól különbözik. Minden objektum rendelkezik ehhez, egy egyedi objektumazonosítóval (ez az Object IDentifier, vagy röviden OID), amelyet valamilyen nyelvi mechanizmus valósít meg.

Egy osztály attribútumai és metódusai lehetnek osztály és példány szintűek. A példány szintű attribútumok minden példányosításnál elhelyezésre kerülnek a memóriában, ezek értékei adják meg a példány állapotát. Az osztály szintű attribútumok ezzel szemben az osztályhoz kötődnek, nem „többszöröződnek”. Például ilyen attribútum lehet az osztály kiterjedése, ami azt adja meg, hogy az adott osztálynak az adott pillanatban hány példánya van.

Az objektum-orientált nyelvek az osztályok között egy aszimmetrikus kapcsolatot értelmeznek, melynek a neve öröklődés. Az öröklődés az újrafelhasználhatóság eszköze.

Az öröklődési viszonynál egy már létező osztályhoz kapcsolódóan – melyet szuperosztálynak (szülő osztálynak, alaposztálynak) hívunk – hozunk létre egy új osztályt, melynek elnevezése alosztály (gyermek osztály, származtatott osztály). Az öröklődés lényege, hogy az alosztály átveszi (örökli) szuperosztályának minden (a bezárás által megengedett) attribútumát és metódusát, és ezeket azonnal fel is tudja használni. Ezen túlmenően új

attribútumokat és metódusokat definiálhat, az átvett eszközöket átnevezheti, az átvett neveket újradeklarálhatja, megváltoztathatja a láthatósági viszonyokat, a metódusokat újraimplementálhatja.

Az öröklődés lehet *egyszeres* és többszörös. Egyszeres öröklődés esetén egy osztálynak pontosan egy, többszörös öröklődés esetén egynél több szuperosztálya lehet. Mindkét esetben igaz, hogy egy osztálynak akárhány alosztálya létrehozható.

Öröklődésre vehetjük példának, hogy van egy alakzat nevű szülőosztályunk, ebből érthetően leszármaztatható egy nyílt vagy egy zárt alakzat, mint gyermekosztály. Ezeknek pedig újabb gyermekeik lehetnek. Ha két osztály nincs egymással előd-leszármazott viszonyban, akkor ezeket kliens osztályoknak nevezzük. Azt, hogy mi legyen látható egy osztályból, azt a bezárással tudjuk biztosítani. Pl.: publikus láthatóság esetén az összes kliens számára látható lesz.

Az objektum-orientált nyelvek egy része a dinamikus kötést vallja. Másik részükben mindkettő jelen van, az egyik alapértelmezett, a másikat a programozónak explicit módon kell beállítania. Az objektum-orientált nyelvek általában megengedik a metódusnevek túlterhelését. Ez annyit jelent, hogy egy osztályon belül azonos nevű és természetesen eltérő implementációjú metódusokat tudunk létrehozni. Ekkor természetesen a hivatkozások feloldásához a specifikációknak különbözniük kell a paraméterek számában, sorrendjében vagy azok típusában (ez nem mindig elég).

Az OO nyelvek általában ismerik az absztrakt osztály fogalmát. Az absztrakt osztály egy olyan eszköz, amellyel viselkedésmintákat adhatunk meg, amelyeket aztán valamely leszármazott osztály majd konkretizál.

Egy absztrakt osztályban vannak absztrakt metódusok, ezeknek csak a specifikációja létezik, implementációjuk nem. Egy absztrakt osztályból származtatható absztrakt és konkrét osztály.

A konkrét osztály minden metódusához kötelező az implementáció, egy osztály viszont mindaddig absztrakt marad, amíg legalább egy metódusa absztrakt. Az absztrakt osztályok nem példányosíthatók, csak örököltethetők.

Egyes objektum-orientált nyelvek ismerik a paraméterezett osztály (kollekció) fogalmát. Ezek lényegében az objektum-orientált világ generikusai.

Az objektum-orientált nyelveknek két nagy csoportja van. A tiszta objektum-orientált nyelvek teljes mértékben az objektum-orientált paradigma mentén épülnek fel, ezekben nem lehet más paradigma eszközeinek segítségével programozni. Ezen nyelvekben egyetlen osztályhierarchia létezik. Ez adja a nyelvi rendszert és a fejlesztői környezetet is egyben. Ezen nyelvekben a programozás azt jelenti, hogy definiáljuk a saját osztályainkat, azokat elhelyezzük az osztályhierarchiában, majd példányosítunk. Egyes tiszta objektum-orientált nyelvek az egységesség elvét vallják. Ezen nyelvekben egyetlen programozási eszköz van, az objektum. Tehát ebben az esetben minden objektum, a metódusok, osztályok is. Tiszta objektum-orientált nyelv például a Smalltalk.

A hibrid objektum-orientált nyelvek valamilyen más paradigma (eljárásorientált, funkcionális, logikai stb.) mentén épülnek fel, és az alap eszközrendszerük egészül ki objektum-orientált eszközökkel. Ezen nyelvekben mindkét paradigma mentén lehet programozni. Általában nincs beépített osztályhierarchia (hanem pl. osztálykönyvtárak vannak), és a programozó saját osztályhierarchiákat hozhat létre. Hibrid objektum-orientált nyelvre példa a C++.

Az objektum-orientált paradigma imperatív paradigmaként jött létre. Tehát ezek a nyelvek algoritmikusak, és így eredendően fordítóprogramosak. A paradigma fogalmait a Smalltalk fejlesztői csapata tette teljessé. Ezután kezdtek el folyamatosan kialakulni a hibrid objektum-orientált nyelvek, és megjelent a deklaratív objektum-orientált paradigma is. [13]

4. A Java

A Java egy objektumorientált programozási nyelv, amelyet a Sun Microsystems fejlesztett a ’90-es évek elejétől, egészen 2009-ig, amikor is az Oracle felvásárolta.

A Java alkalmazásokat jellemzően bájtkódra alakítják át, de közvetlenül natív (gépi) kód is készíthető egy Java forráskódból. A bájtkód futtatása egy úgy nevezett Java virtuális géppel történik (angolul JVM – Java Virtual Machine), ami vagy interpretálja a bájtkódot vagy natív gépi kódot készít belőle, és ezt futtatja a használt operációs rendszeren. Létezik közvetlenül Java bájtkódot futtató hardver is, az úgynevezett Java processor.

A programozási nyelv szintaxisát főleg a C és a C++ nyelvekből örökölte, ugyanakkor a Java sokkal egyszerűbb objektummodellel rendelkezik, mint a C++. Itt megemlíthető még a Javascript, melynek a szintaxisa és neve hasonló a Javahoz, de nem egyazon fejlesztői csapat alkotása, csupán idővel kezdték el közösen a két nyelv fejlesztését (mj.: eleinte nem Javascript volt a neve, hanem Mocha, majd LiveScript). [3]

4.1. A Java eredete

A Java nyelv neve először Oak (tölgyfa) lett volna, (James Gosling, a nyelv megalkotója nevezte így az irodája előtt növő tölgyfáról), de később kiderült, hogy ilyen nevű programozási nyelv már létezik, ezért végül a Java névre lett elkeresztelve. A Java szó a Sun Microsystems védjegye, ennél fogva az engedélye nélkül más nem használhatja fel a maga által kifejlesztett termékek megjelölésére, még összetételekben sem: például Java-szerű stb, mert ez a védjegyjogosult jogaiba ütközik. A Java logó eredete szintén egyszerű. Mivel a nyelvet kávézás közben találták ki, így egy kávéscsésze lett a népszerű nyelv ikonja. [3]

4.2. Általánosságban a Javáról

Négy fontos szempontot tartottak szem előtt, amikor a Javát kifejlesztették:

- legyen objektum-orientált
- független legyen az operációs rendszertől, amelyen fut

- olyan kódokat és könyvtárakat is tartalmazzon, amelyek segítik a hálózati programozást
- távoli gépeken is képes legyen a biztonságos futásra

4.3. Objektum-orientáltság

A nyelv első fontos tulajdonsága, az objektum-orientáltság, ami a programozási stílusra és a nyelv struktúrájára utal. Az objektum-orientáltság fontos szempontja, hogy a szoftvert „dolgok” (objektumok) alapján csoportosítja (a tulajdonképpeni valós világ egyedei), nem az elvégzett feladatok lesznek a fő alkotóelemek. Ez azon a tényen alapul, hogy az előbbi sokkal kevesebbet változik, mint az utóbbi, így az objektumok (az adatokat tartalmazó entitások) jobb alapot biztosítanak egy szoftverrendszer megtervezéséhez. A cél az volt, hogy nagy fejlesztési projekteket könnyebben lehessen kezelni, így csökken az elhibázott projektek száma.

4.4. A Java szerepei

A Java szoftver három igen fontos szerepet tölt be, úgy:

1. mint programozási nyelv;
2. mint középszint (köztes réteg - middleware);
3. mint platform.

A Java legfontosabb része a JVM. A JVM mindenütt jelen van (sokféle berendezés, chip és szoftvercsomag tartalmazza), így a nyelv középszintként és platformként egyaránt működik. Ugyanakkor a nyelv „platformfüggetlen” is, mert a Java virtuális gépek interpretálják a szabványos Java bájtkódot. Ez azt jelenti, hogy egy PC-n megírt Java program minimális módosítás után ugyanúgy fog futni egy Javás telefonon is. Innen jön az úgynevezett „írd meg egyszer, futtasd bárhol kifejezés”.

4.5. Platformfüggetlenség (hordozhatóság)

A második tulajdonság, a platformfüggetlenség azt jelenti, hogy a Javában íródott programok ugyanúgy fognak futni különböző hardvereken. Ezt úgy lehet megvalósítani, hogy a Java

fordítóprogram csak egy úgynevezett Java bájtkódra fordítja le a forráskódot, ami aztán futtatva lesz a virtuális gépben, amely lefordítja az illető hardver gépi kódjára. Továbbá, léteznek szabványos könyvtárcsomagok, amelyek egységes módon teszik elérhetővé az illető hardver sajátságosságait (grafika, szálak és hálózat).

Vannak olyan Java fordítóprogramok, amelyek natív gépi kódra fordítják le a forráskódot, ilyen például a GCJ, így valamelyest felgyorsítják a futtatást, de ugyanakkor a lefordított program elveszti a hordozhatóságát.

4.6. Biztonságos távoli futtatás

A Java rendszer volt az első, amely lehetővé tette a távoli gépeken való futtatást. Egy kisalkalmazás futtatható volt a felhasználó gépén (letöltve a Java kódot egy HTTP kiszolgálóról). A kód egy biztonságos környezetben fut, amely nem teszi lehetővé a „rossz szándékú” kód futtatását; a kiadók kérhetnek olyan tanúsítványokat, amelyeket digitálisan aláírnak, tanúsítva, hogy a kisalkalmazás biztonságos, lehetővé téve ennek kilépését a biztonságos környezetből (ugyancsak a felhasználó felügyelete alatt).

4.7. A web és a mobil eszközök

2006-ban megjelent egy változata a Javának JavaFX néven, ami egy szkriptnyelvet takar. Gyorsan és egyszerűen tudunk vele asztali alkalmazásokat készíteni.

A JavaFX drasztikusan lerövidült termelési ciklust kínál a Java-fejlesztők és a webfejlesztők számára egyaránt, valamint megkönnyíti a grafikát, videót, audiót, animációt és gazdag szövegfunkciókat tartalmazó alkalmazások létrehozását. A JavaFX abból a szempontból egyedülálló, hogy egységesített fejlesztési és telepítési modellt kínál gazdag internetes alkalmazások (RIA) asztali számítógépen, böngészőben és mobilon történő kiépítéséhez. Ráadásul az új JavaFX mobil emulátor használata mellett a fejlesztők előzetesen megtekinthetik alkalmazásaikat a nemrégiben bevezetett JavaFX mobilplatformon, amely 2009 tavasza óta már elérhető a Sun mobilpartnerei számára. [3]

5. A Swing

5.1. A Swing elődje

A Swing elődje, az AWT (Abstract Windowing Toolkit). Ez a Java kezdetben megalkotott ablakkezelő rendszere, mely segítségével grafikus felhasználói felületet hozhatunk létre.

Az AWT az operációs rendszer magas szintű, grafikus komponenseket megjelenítő szubrutinjait használja fel a saját komponenseinek megjelenítéséhez, ezért más Java osztályoktól eltérően nem nyújt platformfüggetlen absztrakciós réteget: az AWT-ben írt alkalmazások megjelenése különböző operációs rendszereken nagyon eltérő lehet. Emiatt kezdetben, amikor még nem volt alternatívája, az AWT-t a Java egyik leggyengébb pontjának tartották.

A JDK 2-es verziójának megjelenésével az AWT-t nagyrészt kiszorította a Swing, egy részben az AWT-re épülő, fejlettebb komponensgyűjtemény, ami alacsony szintű szubrutinok segítségével maga rajzolja meg a grafikus komponenseit, így a megjelenése független az operációs rendszertől (emellett több komponenst is tartalmaz, mint az AWT). [16]

5.2. A divatos zene, avagy az eredet

A Java Foundation Classest (JFC-t) tervező csapatot a név kitalálásakor tájékoztatták a zenében jártasak, hogy a Swing az egyik épp divatossá váló zene, így kaptak az ötleten, innen származik tehát a név. A Swinget úgy tervezték, hogy a segítségével olyan Java alkalmazást lehessen írni, amely az általános Windows-os alkalmazások megjelenését tartja szem előtt. Ennek megfelelően a Swing viszonylag nagy és sokat tud. Szerencsére azonban nem feltétlenül szükséges minden kis részletét ismerni ahhoz, hogy egy egyszerűbb alkalmazást el lehessen készíteni a segítségével.

5.3. Miért is szükséges a Swing?

Miért is volt szükség az AWT meghaladására? Az AWT-nek volt egy hibája, amit rendszeresen meg is kaptak azok, akik a Java platformfüggetlenségét hirdették: ez az volt,

hogy az AWT sajnos nem platformfüggetlen, ahogy azt az előbb már említettük (5.1). Az volt oka, hogy a Java ablakelemek egy az egyben leképeződnek a platform saját ablakelemeire. Ha valaki létrehoz egy java.awt.TextArea-t, az szinte azonnal létrehoz egy ennek megfelelő natív ablakelemet, ami persze másként fog kinézni az egyes platformokon, ami csupán egy dolog, de a baj az, hogy másként is viselkedik. Ráadásul szintén problematikus a tény, hogy Windows platformon a TextArea-ba írható leghosszabb szöveg 32 KBájt lehet [6].

A válasz az elégetlenkedőknek a Swing lett. A Swing maga rajzolja az összes ablakelemet, de még a betűket is. És ha már így van, megvalósítottak egy egyedülálló dolgot, amit ők „cserélhető megjelenésnek” hívnak (pluggable look and feel). Ez annyit tesz végül is, hogy a Swing szétválasztja az ablakelemek modelljeit a megjelenésüktől és a konkrét kirajzolással foglalkozó programrészeket akár cserélgetni is lehet. Egy Swing program teljesen új kinézetet kaphat és egy sort sem kell benne megváltoztatni. A Swing képes arra is, hogy kitalálja, milyen platformon fut és a platformnak megfelelő kinézetre kapcsoljon. Ezeknek persze ára van, a Swing kb. 25 Mbájtnyi tiszta Java kód.

A Swinget igen ügyesen tervezték, ismét szép példája annak, milyen hatékony a Sun által űzött kooperatív specifikáció. A Swingnek számos rétege van, mindegyik egyre több funkciót tesz lehetővé. Nemigen lehet elképzelni olyan feladatot grafikus felülettel, amire a Swing nem képes. Ez a komplexitás azonban nem zúdul rögtön a kezdő Swing felhasználó nyakába, a Swing még fontos alapkoncepciókat is eltakar, csakhogy egyszerű programokat gyorsan össze lehessen benne dobni. A Swingben programozó illető folyamatosan mélyülhet el benne és viszonylag kevés ismerettel már egy egyszerűbb saját kinézetet is írhat.

5.4. A Swing működéséről

A Swing használatát elsajátítani könnyű és nehéz is egyben. A NetBeans ugyanis ad egy lehetőséget, hogy csupán vizuálisan összerakva a számunkra tetszetős felületet a kódhoz szinte hozzá se kelljen nyúlni (már ami a grafikus felületet illeti). Ha egy ablakot akar az ember összerakni, csupán egymásra rakja szépen a dolgokat, a framek, panelek, és egyéb elemek sora rendelkezésére áll. És ha a felületünk összeállt, akkor a kód már generálva is van, csupán a működést kell leírnunk.

Ez ugyan valóban egyszerűbbé teszi a dolgokat, azonban vannak funkciók, amelyeket így nem tudunk olyan egyszerűen megvalósítani. Amennyiben például levesszük az ablakról a keretet, akkor arra lesz szükség, hogy az ablak felületére kattintva legyen mozgatható. Ezt azonban saját kóddal szükséges megoldani, amit a designer által generált kódba nem a legegyszerűbb beleépíteni. A designer fél megoldást nyújt ezzel, és ezt is tükrözi azoknak a felhasználóknak a véleménye, akik azt írják és mondják, hogy aki komolyabban akar ilyennel foglalkozni, az jobban jár, ha az elejétől kezdve a saját kódjára támaszkodik, mindenféle vizuális segédeszköz nélkül.

6. Az XML és a Java

6.1. Az XML

Az XML (Extensible Markup Language, Kiterjeszthető Leíró Nyelv) egy általános célú leíró nyelv speciális célú leíró nyelvek létrehozására. Az elsődleges célja strukturált szöveg és információ tárolása és megosztása az interneten keresztül.

Az XML-t gyakran használják dokumentumtárolási és feldolgozási formátumként, mind on-line mind off-line módon, és több előnnyel is jár:

- internetes szabványokon alapuló, erőteljes, logikailag ellenőrizhető formátum
- a hierarchikus struktúrája megfelel a legtöbb (de nem mindegyik) dokumentum típusnak
- egyszerű szöveg formátumban valósul meg, licencektől és korlátozásoktól mentesen
- platformfüggetlen, így viszonylag immúnis a technológiai változásokkal szemben
- az XML-t és elődjét, az SGML-t már több mint tíz éve használják, így széles tapasztalat és eszközkészlet áll rendelkezésre

6.2. A DOM

DOM (Document Object Model) egy platform- és nyelvfüggetlen megoldás, melyet a Javában használhatunk standard XML dokumentumok feldolgozására. A dokumentumot egy faszerkezet formájában ábrázolja, melynek csomópontjai az elemek, attribútumok, illetve szövegrészek. A DOM segítségével adatokat nyerhetünk XML-ből és ugyanilyen formában menthetjük azokat. Ehhez a DOM eszközt nyújt arra, hogy a feldolgozni kívánt XML-t betöltsük a memóriába és ezek után dolgozhassunk vele. A DOM akkor ajánlott, amikor nem szekvenciális, hanem ismételt hozzáférés történik egy dokumentumhoz. A SAX (Simple API for XML) és főleg a STAX (Streaming API for XML) ezzel ellentétben szigorúan szekvenciális feldolgozás esetén ajánlott használni, ami nagyobb fájlok esetén jobb megoldás lehet.

7. A futtatható jar

7.1. A JAR bemutatása

A JAR fájl nem más, mint egy olyan ZIP fájl, amely Java osztályokat és hozzájuk tartozó metaadatokat tartalmaz.

- A WAR (Web Application aRchive) fájlok olyan ZIP fájlok, amelyek XML fájlokat, lefordított Java osztályokat, JSP fájlokat és egyéb webalkalmazás-objektumokat tartalmaznak.
- RAR (Resource Adapter aRchive) fájlok (nem összekeverendő a RAR fájlformátummal), szintén Java archive állományok, tartalmazhatnak XML fájlokat, Java class-okat és más objektumokat a J2EE Connector Architecture (JCA) alkalmazások számára.
- Az EAR (Enterprise ARchive) fájlok XML fájlokat, lefordított Java osztályokat és egyéb üzleti alkalmazás-objektumokat tartalmaznak.

JAR fájlokat készíteni egy, a JDK részét képező eszközzel lehet. Ez NetBeans esetén azt jelenti, hogy buildeléskor generálunk JAR formátumú futtatható fájlt a kódunkból. Mivel tömörített fájlokról van szó, más tömörítő eszköz is használható, de például a WinZip kerülendő, mert a nagybetűs könyvtár- és fájlneveket átnevezi kisbetűsre.

A JAR fájl tartalmazhat egy úgynevezett manifest fájlt a META-INF/ könyvtárban MANIFEST.MF néven. Ez a fájl írja le a futtató környezet számára, hogy hogyan kell a JAR fájlt használni. Szerepelhet benne például egy bejegyzés arról, hogy a JAR fájl futtatásakor melyik osztály melyik statikus metódusát kell elindítani (például: *Main-Class: myPrograms.MyClass*). A JAR fájlokat jellemzően így indítják: *java -jar valami.ja.*

A manifest fájl tartalmazhat classpath bejegyzést is, ami azt mondja meg, hogy a virtuális gép mely jarokat töltse be (pl.: *Class-Path: /lib/jxta.jar /lib/bcprov-jdk14.jar*). A jar fájl tartalmazhat digitális aláírást is, ami szintén a manifest fájlban tárolódik. [2]

7.2. A jar használatának előnye és hátránya

Az oktató program írása során az egyik elsődleges kérdés volt, hogy hogyan is lesz a kódból futtatható fájl, hiszen egy átlagos felhasználó csak a megszokott duplakattintással fogja tudni futtatni a programot.

A JAR fájl szinte adja magát a Javanak köszönhetően. Ez egyszerű megoldás, de ugyanakkor magában rejti a hibákat is. Egyrészt itt is szükség van Javás letöltésre a felhasználónak, ha a programot futtatni szeretné, ami még nem is akkora gond, hiszen a programok nagy része igényel valamiféle kiegészítőt, akár DirectX akár a .NET jön szóba. Így ez vehető relatíve kis árnak. A másik probléma, hogy a Windowson a futtatás egyszerű, de a Linux esetén nem biztos, hogy olyan egyszerűen futni fog a program. Ugyanakkor azt is megemlíthetjük, hogy aki nem a nagy többség Windows felhasználói körébe tartozik, annak nem okoz gondot egy parancssori indítás más operációs rendszer alatt.

Egy másik felmerült kérdés volt még, az, hogy az eredetileg elkészült kódot, át kell írni, ahhoz, hogy JAR formátumban is ugyanúgy működjön, mint ha NetBeans segítségével futtatnánk. Ez a következő okok miatt történik: ahogy a fentebb írt rész taglalja (7.1 fejezet), a JAR fájl egy tömörített formátum, aminek köszönhetően a felhasznált és tárolt fájlok nem érhetőek el ugyanúgy, mint egy NetBeansből történő futás során, közvetlenül (például egy könyvtár kilistázása). Ennek egyszerű oka van, a tömörítés után, a JAR fájlba bekerülve már streamként lesznek csak olvashatóak az egyes fájlok, így a mappák elérését/listázását újra kell írni. A függelékbe került rész ennek a megvalósításáról (3. kódrészlet, 4. kódrészlet).

8. Az egyedi betűtípus problémái

Az oktatóprogram írása során felmerült következő probléma, ami az egyedi betűtípus volt, ugyanis a szótárnál van olyan rész, ahol saját betűtípust használ a program. Ez több kérdést is felvetett esetünkben:

Az egyik megoldásra váró gond, hogy olyan betűtípust szeretnénk, amely támogatja a magyar ékezetes karaktereket. Ilyen betűtípus találására van pár lehetőség:

- Keresünk valamilyen stílusában megfelelő betűtípust, majd ezt egy betűtípus szerkesztő programmal kiegészítjük a megfelelő karaktereinkkel, ez viszont nem feltétlenül úgy működik, ahogy kellene. Ezt próbálva azt tapasztaltam, hogy a MS Word és egyéb szövegszerkesztők tökéletesen látták és használták is az általam beleépített ékezetes betűket. A program ezzel ellentétben nem jelenítette meg azon karaktereket, amelyeket utólag a betűtípus szerkesztőjével tettem bele.
- A másik lehetőség: használjuk azt, ami van, hiszen a japán és sok magyar szó esetén sincs szükség ö-ő betűkre. Így a japán olvasatnál megfelelő, a magyarnál pedig lehet mást használni, ha szükséges (erről később bővebben a szótárnál (9.5 fejezet). Ez a módszer bevált, így a későbbiekben ezt használtam.

A következő probléma: a fejlesztéshez használt rendszeren szerepel a betűtípus, de ha a felhasználó gépén futtatnánk a programot, akkor le kellene töltenie/telepítenie a betűtípust? Ez nem lenne túl elegáns megoldás, így erre egy külön osztály készült, amelynek egy metódusa még a program grafikus felületének megjelenítése előtt lefut, és a felhasználó betűtípusai közé másolja az általam kijelölt mappa tartalmát (azaz a használt egyedi betűtípus(okat). Ez a függelékben megtalálható (5. kódrészlet). Erre nem kér engedélyt, ez azonban minimális változtatással (új panellel és egy if elágazással) megoldható. Ez jelenleg csak Windowsos felhasználókat érint, mivel a célmappa elérése más itt más, mint Linux esetében. Ez így egy a jövőben fejleszthető dolgok közül. Megjegyzésnek hozzátenném, hogy belekerült egy ellenőrzés, amivel csak akkor másolja a program a betűtípust, ha az még nem szerepel a telepítettek között. Ennek az volt az oka, hogy a már egyszer vagy többször felülírt betűtípust nem tudta használni semmilyen kép és szövegszerkesztő.

9. A program funkciói

9.1. A funkciók felosztása

A program kitalálása során néhány olyan alapfunkciót terveztünk, amely a japán nyelv nyelvtanához és ábécéjéhez szükséges alapok elsajátításában fog segítséget nyújtani. Ebből kifolyólag a kanjik a programban nem szerepelnek, de később tetszőlegesen beépíthetőek, így továbbfejlesztve a programot magasabb fokú tudás reprezentálásához.

Ahogy már a dolgozatom elején is említettem (1 fejezet), a nyelvtani részeket és az azokhoz kapcsolódó teszteket Ponyi Ildikó készíti. A projekt tervezésénél az a döntés született, hogy több forrást áttekintve kiválasztunk körülbelül tíz leckét, amely az alapvető nyelvtani áttekintést fogja adni. Ezekhez kerül majd egy leírás, valamint teszt, könnyebb vagy nehezebb, választhatóan. Annak érdekében, hogy az egyes részek könnyebben elsajátíthatóak és kevésbé unalmasak legyenek, megpróbáltuk az egyes részeket színesebbé tenni, ábrák, háttér stb. segítségével.

Az általam készített részek főleg az írás és a szavak elsajátítására szolgálnak. Mivel a japán nyelv tanulása során egy teljesen új ábécét is el kell sajátítani, rendkívül fontos hogy ezt gyakorolja a felhasználó. A most következőkben az általam elkészült funkciókról lesz szó. Bár mondhatnánk, hogy kanjik nélkül mit sem ér a japán tudásunk, nem szabad megfeledkeznünk arról, hogy a hiragana és katakana táblázat elsajátítása is rendkívül időigényes. Ugyanis mindkét táblázat 48-48 írásjelet tartalmaz, amelyhez társulnak speciális módosítók is. Így összességében több mint száz jel elsajátítására van szükség, mielőtt még a kanjik felé fordulnánk.. A program tehát a hiragana és a katakana írásjelek elsajátításához nyújt segítséget tesztek, kisegítő táblázatok, valamint egy szótár felhasználásával.

Mivel ez a fejezet a fejlesztett program bevezetése, megemlíteném még, hogy a programunk a Yamo nevet kapta, a japán-magyar szavak alapján.

9.2. A főmenü

A program indításakor a főmenü jelenik meg (2. ábra). Az ablak felső részén a program neve – Yamo – látható valamint a név alatt a menüpontok vannak felsorolva – *SZÓTÁR*, *KANA TESZT*, *KATAKANA TÁBLÁZAT*, *HIRAGANA TÁBLÁZAT* és *KANA DEMO*. Az egyes menüpontokra kattintva választhatunk a program funkciói közül. A kattintás hatására egy új ablakban megjelenik a kiválasztott funkció, majd ezen az ablakon dolgozhatunk.

Az első terv szerint a menüpontra kattintás után ugyanazon ablakban jelent volna meg a megnyitott menürész, azonban ha belegondolunk, valakinek szüksége lehet arra, hogy egy időben több funkciót használjon. Ilyen eset lehet például az, ha az egyik lecke vagy teszt során szótárt akar használni, akkor erre csak abban az esetben lesz lehetősége, ha külön ablakban nyílnak meg az egyes részek. A menü jelenlegi formájában természetesen nem teljes, hiszen a program csak a Ponyi Ildikó által elkészített részekkel egész, azaz a menüpontok listája még növekedni fog, ha a két programrészt összerakjuk.

A menüpontok megjelenítése a következő kódon alapul:

```
   private void hiraChartMouseClicked(java.awt.event.MouseEvent evt) {

       HiraChart hiraganaChart = new HiraChart();
       hiraganaChart.setIconImage(Toolkit.getDefaultToolkit().
       getImage(outOfJar + "favicon.png"));
       hiraganaChart.setVisible(true);
}
```

1. kódrészlet
A menüpontok megjelenítése

A menü alapja a *mainFrame*, amely tartalmazza az egyes menüelemeket. A háttér, a kis favicon és azon elemek, melyekkel valamilyen eseményt hívhatunk meg, mind ehhez a framehez vannak hozzárendelve. Amennyiben az új ablak megjelenítését tekintjük, ez annyit jelent kódban, hogy az az osztály, amely az adott menüt írja le, példányosítva lesz a megfelelő helyen. Ezeken a részeken rendeljük hozzá a kis logót is, amelyet úgy tölt be a program, hogy megkeresi a JAR fájl helyét, onnan visszalép egyet, és ott található meg a favicon képe.

2. ábra
A főmenü

Még az egyes menüből elérhető funkciók leírása előtt megjegyezném, hogy az általam elkészített menüpontok betűtípusa nem tartalmaz ö-ő illetve ü-ű karaktereket, ugyanakkor ezeket képszerkesztővel előre elkészített képként tettem be, hogy abban az esetben, ha szükség lenne a hiányzó ékezetes karakterekre, könnyen megoldható legyen a pótlásuk. Ez a fent (8. fejezet) említett ok miatt – a szöveg-/képszerkesztő tudja használni a szerkesztő által készített ékezetes karaktereket – lenne így megoldható.

9.3. Kana demo

A japán nyelv elsajátítása során első lépésként fontos, hogy megtanuljuk az egyes írásjeleket, illetve hasznos, ha tudjuk, hogy néznek ki nyomtatásban. A Kana demo programrészlet ennek a segítésére készült. Ez a funkció szükséges lenne, hogy a későbbiekben bővüljön, mivel a kézírásban a jelek valamivel másabbak, ráadásul az egyes hiraganák és katakanák írása során számít az is, hogy az egyes vonásokat milyen sorrendben húzzuk meg.

3. ábra
A Kana demo kezdőképernyője

A Kana demo (3. ábra) használatakor a felhasználónak lehetősége van arra, hogy válasszon, mely típusú kanákkal akar foglalkozni, katakana vagy hiragana, illetve lehetőség van a kettő egyidejű tanulására is. Két kis jelölőnégyzet segítségével ki lehet választani a kívánt kanatípust. Ezek után a *KATTINTS RÁM AZ ÚJ KANÁÉRT!* gombra kattintva az ablakban megjelenik egy új japán írásjel, amely alatt egy kis szöveges mező mutatja felette megjelent kép olvasatát (4. ábra). Ami esetleg probléma lehet, az az, hogy az olvasata mi lesz az egyes jeleknek. Ez azt jelenti, hogy pl. ha a „kanjit” tekintjük, akkor ezt magyar olvasatban „kandzsi”-nak, angolban „kanjinak”írjuk az ejtés pedig természetesen azonosan „dzs”. A

programban igyekeztem a leggyakoribb és legtöbb helyen fellelhető olvasatot – az angol karakteres olvasatot – használni.

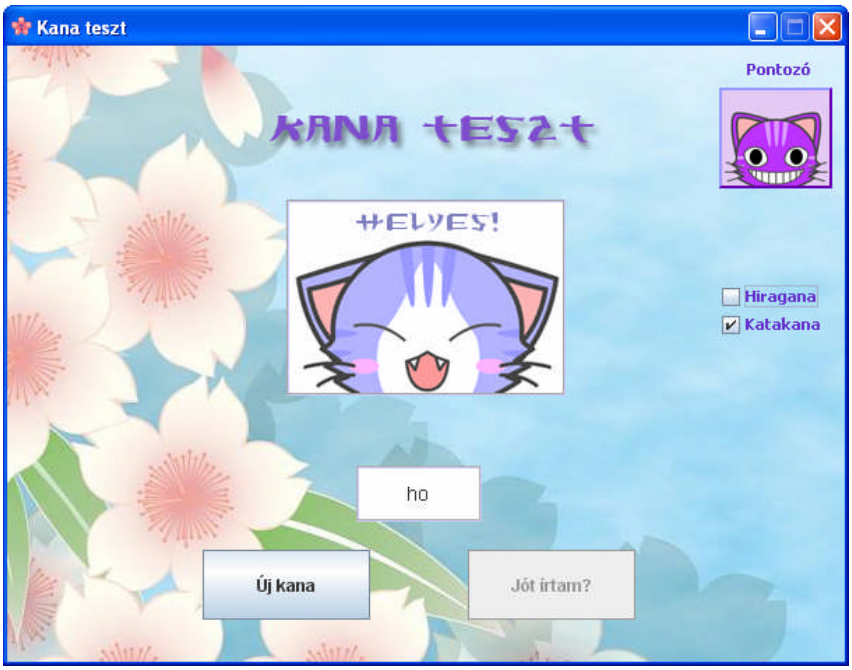
A program ezen része nem csak a tanulás, de a gyakorlás szempontjából is hasznos lehet. Két bővítési lehetőség is ajánlkozik. Egyrészt a fentebb említett kanjik bevitele, illetve a tanulás módosítása. Ezen belül gondolhatunk olyanra például, hogy nem gombnyomásra, hanem valamennyi idő elteltével vált a kép/szöveg. Új funkcióként átalakítható tesztté, amely egy ideig vár a válaszra, majd ha nem kap, nulla pontnak veszi és továbblép.

A működés hátterében a *Random();* függvény áll és válogat a tárolt képek közül, és mivel JAR formában vagyunk, ezért az egyszerű kilistáztatás nem működik. Ha a képet megszereztük a kívánt helyről (*kana.setIcon(…);*), akkor ezt dinamikusan cseréljük, ugyanis futás közbeni eventről van szó. A kép maga egy JLabel (*kana*), amelyhez hozzárendelődik kattintásra az új imageIcon. A tárolt képek nevéből pedig egyszerűen kinyerhető az olvasat, amely pedig a szöveges mező – ami egy JTextField – (*kanaGuess*) új értéke lesz. A működés részletesebben a következő fejezetben lesz leírva.

4. ábra
A Kana demo működése

9.4. Kana teszt

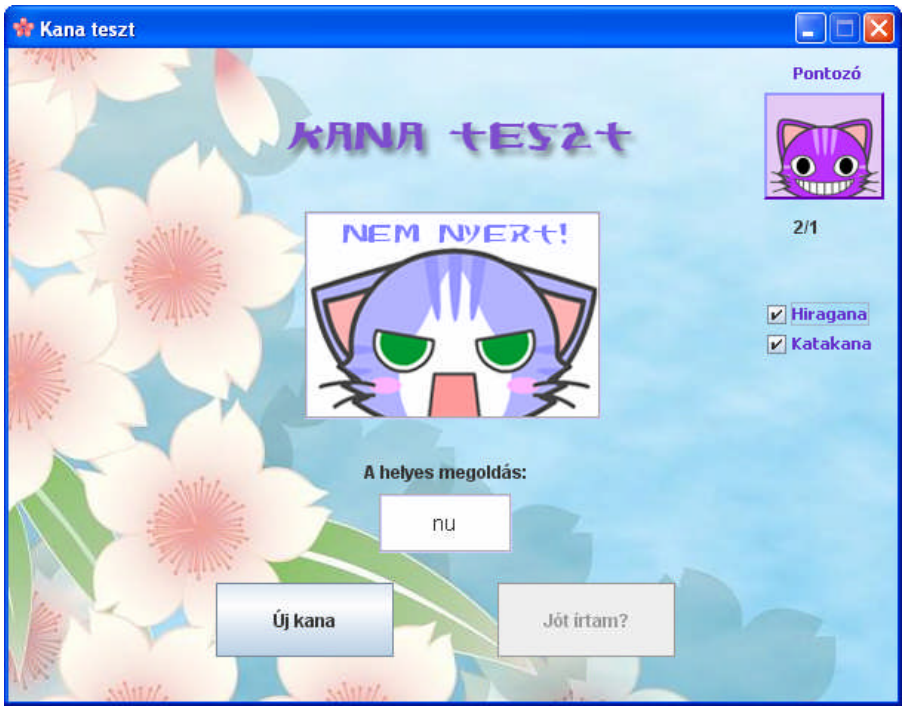
A hiragana és katakana gyakorláshoz készült ez a Kana demohoz hasonló programrészlet, amely arra nyújt lehetőséget, hogy kipróbáljuk az eddig elsajátítottakat. Az 5. ábra és 5. ábra mintái mutatják a Kana teszt programrészletet futás közben A teszt megkezdése előtt „A kezdéshez válassz a jobb oldali kanatípusok közül” szöveges üzenet tájékoztat minket, hogy válasszunk az írásjelek közül, melyekkel akarjuk tesztelni magunkat. Itt is lehetőség van arra, hogy külön-külön (hiragana vagy katakana), vagy pedig egyszerre mindkét kanatípusból kapjunk egy írásjelet.



5. ábra
A helyes megoldás a Kana tesztben

Miután kiválasztottuk a kanatípust, az *ÚJ KANA* gombra kattintva kapunk egy véletlenszerűen kiválasztott írásjelet ami a *kana* nevű JLabelben jelenik meg. A felhasználó annyit érzékel a változásokból, hogy a kanatípus választására felszólító üzenet helyén megjelenik egy kana. A kana képe alatt pedig szerkeszthetővé válik a *kanaGuess* szövegdoboz, ahova a felhasználó begépelheti a megoldást. Ebbe a szövegdobozba beleírjuk a megfelelő olvasatot, és a *JÓL ÍRTAM?* ellenőrző gombra kattintunk a jó – vagy rossz – megoldás megtekintéséhez. Ezek mellett van egy *PONTOZÓ* is a jobb felső sarokban, a

pointCounter, amely segítségével egy kattintással megtudhatjuk, hogy éppen hány találatunk van az eddigi kérdések számához képest. Ezt az eredményt a *points* nevű JLabel írja ki a kattintás hatására.



6. ábra
Rossz megoldás a Kana tesztben

Nézzük egy kicsit részletesebben az ellenőrzést! A *JÓL ÍRTAM?* JButtonra kattintva több dolog is történik. Egyrészt, ha jó a megoldás, egy kis képecske tájékoztat a „Helyes” szöveggel együtt arról, hogy eltaláltuk a kanát (5. ábra), míg rossz tippelés esetén egy másik kép a „Nem nyert” szöveggel az ellenkezőjéről fog minket értesíteni (6. ábra), és a szövegdobozban megjelenik a helyes megoldás. A helyes és helytelen megoldásról értesítő. képek a kanák helyén a *kana* JLabelben jelennek meg Emellett a szöveges mező, amelyet eddig használhattunk, ideiglenesen zárolódik, azaz nem lehet új tippet írni, amíg új képet nem kérünk egy hiraganáról vagy katakanáról. Ez azért volt szükséges, mert ebben a részben van lekódolva az a kiegészítés, ami számolja, hogy hány kérdésre hány jó feleletet adtunk. Ez a pontozórendszer pedig átverhető abban az esetben, ha a mezőnk szerkeszthető marad, ugyanis az *JÓL ÍRTAM?* gomb nyomására következik be az az esemény, melynek hatására a pontok számolódnak. Így ha jó megoldást írunk be, akkor nő a helyes találatok száma, ellenkező esetben csak a kérdések számát növeli. A fentiekből kifolyólag tehát egyrészt az ellenőrzés

csak egyszer végezhető el, utána a gomb mindaddig nem használható, amíg új kanát nem kérünk. Másrészt, ha már egyszer beírtunk egy megoldást, akkor nem írható felül. Persze amennyiben rossz választ adtunk, akkor a helyes megoldásról is kapunk egy kiírást (*kana* alatti *rightGuess* JLabelben), hogy helyesen tanuljuk meg az adott kanát.

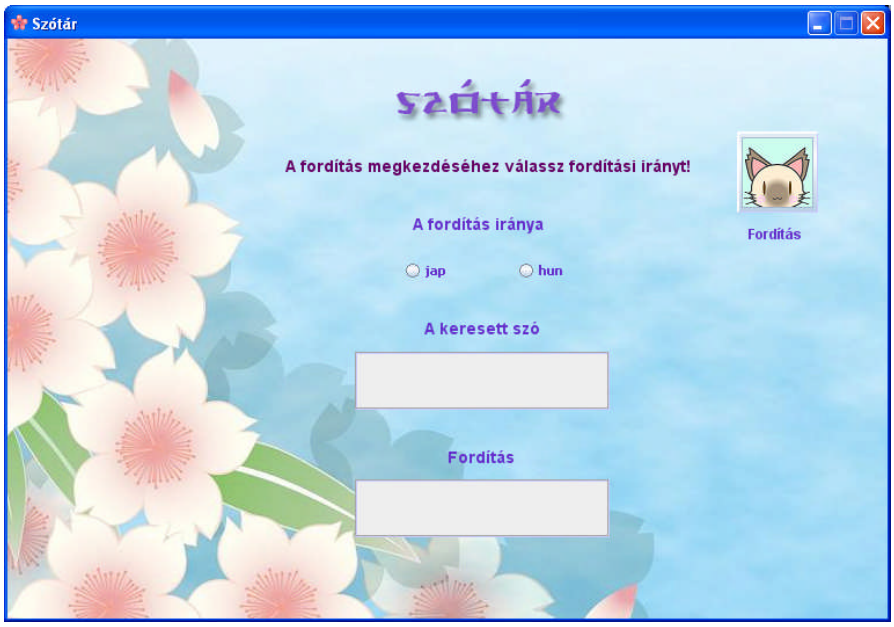
A Kana teszt részben is az előzőhöz (9.3 fejezet) hasonló elven történik a működés futás közben. Alapvetően kiindulunk a tájékoztató képből, mely szintén a *kana* JLabelhöz rendelt imageIcon. Ezek mellett a kis jelölőnégyzetek üresen indulnak, és a pontozó is alaphelyzetbe áll 0/0-ra. Ha kiválasztottuk a számunkra megfelelő kanatípust/típusokat, akkor a *newKana* JButtonra kattintva kérhetünk egy új képet. Ekkor a *kana* JLabelhöz rendelődik egy random kép attól függően választva valamely entryből, hogy mely kanákat jelöltük ki. A változás után újrarajzolja a program a kanát, valamint üresre állítja a *kanaGuess* JTextField szöveges mezőt. Amennyiben pedig előzetesen kértünk ellenőrzést, akkor a *check* gombot ismételten aktívvá teszi. Ha előzőleg rossz megoldásunk volt, akkor kaptunk egy kis szöveges üzenetet a helyes olvasatról. Ez szintén eltűnik az új kana kérésével.

Miután a kép váltott és az egyéb szerkeszthetőségi tényezők is végrehajtódtak az event hatására, akkor jöhet tippelés. Ennek során a *kanaGuess* szöveges mezőbe érkezik a felhasználótól egy karaktersorozat, amit ellenőrzéskor a *kanaGuessből* kiolvastatunk, majd a képünk nevéből kinyert megoldással összehasonlítjuk. Ha helyes, megjelenítjük a helyes megoldáshoz tartozó képet, növeljük a kérdések és a helyes megoldások számát is. Ha a megoldás helytelen, akkor csak a kérdések száma növekedett, a rossz megoldáshoz tartozó képecske jelenik meg, valamint a helyes olvasat is kiírásra kerül. Természetesen az előbb említett aktív szerkeszthető elemek beállítása megtörténik jó vagy rossz megoldás esetén is.

A funkció alapötletét egyébként számos internetes megoldásban láthatjuk ([16][18][19]). További fejlesztés az lehetne, ha ezt kanjikkal kibővítenénk, ami két újabb dolgot igényelne. A kanjik nem csak egy, hanem két fajta olvasattal rendelkeznek. Így ennél két mező lenne szükséges, amely plusz mező nyilván attól függően jelenne meg, hogy a kanjikat is akarjuk-e gyakorolni. Az egyik JTextField a japán, a másik pedig a kínai olvasat tippeléséhez és elsajátításához nyújtana segítséget. Ekkor, ha csak egy hiragana vagy katakana tippelése lenne a cél, akkor a japán olvasatba lehetne írni, vagy pedig a kanji teszt külön funkcióként üzemelhetne.

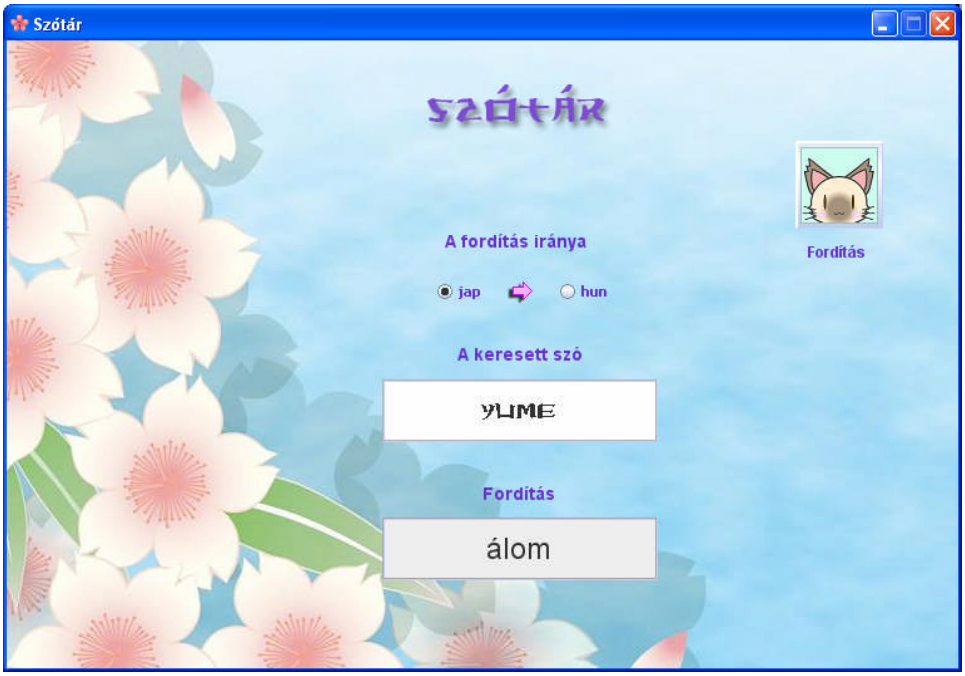
9.5. Szótár

Egy másik alapvető kellék, ha nyelvet akarunk tanulni, az, hogy legyen a közelünkben egy szótár. Ez fontos, hiszen egy idegen nyelv elsajátítása közben bármikor szükségünk lehet arra, hogy új szóval bővítsük a tudásunkat. A program egy pár alap szó segítségével igyekezett ezt elősegíteni. A fejlesztés során nem a szótár nagyságára helyeztem a hangsúlyt, hanem a megvalósításra. Ennek eredményeképpen a szótár csak bétaverzió, jelenleg csupán pár szót tartalmaz, ami azért kellett, hogy a működését tesztelni tudjam. A szótár természetesen magyar és a japán nyelvek között fordít. Lássuk először, hogy felhasználóként mit és hogyan kezelhetünk benne, illetve mi hogyan jelenik meg.



7. ábra
A szótár kezdőképernyője

Miután megnyitottuk a Szótár ablakot (7. ábra), az elsőként aktív és választható lehetőség, az a fordítás iránya, amelyről „A fordítás megkezdéséhez válassz fordítási irányt!” szöveg is tájékoztatja a felhasználót. Amíg ezt nem tesszük meg, addig nem lesz lehetőség a szó keresésére. A fordítás irányának kiválasztására két rádiógomb ad lehetőséget. Ha valamelyiket kiválasztjuk, akkor a megfelelő fordítási irányt jelző kis nyilacska fog megjelenni, egyértelműsítve a döntést. Ezek után aktívvá válik A KERESETT SZÓ szöveges mező, és az előző választásunktól függően a keresőmezőbe gépelhetjük a kívánt szót.



8. ábra
A szótár működése

A szó beírása után a *FORDÍTÁS* gombra kattintva a program egy XML-ben tárolt adatbázis alapján kikeresi a programunk a szóhoz tartozó fordítást, és a *FORDÍTÁS* (nem szerkeszthető) mezőben megjelenítve olvashatjuk a megoldást (8. ábra). A fordítás megkezdéséhez egy gomb áll a rendelkezésünkre, ennek hatására hajtódik végre a művelet. Amennyiben a szó nem áll a rendelkezésünkre, pl.: rosszul gépeltünk be valamit, vagy nincs benne az adatbázisban, akkor értesítést kapunk róla, hogy ilyen szóval nem tudunk szolgálni a felhasználó számára – *NINCS ILYEN SZAVAM.*

Az XML mellé megemlítenék még egy érdekességet. Ahogy fentebb említettem (7.2 fejezet), a program JAR formában van, és ez bizonyos átalakításokat igényelt. Többek között az elérés is változik, ami nem lehet gond alapvetően. Az XML elérést és azzal való munkát a JAR fájlunkon belül több helyen is megnéztem, és több módon is próbáltam. Végül elég egyszerű lett a megoldás: egy URL-ként megkeresi a program az XML fájlt, majd azt átalakítja karaktersorozattá, és ez a szöveges megfelelő lesz átadva a parsernek.

A szótár működése során az egyes kattintások különböző hatásokat váltanak ki. Kezdjük az elejétől, a fordítási irány megadásától.

Mint írtam a fejezet elején, a *toTranslate* keresőmező nem szerkeszthető az elején, ami a következő miatt fontos: a japán szavak esetében más betűtípust használok, mint a magyar nyelvűeknél. Ez lehetséges, hiszen a japán olvasat angol karaktereket használ, míg a magyar nyelv egy egyszerű Arial betűtípussal írja ki a szavakat, ami az összes magyar karaktert támogatja. Emiatt van még az is, hogy ha új fordítási irányt adunk meg, akkor az eddig beírt szavak – mind a keresőmező mind a fordítási mező – törlődnek. A fordítás irányával ugyanis változni fog a két mező betűtípusa is. Ez a vegyes betűtípusos megoldás azért is hasznos lehet, mert a program bár jelenleg nem ír és keres japán karakteres szavakat, a későbbiekben tervezem ezen rész felderítését, hiszen ez elengedhetetlen a latin betűs olvasat kiírása mellett.

Ennél a résznél megemlíthetjük, hogy csupán egy japán karaktereket támogató font beépítése nem elég, hanem nagy valószínűséggel szükséges egy properties fájl is, azonban ez önmagában nem ilyen egyszerű. A properties fájlban ugyanis nem a japán karakteres megfelelőt kell megadni, hanem azoknak az egyes karaktereihez tartozó ASCII kódját. Ráadásul felmerül a kérdés, hogy ha valakinek nincs egy Windowsos környezetben a kelet-ázsiai nyelvekre vonatkozó kiegészítés telepítve, az valóban tudja-e majd használni. Szintén jó kérdés az adatbázis tárolása és megoldása ebben az esetben. Ez a pár dolog tehát későbbiekben tesztelésre szorul. Felmerült, hogy az UNICODE támogatja a japán karaktereket is, ez azonban nem jelenti azt, hogy a program meg is tudná jeleníteni sajnos.

Egy másik esemény ami fontos még az, hogy abban az esetben, ha a keresőmezőbe kattintunk, akkor törlődik az előzőleg beírt szó és a hozzá tartozó fordítása is. Ez azért is fontos, hogy ne legyen ott a fordítása másnak, és így ne rögzüljön véletlenül se rosszul.

A fentebb tárgyaltakban (8 fejezet) volt róla szó, hogy az adott betűtípus a felhasználónak nem jelenik meg, ha az nincs telepítve. Mivel itt használunk ilyet, ezért szükség volt a külön telepítésre még a program indulása előtt.

A problémák listájára felkerült a szótár során még az is, hogy eleinte JTextArea lett volna a fordítási mező, ezt azonban – a megjelenése miatt – JTextFielddel helyettesítettem. Ennek egyszerű az oka: a textFieldben lehet középre igazítani, viszont hiába kerestem megoldást, a textArea nem támogat ilyen funkciót.

A bővíthetőség is felmerült a szótár esetében – ahogy a többi funkciónál – is. Amennyiben ugyanígy XML formában van az adatbázis, elég könnyen adható hozzá új szó, hiszen a Java ehhez jó eszközöket biztosít. A kérdés csupán az, hogy hogyan oldjuk meg a jelek kiíratását is ezzel együtt, illetve új szó hozzáadása esetén ez hogy valósul meg, főleg ha ASCII karakterekként tárolhatóak.

Az, hogy milyen szavak kerülhetnek bele, mi számít alapvető szókincsnek, az elég könnyen felderíthető. A japán nyelvvizsgára lehetőség van Budapesten, és ott 4 fokozatot határoznak meg . Minden fokozathoz tartozik egy leírás és ebből, valamint az internet és könyvek segítségével összeállítható egy alapvető csomag, ami tartalmazza az egyes szintekhez tartozó szavak, leckék és tesztek összességét. Sajnálatos módon a nyelvvizsga csupán a saját magunk megerősítését szolgálja jelenleg, mivel nem tartozik hozzá szóbeli rész, ebből kifolyólag pedig nem elfogadott nyelvvizsga.

A fejezet végére hagytam egy részt a kódból, ami az XML fájlból olvassa be, hogy melyik szóhoz melyik a hozzátartozó fordítás. Gyakorlatban egy listába raktam a gyermekelemeket, és az XML-ben is két egymást követő gyerek a két nyelven szereplő szó. Ebből kifolyólag (gondoltam én) az elejétől indulva található majd meg a két elem. Ehhez képest, az első elem az egyes indexen foglalt helyet, a következő pedig a harmadik pozíción foglalt helyet, ezzel a kód az alábbi (2. kódrészlet) formát kapta.

```
if (langJp.isSelected()) {

for (Node i = root.getFirstChild(); i != null;
      i = i.getNextSibling()) {
            if (i.getNodeName().equals("word")) {
            NodeList words = i.getChildNodes();

      if (words.item(1).getTextContent().
            equals(toTranslate.getText())) {
            transResult.setText(words.item(3).getTextContent());
            found = 1;
            }
        }
      }
   }
```

2. kódrészlet
Keresés az XML-ben

Egy alkalommal kipróbáltam, hogy az XML-be raktam bele japán karakteres megfelelőt egy szó esetén, amit az XML megjelenít, de a program ezzel nem működött, viszont a

harmadik gyerekként szerepelve az 5. helyet kapta meg a listában, így kettesével lépkedve kerültek be az elemek.

Egy bugnak mondható a programban, hogy – bár a váltása le van kezelve – a rádiógombok időnként „beragadnak”. Első alkalommal a két rádiógomb önállóan lett a felülethez csatolva és egy if segítségével volt a váltás megoldva. Ekkor időnként egyszerre kettő is ki lehetett jelölve, majd kattintásra egyik sem. A másik megoldásnál csoportba foglaltam a rádiógombokat, ebben az esetben nem engedte az egyszerre való kijelölést, ugyanakkor időnként csak a jelölő pontot helyezte át, de nem végezte el a kattintáshoz kapcsolódó eseményeket.

Aki esetleg jobban megfigyelte a képeket láthatja, hogy itt is – mint mindenütt – le van tiltva az ablak átméretezése. Ez azon egyszerű ok miatt lett így, hogy mivel a háttér fix méretű, és az ablak széthúzásával együtt nem növekszik magától, és mivel a programfunkciók esetemben nem igényelték, hogy nagyobb felületen kelljen dolgozni, így ez volt a legegyszerűbb és legidőtakarékosabb megoldás.

9.6. Segítő táblázatok

Amint a funkciók bevezetésénél említettem (9.1 fejezet), két táblázat is elérhető olvasásra és tanulásra a felhasználónak (Hiragana táblázat (10. ábra), Katakana táblázat (9. ábra)). Ezek azon olvasatot tartalmazzák az egyes kanák esetében, amit használhatnak mind a szótár, mind a tesztek esetén, illetve ezt használja a program is kiíratásoknál. Az itt használt olvasat elsajátításához fontosak a táblázatok, hiszen nem feltétlenül tudná értelmezni a felhasználó mit is vár a program, ha egy másfajta olvasatot esetleg megtanult és itt nem lenne lehetősége megtanulni az általunk használtat. Ez a két tábla képként egy JLabel segítségével jelenik meg.

9. ábra

Katakana táblázat, a „katakana wo yomimasho!„ felirattal „Olvassunk katakanát!”. Itt a táblázat alján a felkiáltójeles mondatban látszik, hogy vannak összetettebb részek: például a sho – a mondat utolsó 3 karaktere a felkiáltójel előtt – 3 karakterből áll össze (si+jo+u).

Ez a két táblázat a főmenüből érhető el, külön menüpontokból – *HIRAGANA TÁBLÁZAT*, *KATAKANA TÁBLÁZAT*. Későbbiekben a további fejlesztésekkel együtt csatolható lenne hasonló módon a kanjikhoz tartozó táblázat is. A kanjik esetén az jelenthet problémát, hogy mivel több ezer kanjiról beszélünk, így egy nagy képként megjeleníteni nem a legjobb ötlet, ezért ezek valószínűleg szöveges formában szerepelnének, vagy sok kisebb képre szétszedve, valamilyen rendszer szerint. Másik megoldás lehetne, hogy a Ponyi Ildikó által készített nyelvtani részekhez kapcsolódva lennének az adott esetben szereplő kanjik kigyűjtve és olvasataikkal együtt feltüntetve.

10. ábra

Hiragana táblázat, az alján a „hiragana wo yomimasu!„ felirattal „Olvassunk hiraganát!”.

Ha már segítségről beszélünk, ahogy a program bonyolódik, úgy a későbbiekben szükség lehet egy menüpontra, ami leírást ad a használatról, mert – bár jelenleg egyértelműen adják magukat a használati esetek – tudjuk, hogy egy átlagfelhasználó nem feltétlenül találja majd ki, hogy mi hogyan működik.

A táblázatok, amelyeket használtam a fejlesztés során, nem saját készítésűek; próbáltam olyan képeket/táblázatokat keresni, amelyek illenek a program stílusához, illetve egyben az olvasat is megegyezik a programban használttal. A forrásjegyzékben megtalálható a két kép eredete és az alkotója is [14].

10. A design

A program készítése során igyekeztem felhasználóbarát megjelenítésre törekedni az egyes programrészek esetén, így a kész Yamo egésze egy stílusra lett alapozva. A háttér a színek a betűtípusok stb. próbálnak egymáshoz illeszkedni, amennyire erre lehetőségem volt. Mivel „japánosra” szerettem volna a programot, a háttér a Japánban közismert cseresznyevirágos témán alapul, és a logó, ami az ablak sarkában van szintén egy sakura virág, vagyis a japán cseresznyefa virága.

Mivel a virág maga rózsaszínű és a háttér emellett kékes, így a menüpontok és feliratok egy sötét lilás-kékes színt kaptak. A betűtípus vagy japán stílusú, vagy ahol szükségesek voltak magyar karakterek, ott a NetBeans designer által felajánlott alap betűtípus (Tahoma), ami szintén nem talpas betű, akárcsak az Arial.

A másik dolog a design esetében ami még felmerült, hogy a szokásos Windowsos témát ne kelljen belerakni, ami odáig működik is, hogy levesszük a keretet egy ablakról; ekkor viszont a mozgatást külön le kell kódolni: az egérhez kapcsolódó eseményeket (pl.: az ablak mozgatása) ami a kurzor koordinátái alapján történik. Ez önmagában nem probléma, de a keret nélkül nem volt túl tetszetős egy szögletes háttér. Ekkor felmerül, hogy kerekítsük le a szélét, de ez nem olyan egyszerű. Bár a kép, amely a hátteret kitölti megoldható, hogy lekerekített sarkú legyen, de maga a frame, ami tartalmazza a képet, nem változik olyan egyszerűen, így erre külön megoldás kellett volna. Ráadásul a gombok, szöveges mezők stb. ugyanúgy megmaradtak volna a Windows-stílusban, így maradt a keret és megpróbáltam ezzel együtt is elfogadható kinézetet összeállítani.

11. A felhasználókról

A felhasználói felületek alapvető szerepet játszanak, mert a felhasználó csak ezt látja, csak ezen keresztül találkozik a rendszerrel. Fizikai és mentális igények alapvetően középpontban kell, hogy legyenek.

11.1. Mentális elvárások

Szokás beszélni az úgynevezett mágikus hetesről [15]. Ez azt jelenti, hogy egy átlagember esetében a rövidtávú memória olyan, hogy átlagosan 7 dolgot tud megjegyezni. Amennyiben ennél több dolog rögzítésére kényszerül az agyunk, ezek vagy nem tárolódnak, vagy pedig jobb esetben a hosszú távú memória fogja őket rögzíteni.

Az ember elfárad, hibázik, tehát fel kell készíteni a felhasználói felületet (továbbiakban User Interface, UI) arra, hogy az embert minél kevésbé fárassza, másrészt arra, hogy a felhasználó hibázik, és ezt a hibázási lehetőséget minél hamarabb ki kell küszöbölni.

Az emberek mentálisan sokfélék, tehát az UI-kat az ízlések is befolyásolják: ki mit szeret látni, mi nem tetszene neki egy felületen, az információk hogyan nézzenek ki; ez sokféle lehet. Ez persze nem egyszerű, de egy általános középutat nem olyan vészes eltalálni. [12]

11.2. Fizikai elvárások

Az ember esetében több korlátozó tényező is felmerülhet: a színtévesztés, bizonyos fizikai hiányok például rövidlátás vagy nagyothallás stb. mind problémát jelenthet. Egyszóval lényeges a felhasználói felület akadálymentesítése. [12]

11.3. Ökölszabályok

A felhasználói jártasság elve*:* A felületen olyan terminológiát használunk, amely terminológiát a felhasználók értenek, ismernek, alkalmaznak. Eszerint a program azon olvasatot használja pl. a japán szavak esetében, amely olvasatot használnak máshol is, illetve, ahogyan azt a segítő táblázatok is megadják.

A felhasználói felület konzisztenciája: Két ugyanolyan dolog ugyanazt jelentse, ugyanúgy kelljen használni. Ha valamihez egyszer kell kattintani, akkor mindenhol egyszer kelljen. Az oktatóprogramban ez is megvalósul, pl.: kanatípusok kiválasztása, szöveges mezők szerkeszthetősége.

A minimális meglepetés elve: Általában van valamilyen működési profilunk, ami kialakul használat közben, nem történik más, amikor ugyanazt csináljuk két különböző felületrészen. Azaz a felhasználó nem ütközik abba a problémába, hogy egyik helyen nem úgy működik valami, mint ahogy várná. Az oktatóprogramban is így van, ha az egyik helyen ki kell választani pl. a kanatípusokat, akkor a másik helyen is így fog működni.

A visszaállíthatóság elve: Ellenőrző pontokat kell képezni. Ha például valamit menteni lehet, vagy törölni, akkor módosítás vagy törlés esetén a felhasználónak küldjünk üzenetet arról, mi fog történni, amennyiben jóváhagyja a műveletet. A program ezzel a funkcióval nem rendelkezik.

Felhasználói támogatás: A szoftverek egy fontos pontja a súgó. Egy jó súgó egy sok belépési ponttal rendelkező szöveggráf. Megtervezése nem egyszerű dolog, azonban egy bonyolultabb programnál (igazából átlagfelhasználónak mindenhol) szükségszerű. Sajnos tény, hogy jó és hasznos súgót ritkán látni. A program elég egyszerű, és utasítások is szerepelnek benne, ezért súgó jelenleg még nincs, azonban ha Ponyi Ildikó része is hozzákapcsolódik majd, akkor a későbbiekben indokolt lesz az elkészítése.

A felhasználói sokszínűség elve: Ez annyit takar, hogy milyenek legyenek a színek, a felhasznált betűk mérete stb. Erről részletesebb leírás fentebb a 10 fejezetben látható [12].

11.4. Felhasználói interakciók

A felhasználói interakciók közé tartoznak azon interakciók, amelyek során a felhasználó kommunikál a rendszerrel, parancsot ad, kijelöl valamit stb…

Közvetlen manipuláció*:* Ebben az esetben arról beszélünk, hogy a felületen valamilyen objektumok jelennek meg. Ezek olyan felületelemek, amelyeket a felhasználó valamilyen eszközzel közvetlenül tud manipulálni, például egérrel, és ez a külső eszközzel végrehajtott cselekvés eredményezi majd valamely esemény bekövetkezését.

- Előnye: Intuitív, könnyen tanulható és alkalmazható.
- Hátránya: Bizonyos esetekben nem implementálható.

Menürendszer: A menü megfelelő elemét kiválasztva idézi elő a felhasználó a megfelelő hatást. Pl.: kattintunk egy menüelemre és felugrik egy ablak.

- Előnye: Kevés gépelést igényel, könnyű a használata, olyan dolgok is megfoghatók, amik objektumokkal nem. Ráadásul egy kezdő felhasználó is felfedezi, hogy egy menüelem hogyan működik.
- Hátránya: A menürendszer bonyolulttá válhat, a hetedik almenüben nem fogjuk tudni, hogy mit is kellene keresni. Szintén problémás, hogy egy idő után egyes felhasználóknak lassú lehet a használata.

Űrlap kitöltés: A szükséges adatokat bevinni elsődlegesen a billentyűzetről tudjuk. Az űrlap tulajdonképpen csak a struktúrát jeleníti meg.

- Előnye: Ez is elég egyértelmű, és könnyen használható.
- Hátránya: A gond többféle módon jelentkezhet ebben az esetben. Lehetséges például, hogy rosszul lett megtervezve vagy például kitölthetetlen, esetleg megenged olyan kitöltési módot, ami a későbbiekben probléma forrása lehet.

Parancsnyelv: Ez nem más, mint a grafikus világ előtti kommunikáció.

- Előnye: Ez a legrugalmasabb megoldás, jobb, mint a közvetlen manipuláció, tömör.
- Hátrány: Meg kell tanulni, rossz a hibakezelése, ha valamit elgépelünk, akkor nem jelzi, csak maximum mikor valami nem úgy működik, ahogy kellene.

Természetes nyelv*:* Nem szabványos, de próbálják elterjeszteni. Egyrészt egy speciális parancsnyelv, ami magyarul van, és mindezt írjuk. Egy másik fajtája a korlátozott szókészletű beszéd felismerése.

Webes felület*:* Az UI egy másik módszerrel történő megvalósítása a webes felületek létrehozása. Itt is ugyanazokat az elemeket használjuk csak valamilyen scriptnyelvvel. A napjainkban nagyon (egyre jobban) elterjedt megoldás.

12. Összefoglalás

A szakdolgozatom végére érve szeretném összegezni, hogy milyen pozitívumokkal illetve negatívumokkal tudnám leírni az elkészült programot, illetve szeretnék szólni pár jövőbeli tervről is.

A szoftver fejlesztése előtt összegyűjtött terveket sikerült megvalósítani és azokon kívül is került bele utólagosan felmerült ötlet pl.: a Kana demo (9.3 fejezet). Sokszor tapasztaltam, hogy az elméletben elgondolt dolgok gyakorlatban nem, vagy nem úgy működtek. Ez persze nem jelent problémát alapvetően és tapasztalatnak is jó volt (9.5 fejezet). Ugyanakkor a Swingen látszik, hogy régóta nem fejlesztik, illetve az is, hogy sok esetben túl van bonyolítva egy-egy dolognak a megvalósítása (pl.: a hátteret nem lehet egyszerűen a background színek mellett valahol meglelni és beállítani, hanem JLabel szükséges hozzá; az átlátszóság szintén külön helyen, opaque néven szerepel – nekem transparent lenne a logikusabb).

A készítés során szinte minden egyes részénél jöttek elő újabb tervek, amelyeket a későbbiek során majd meg szeretnék valósítani. Itt megemlíteném, hogy ez a szoftver egy a jövőben elkészülő hasonló oktatóprogram tesztverziójának készült számomra. Nemrég segítettem egy koreai barátomnak, aki koreai és magyar emberekkel dolgozott együtt azon, hogy egy koreai-magyar szótárat hozzanak létre. Ehhez szeretnék majd egy komolyabb programot készíteni, amely nem csupán szótár, hanem egy ennél is több, tanulást segítő funkciókkal rendelkező szoftver lenne. Mivel hasonlóan a japánhoz a koreai nyelv is saját ábécével rendelkezik, hasonló ötletekkel megvalósítható lenne, mint a szakdolgozatként készült program.

Ami összességében tehát negatívum volt, az az egyes esetekben megjelenő túlságos bonyolítása volt a Swingnek. A Javában viszont plusz tapasztalat és gyakorlat volt ezt elkészíteni, főleg ami a grafikus részeket illeti. Szerencsére a japános téma közel áll hozzám, mind a japán nyelvet mind magát a kultúrát tekintve így különösen élveztem a készítését ennek az oktatóprogramnak.

13.Függelék

```
Fájl f = null;
f = new Fájl(Main.class.getProtectionDomain().
            getCodeSource().getLocation().toURI().getPath());

int ind = f.getAbsolutePath().lastIndexOf("\\");
String outOfJar = f.getAbsolutePath().substring(0, ind + 1);
```

3. kódrészlet
A mappa elérési útja, amelyben a jar fájl található

```
{
URL jar = src.getLocation();
ZipInputStream zip = null;
      try {
            zip = new ZipInputStream(jar.openStream());
      } catch (IOException ex) {
            Logger.getLogger(Kana.class.getName()).
            log(Level.SEVERE, null, ex);
      }
ZipEntry ze = null;

      try {
            while ((ze = zip.getNextEntry()) != null) {
            String entryName = ze.getName();
                  if (entryName.startsWith("kana/hiragana") &&
                        entryName.endsWith(".PNG")) {
                  list.add(entryName);
                  } else if (entryName.startsWith("kana/katakana")
                  && entryName.endsWith(".PNG")) {
                  list2.add(entryName);
                  } else if (entryName.startsWith("kana/allkana")
                  && entryName.endsWith(".PNG")) {
                  list3.add(entryName);
                  }
            }
      } catch (IOException ex){
                  Logger.getLogger(Kana.class.getName()).
                  log(Level.SEVERE, null, ex);
      }
}
```

4. kódrészlet
Olvasás a JARon belülről

```
public void copyDirectory(File srcPath, File dstPath) throws IOException {

      if (srcPath.isDirectory()) {
            String files[] = srcPath.list();
            for (int i = 0; i < files.length; i++) {

                  copyDirectory(new File(srcPath, files[i]),
```

```
                    new File(dstPath, files[i]));
                    }
                } else {
                InputStream in = new FileInputStream(srcPath);
                OutputStream out = new FileOutputStream(dstPath);
                byte[] buf = new byte[1024];
                int len;
                while ((len = in.read(buf)) > 0) {
                out.write(buf, 0, len);
                }

                in.close();
                out.close();
        }
}
```

5. kódrészlet
Font másolás adott mappából adott mappába

14.Köszönetnyilvánítás

Ezúton szeretném megköszönni Dr. Csernoch Mária tanárnőnek – aki remek tesztelőnek bizonyult – a sok hasznos útmutatást.

Köszönöm Ponyi Ildikónak is, aki nélkül nem lett volna egész ez a projekt.

Szeretném még megköszönni azoknak is, akik végig támogattak a dolgozat írása során.

15.Irodalomjegyzék

[1] A japán nyelv: http://hu.wikipedia.org/wiki/Jap%C3%A1n_nyelv 2011. 04. 15.

[2] A Jar fájl: http://hu.wikipedia.org/wiki/JAR_%28f%C3%A1jlform%C3%A1tum%29 2011. 04. 15.

[3] A Java: http://hu.wikipedia.org/wiki/Java_%28programoz%C3%A1si_nyelv%29 2011. 04. 15.

[4] OO programozás:http://hu.wikipedia.org/wiki/Objektumorient%C3%A1lt 2011. 04. 15.

[5] Az XML: http://hu.wikipedia.org/wiki/XML 2011. 04. 15.

[6] A Swing: http://javagrund.hu/javasite/dokument/swing/swing.html 2011. 04. 15.

[7] A DOM: http://problemjava.blogspot.com/2007/06/dom-sax-stax.html 2011. 04. 15.

[8] A Java: http://www.cab.u-szeged.hu/WWW/java/kiss/article.html 2011. 04. 15.

[9] Az XML: http://www.cid.com.ro/ksimon/upload/xml.pdf 2011. 04. 15.

[10] Hiragana és katakana táblázat: http://www.feyrer.de/JP/hk.gif 2011. 04. 15.

[11] A japán nyelv:http://www.keletinyelvek.hu/index.php?page=japannyelv 2011. 04. 15.

[12] Juhász István – A rendszerfejlesztés technológiája jegyzet

[13] Juhász István – Magas szintű programozási nyelvek 2 jegyzet

[14] Segítő táblázatok: http://pullmeoutalive.deviantart.com/gallery/2563069 2011. 04. 15.

[15] Felhasználók, „mágikus hetes”: http://menedzser.ymmf.hu/files/menedzser/tantargyak/pszichologia/pszichologia_07ea.pdf 2011. 04. 15.

[16] AWT: http://hu.wikipedia.org/wiki/Abstract_Windowing_Toolkit 2011. 04. 15.

[17] Kanateszt 1: http://www.realkana.com/ 2011. 04. 21.

[18] Kanateszt 2:http://www.easyjapanese.org/kanaquiz.html 2011. 04. 21.

[19] Kanateszt 3: http://www.donmouth.co.uk/web_design/javascript/kana_test_001.html 2011. 04. 21.

[20] Japán nyelvi szintek: http://www.jlpt.jp/cn/about/levelsummary.html 2011. 04. 21

[21] Multimodalitás: http://books.google.hu/books?id=_QmV-oN3rP4C&pg=PA143&lpg=PA143&dq=multimodalit%C3%A1s+tanul%C3%A1s&source=bl&ots=6Sev5eNGhS&sig=R-HIAsQiyOe848QtglHl7Cxc9E0&hl=hu&ei=Zi6wTaGpMs_rsgaQ5J3jCw&sa=X&oi=book_result&ct=result&resnum=7&sqi=2&ved=0CEYQ6AEwBg#v=onepage&q&f=true 2011. 04. 22.