Szakdolgozat

Koszczelnik Tamás

Debrecen

2008

Debreceni Egyetem
Informatikai Kar

ACM programozási verseny lebonyolítását végző szoftver írása

Témavezető:
Kósa Márk
egyetemi tanársegéd

Készítette:
Koszczelnik Tamás
programozó matematikus

Bevezetés

Amikor megpillantottam ezt a szakdolgozat kiírást a felsorolás közepén, rögtön éreztem, hogy ezt nekem találták ki. Izgatottan vártam, hogy belekóstoljak a programozási versenyek világába, amire még soha nem volt lehetőségem. Elhatároztam, hogy a program megvalósításához számomra eddig teljesen ismeretlen technológiákat használok majd fel, hiszen tanulni mindig jó. És végül megfogadtam, hogy a szakdolgozatom nem csupán száraz kódelemzés lesz: a programom bemutatásán kívül szeretnék beszélni az ACM versenyről is, továbbá betekintést szeretnék nyújtani egy fiatal, zseniális programozási technológiába, a JSP-be is.

ACM

Számomra még az egyetemi körökben is meglepően kevesen tudják, mit is takar az ACM rövidítés és még azok is, akik tudni vélik, általában tévesen csupán egy programozási verseny neveként tartják számon. Pedig ez a három betű ennél sokkal többet takar. Az ACM (**A**ssociation for **C**omputing **M**achinery), a világ egyik legrangosabb és egyben legnagyobb informatikai társasága. 1947-ben alapították, hogy összefogja az információ-technológia vezető tudósait és tanulóit és fórumot biztosítson a tudományos informatika ötleteinek, felfedezéseinek, információinak kicseréléséhez. A társaság azóta együtt fejlődik az érdeklődését képező tudományággal, tagjainak száma világszerte lassan eléri a százezret.

És hogy mit is csinál napjainkban egy ilyen hatalmas, világméretű társaság?
Online fórumokat biztosít az informatika szakemberei számára és online kurzusokat azoknak, akik ebben a tudományágban szeretnék tudásukat és tehetségüket kamatoztatni.
Számos folyóiratot és magazint publikál, pl. ACM Crossroads, ACM Transactions on Programming Languages and Systems. A társaság folyóirataiban láttak napvilágot olyan híres cikkek, mint pl. Dijkstra elmélete a szemafor-alapú szinkronizációról, vagy Ronald L. Rivest,

Adi Shamir és Leonard M. Adleman első publikus írása a nyilvános kulcsú titkosítás elméletéről és az RSA algoritmusról. Informatikai társaság lévén, ezeknek a cikkeknek óriási hányada hozzáférhető elektronikus formában is, ezt a gyűjteményt nevezik ACM Digital Library-nak.
Számos informatikai konferenciát szponzorál, ilyenek pl. a International conference on Object-Oriented Programming, Systems, Languages, and Applications, vagy az International conference on World Wide Web.
Olyan híres tudományos események szervezésében vesz részt, mint a híres sakkmeccs Garry Kasparov és az IBM Deep Blue számítógép között.
Díjakat oszt azoknak, akik a legtöbbet tesznek a számítástechnika fejlődéséért. Ezek közül a leghíresebb az Intel céggel közösen felajánlott ACM Turing Award, amit többek között olyanok kaptak meg, mint Dennis M. Ritchie, vagy Niklaus Wirth.
Végül, de nem utolsó sorban az ACM több – nemritkán világméretű – tudományos verseny szervezéséből és támogatásából is kiveszi a részét. Ezek közül az egyik legismertebb az ACM International Collegiate Programming Contest.

ACM International Collegiate Programming Contest

Az ACM ICPC – vagy más néven egyszerűen csak „Az agyak csatája” - egy évente megrendezett többfordulós programozási verseny a világ egyetemei között. Tulajdonképpen ez a világ legrégebbi, legnagyobb és legrangosabb programozási versenye. Egy olyan kőkemény megmérettetés, ahol a versenyző csapatoknak valós élet szülte problémákra kell szoftveres megoldást találniuk, versenyt futva egymással és a könyörtelen határidővel. Egy küzdelem, melynek nyomással teli órái próbára teszik a kreativitást, az innovatív gondolkodást, a csapatmunkára való képességet, a precizitást.

Az ACM ICPC története

Az ACM ICPC gyökerei egészen 1970-ig nyúlnak vissza, amikor a Texas A&M Egyetemen meghirdettek egy házi, egyfordulós programozó versenyt, akkor még az Upsilon Pi Epsilon társaság kizárólagos támogatásával. Ez a szintén nagy múltú egyesület a mai napig is az esemény aktív támogatói közé sorolandó. A verseny jelenlegi többfordulós formáját 1977-re érte el, amikor is már olyan hírnévre tett szert, hogy az – akkorra már az informatika tudományában meghatározó szerepet játszó - ACM társaság felajánlotta, hogy a döntőt az ACM Computer Science Conference keretein belül tartsák meg.

1977 és 1989 között a versenyen résztvevő csapatok túlnyomó többségben az Egyesült Államok és Kanada egyetemeiről kerültek ki. 1989-ben a szervezési központ átkerült a Baylor Egyetemre (és a mai napig ott is onnan végzi munkáját a verseny jelenlegi központi embere, gondoskodó atyja, Professor William B. Poucher). Ekkorra az ACM már a küzdelem legnagyobb támogatójának számít. A Baylor irányításával egy világméretű hálózat szerveződött az egyetemek közt a verseny zászlaja alatt, ami helyi selejtezők és a döntő mellett lehetővé tette országos és regionális „középdöntők” rendezését is.

1997 óta az ACM társaság mellett a legnagyobb hivatalos szponzor az IBM. A világ egyik legnagyobb és legismertebb informatikai cégének neve új lendületet adott a verseny fejlődésének és ennek köszönhetően ma hat kontinens 83 országának 1821 egyeteme küldi harcba csapatait az ACM ICPC fődíjáért. Mivel a helyi selejtezőkön bárki részt vehet, a fenti számok alapján könnyen kikalkulálható, hogy a verseny első fordulóján évente több tízezer vállalkozó kedvű informatikát hallgató fiú és lány teszi próbára magát, és ezek a számok várhatóan tovább növekszenek majd az elkövetkezendő évek során.

A verseny hivatalos logója

Szabályok

Az ACM ICPC egy csapatjáték. A jelenlegi szabályok szerint egy csapat három versenyzőből áll. A résztvevőknek egyetemi hallgatóknak kell lenniük, de a verseny évét megelőzően összesen maximum öt évet tölthetnek el felsőoktatási intézmény tanulójaként. Egy fő legfeljebb két döntőn, vagy öt regionális középdöntőn vehet részt.
A csapatoknak 5 órájuk van arra, hogy minél többet megoldjanak a 8-11 problémát tartalmazó „problémahalmazból". (A regionális fordulókban általában 8, a döntőn általában 10 feladat a megszokott). A problémahalmazt leíró dokumentum meglepően rövid instrukciókat tartalmaz a feladatokhoz, ezen kívül 1-2 bemeneti adatsorozat – kimeneti adatsorozat párt, annak ellenére, hogy a zsűri egy megoldást több száz, esetleg több ezer tesztadatra is lefuttat. Mindezt nem csupán azért teszik, hogy az algoritmus helyességét vizsgálják, hanem hogy megállapítsák, hogy a megoldás előállítása elég gyorsan történik-e. Ez természetes követelmény, hiszen – ahogy már említettem – a problémákat az élet szüli, a leggyorsabb megoldásokat hamarosan valódi, komplex szoftverrendszerekbe építik be, olyanokba, amelyek sokak számára elképzelhetetlen méretű adattömegeken operálnak. Ezért tehát a beküldött programoknak másodpercek alatt le kell futniuk a több ezer tesztadatra, vagy különben nem tekinthetők megoldásnak – akkor sem, ha a végeredmény amúgy tökéletes. Tulajdonképpen ez a néhány másodperces időkorlát az igazán jó megoldásoknak bőven elég, hiszen ezek logaritmikus, vagy maximum lineáris időbonyolultságú algoritmusok, míg egy exponenciális időbonyolultságú megoldás percekig, sőt, akár órákig is futhat.
A megoldást szolgáltató algoritmust C, C++, Java, vagy Pascal nyelven kell lekódolniuk. A csapatok pontos visszajelzést kapnak a beküldött feladataik értékeléséről, tehát egy helytelen, vagy lassú kód beküldése után előállhatnak egy újabb megoldással is – ahányszor csak szükséges. Ez talán amatőr jellemvonásnak tűnhet egy ilyen rangos verseny esetén, de lentebb látni fogjuk, hogy ez egyáltalán nincs így.
A győztes az a csapat, amelyik az idő lejárta előtt a legtöbb helyes megoldást küldte be. Nyilván ilyen kevés számú feladat esetén igencsak sűrűn fordul elő holtverseny, vagyis

rengeteg csapat küld be azonos számú problémát megoldó programot. Ebben az esetben az a döntő, hogy ki mennyi idő alatt készítette el a jó megoldásokat – természetesen minél hamarabb, annál jobb. Nagyon fontos, hogy minden egyes beküldött helytelen, vagy túl lassú algoritmus 20 perces időbüntetést von maga után, szóval tanácsos alaposan átnézni a kódot olyan szemantikai hibák után kutatva, amelyek az 1-2 általános tesztadatra nem, csak esetleg szélsőséges inputra futtatva jönnek elő, vagy átgondolni, hogy a megoldás előállítása elég gyors lesz-e. A kapkodás tehát nem hozza meg a gyümölcsét és a fent említett amatőrnek tűnő szabály igazából megszűri a mezőnyt, hiszen könnyen kerülhetnek akár órákkal is hátrányba azok a csapatok, amelyek tagjai – programozóhoz nem méltó módon gondolkodva – úgy vélik, hogy a tesztbemenetre adott helyes kimenet a program tökéletességét jelenti.

Szabályok a gyakorlatban

Nézzünk egy egyszerűsített példát arra, hogyan is alakul ki az indulók végső sorrendje! Vegyünk két csapatot, nevezzük őket Kék és Piros csapatnak. Tegyük fel, hogy mindketten két-két problémát oldottak meg helyesen, vagyis a beküldött feladatok számát tekintve holtversenyben vannak. A Piros csapat kerek 1 órával a verseny kezdete után beküldött egy helyes megoldást az A feladatra, és 2 óra 45 perccel a verseny kezdete után egy helyes megoldást a B feladatra. Beküldtek egy algoritmust a C-re is, de az túl lassúnak bizonyult és már nem volt idejük gyorsítani rajta. Lényeges, hogy ezért ők nem kapnak időbüntetést, mivel a plusz 20 perc csak megoldások idejéhez adódik hozzá!
A Kék csapat az A és a C problémát oldották meg, 1 óra 20 perc és 2 óra alatt. A C megoldását csak másodjára sikerült tökéletesen megalkotni, tehát ezért ők egyszer 20 perc időbüntetéssel lettek „gazdagabbak”.
Végül tehát a Piros csapat összesített ideje 1:00 + 2:45 = 3:45, míg a Kék csapaté 1:20 + 2:00 + 0:20 = 3:40, vagyis a győztes a Kék csapat.

Most, hogy a kedves Olvasó tisztában van azzal, mi is az ACM programozási verseny, rátérhetek egy ilyen versenyt lebonyolító program bemutatására, írásának körülményeire. Az általános követelmények felsorolása után először felhasználói szempontból közelítem meg a szoftveremet, majd a fejlesztés részleteibe engedek betekintést.

Mit kell tudnia egy ACM versenyt lebonyolító programnak?

Az ACM versenyt lebonyolító programot a zsűri és versenyző csapatok a verseny egész időtartama alatt számos dologra használják. A versenyzők szempontjából a program a következő funkciókat látja el (külön kezelve tetszőleges számú csapatot):

- Fogadja és kezeli a csapatoktól érkező forráskódokat. A kódokból forrásfájlt készít, amit lefordít, majd – sikeres fordítás esetén – lefuttat az adott problémához tartozó tesztadatokra. A futás eredményét összehasonlítja a tesztadatokhoz előre előállított kimenetekkel és amennyiben 100%-ban egyeznek és a futási idő nem lépett át egy előre beállított időkorlátot, a beküldött algoritmust megoldásnak tekinti, egyébként nem.
- Tárolja a csapatok eredményeit és a beküldött megoldások száma, valamint a beküldési idők alapján egyértelmű sorrendet állít fel az indulók között. Az eredmények publikusak a versenyzők és természetesen a zsűri számára is.
- Lehetőséget biztosít, hogy a csapatok kommunikáljanak a zsűrivel. Bármikor lehet a problémákkal kapcsolatban kérdéseket küldeni a zsűrinek, aki vagy válaszol rá, vagy sok esetben csak megkéri a csapatot, hogy újra olvassák el és gondolják át a feladat kiírást, mert a válasz ott lapul a sorok között.

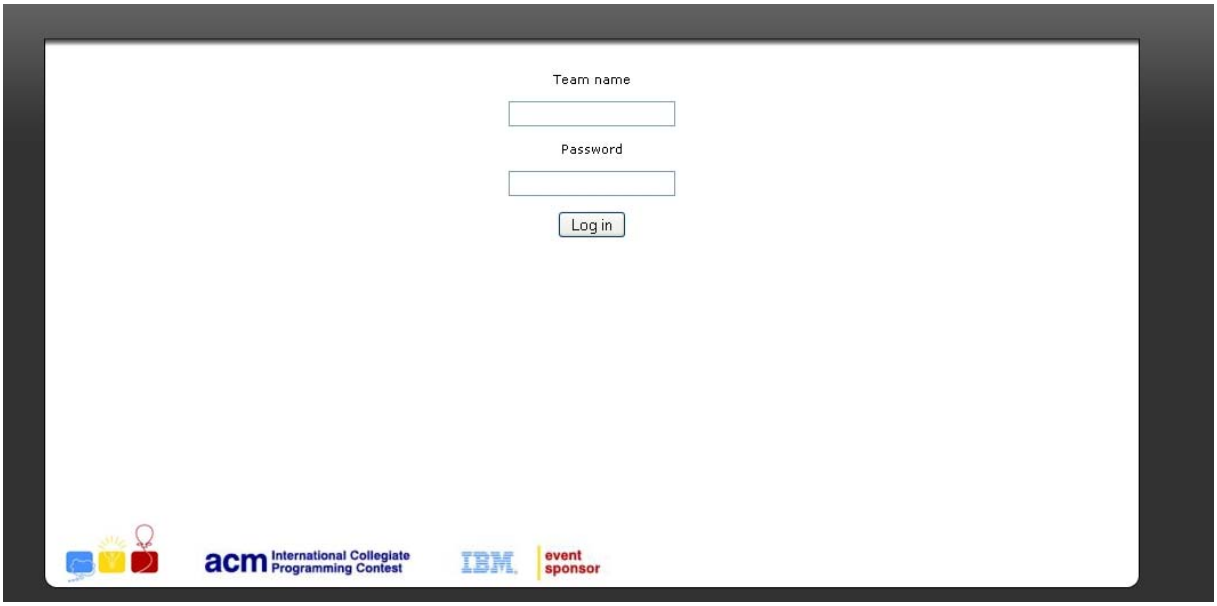
A zsűri tehát a verseny állásának nyomon követésére és a csapatok üzeneteinek megválaszolására használja a szoftvert, valamint a verseny legelején az induló csapatok rögzítését is ők végzik.

A továbbiak folyamán szeretném a felhasználói felületet végignézve bemutatni a programomat. Ez a fejezet tulajdonképpen kézikönyvként is használható, bár mivel nem

kimondottan egy nagyvállalati szoftverrendszerről beszélünk, a használata elég egyértelmű, főleg ha fontolóra vesszük a tényt, hogy a felhasználók mind informatikusok lesznek.
A technikai megvalósítást illetően annyit szeretnék itt megjegyezni, hogy a program web-es technológiákon alapul, tehát a kliensprogram szerepét egy tetszőleges böngésző tölti be. A képernyőképek készítésénél a böngészőablakot helytakarékosság céljából – és hogy a program komponensei jobban kivehetők legyenek - rendre levágtam a képről.
Ahogy látni lehet, a szoftver angol nyelvű, így lehetőség nyílhat majd arra, hogy egy későbbi – tökéletesített – változat esetleg nemzetközi versenyek lebonyolítására is alkalmas legyen.

A program felhasználói szemmel

A bejelentkező képernyő

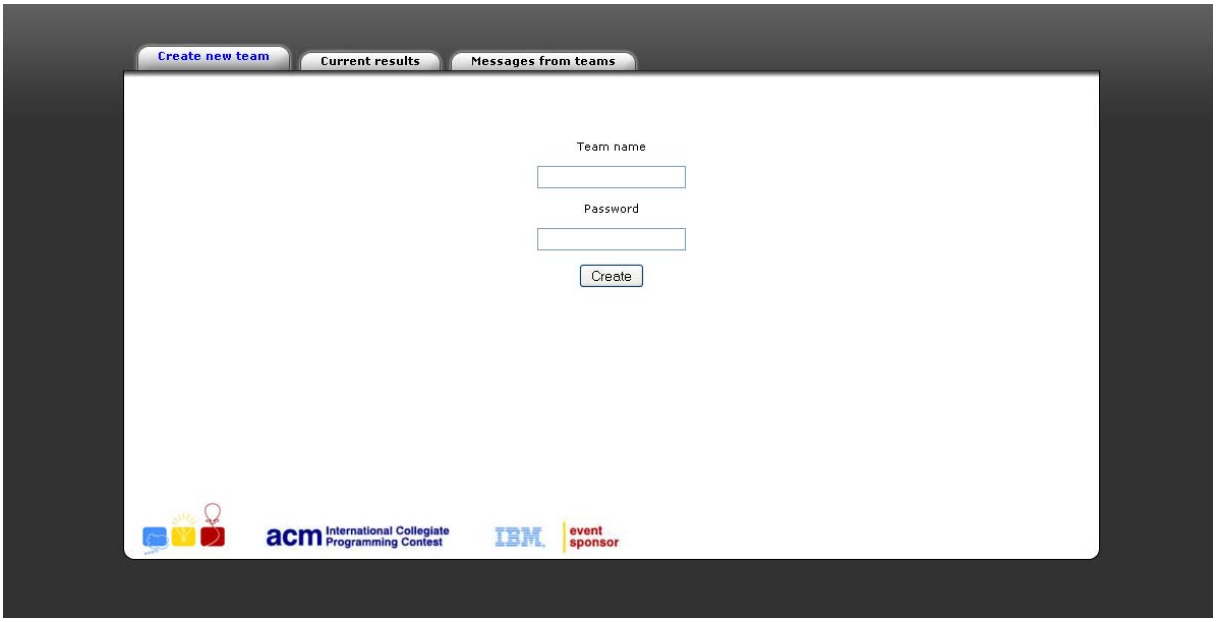


Bejelentkezéskor egyszerűen meg kell adni a csapatnevet és a jelszót. A nevét minden csapat maga választhatja meg, míg a jelszavakat a zsűri osztja ki, akinek mellesleg a „jury” szó, mint csapatnév fenn van tartva. Bár ők is ugyanitt jelentkeznek be, ezt a nevet használva a

továbbiakban csak a zsűri által használt funkciók érhetők el, és természetesen a csapatok is csak a versenyzéshez szükséges lapokat használhatják.
Hibás név, illetve jelszó begépelésekor a rendszer nem küld hibaüzenetet, csupán visszanavigál ugyanerre a képernyőre és a belépési kísérlet megismételhető – akárhányszor.

A zsűri által használt oldalak

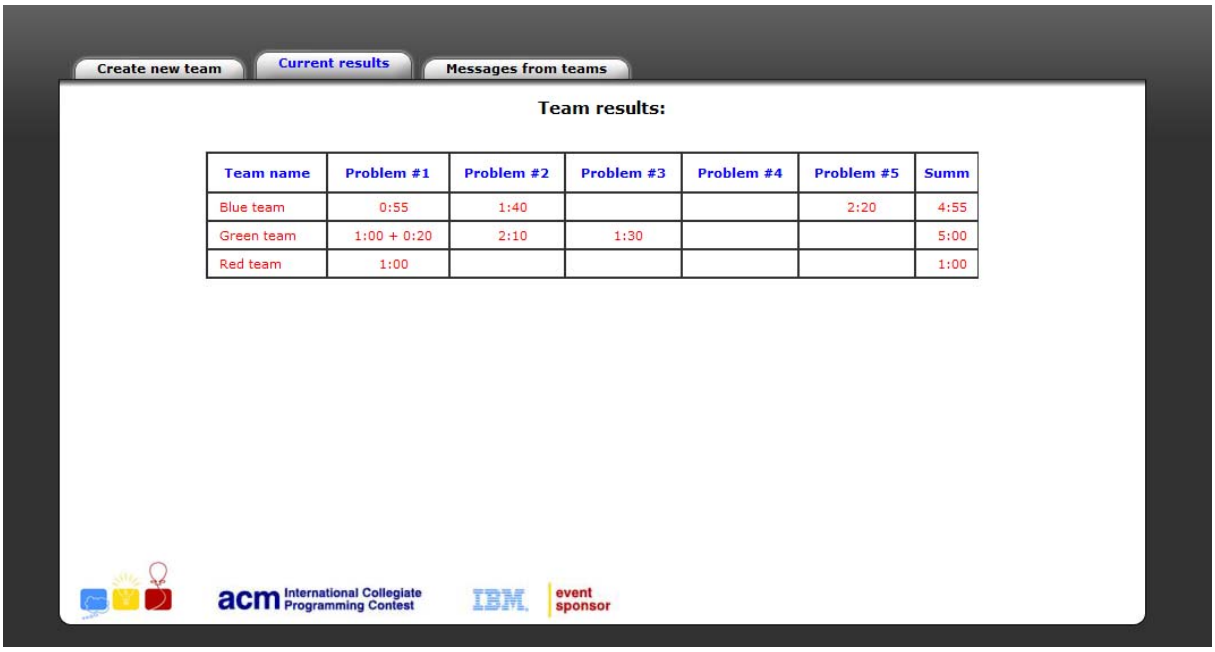
Új csapat létrehozása



Egy új csapat rögzítéséhez a rendszerben csak a csapat nevére és a jelszóra van szükség. Egyelőre a jelszót a zsűri találja ki és sem arra, sem a csapatnévre nincs megkötés, vagyis a szoftvert használó személyeknek kell gondoskodni arról pl., hogy két ugyanolyan nevű csapat ne szerepeljen a rendszerben. Szeretném a – még kezdetleges állapotban levő – programot továbbfejleszteni és terveim között szerepel többek között az adatok validálása és automatikus jelszógenerálás is.

Az oldalon látható megjelenítési elemek jellemzik az egész felhasználói felületet: ezüstszínű háttér, az ACM ICPC hivatalos logó-ja (kissé feldarabolva) mint lábléc, valamint lapfüles navigálási lehetőség az oldalak között.

A verseny állása



A program egy táblázatban foglalja össze a verseny pillanatnyi állását jelentő információkat. A táblázat egy sora egy adott csapathoz tartozó adatokat tartalmaz: a csapat nevét, a problémák sorszámai szerinti sorrendben a megoldásaik beküldési idejét, majd végül az összesített időt. Ha egy problémához még nem küldött be megoldást a csapat, akkor a megfelelő cella üres. Az időbüntetések egyértelműen jelezve vannak, ahogy az látható a „Green team” nevű csapat „Problem #1” jelzésű problémához készített megoldás beküldési idejénél is.
A csapatoknak a táblázatbeli sorrendje a verseny pillanatnyi ranglistáját mutatja. A képről tehát könnyedén leolvasható, hogy jelenleg a „Blue team” vezet 3 beküldött megoldással és 4 óra 55 perces összesített idővel, a második helyet a „Green team” foglalja el, aki szintén 3 feladatot oldott meg, de 20 perces büntetése miatt időhátrányban van az élcsapattal szemben, a sereghajtó pedig a „Red team”, hiszen ők csak egy problémára találtak eddig megoldást.

Az oldal néhány másodperces időközönként frissíti önmagát, így mindig a pontos állást lehet leolvasni a képernyőről.

A csapatoktól érkező üzenetek megtekintése

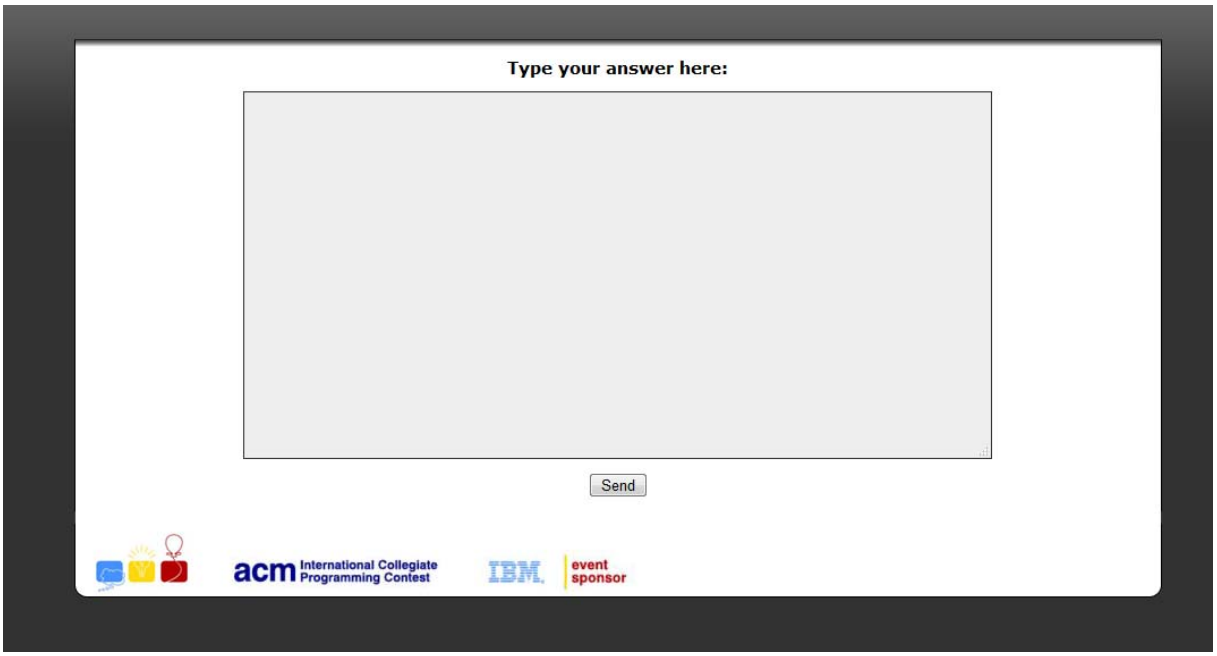
Ahogy már említettem, a versenyzők kapcsolatban állnak a zsűrivel és bármikor kérdésekkel bombázhatják őket. A kérdésekre mindig érkezik valamilyen reakció – ami nem feltétlenül jelenti a választ.

A zsűri a fent látható oldalon követheti nyomon a nekik küldött üzeneteket. Az üzenet szövegén kívül a feladót is láthatják, mindezt érkezésük szerint rendezve.

Ez az oldal is automatikus frissítésre van beállítva, ami garantálja, hogy a kérdések rövid időn belül célhoz érnek.

A zsűritagok az „Answer it” linkre kattintva tudnak egy-egy üzenetre reagálni, ami a lent látható űrlapra hivatkozik. Ez az űrlap nem szerepel a lapfülek alatt, hanem előugró ablakban jelenik meg.

Válasz egy üzenetre

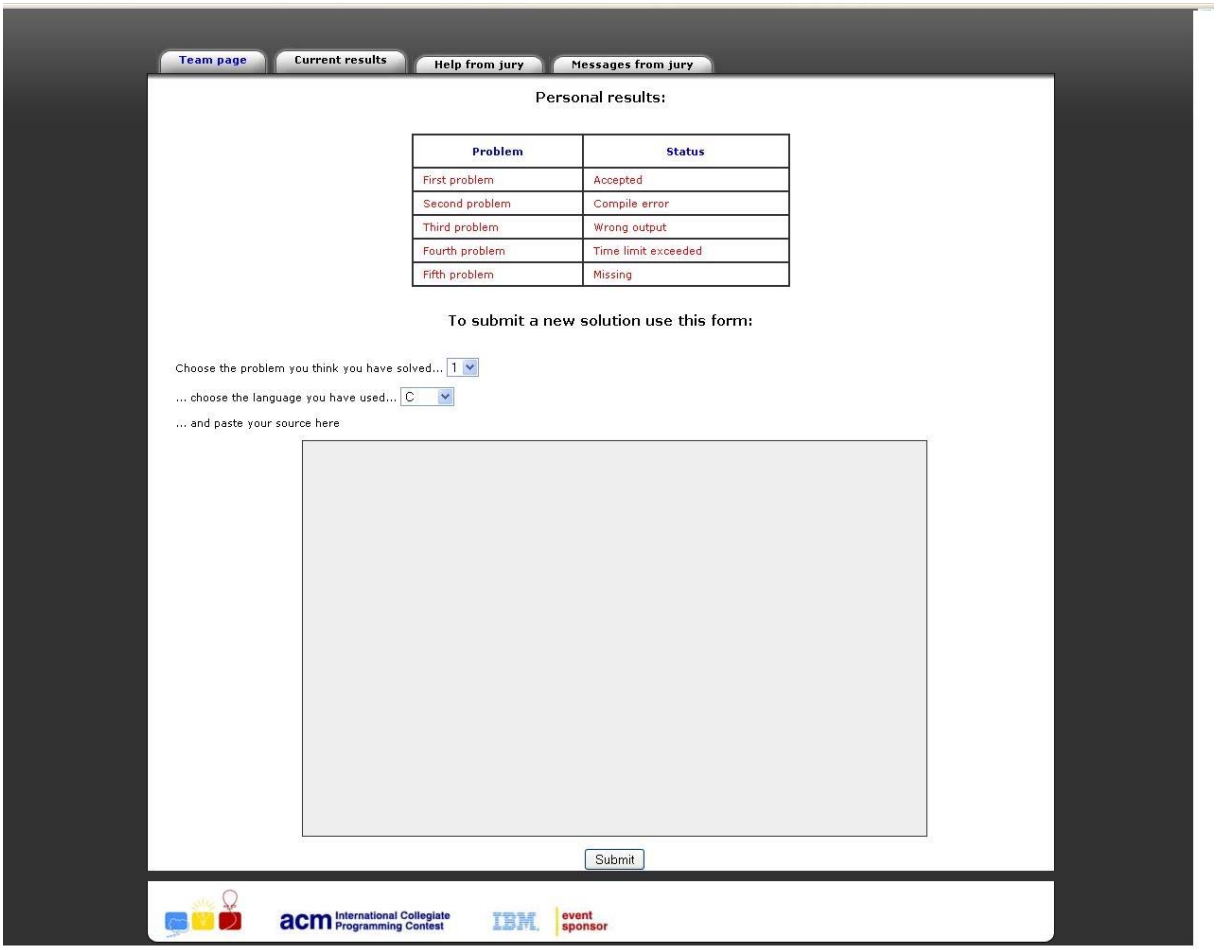


Az űrlap elég egyszerű, a begépelt szöveget a „Send” gombra kattintva lehet elküldeni. Látható, hogy nincs lehetőség kiválasztani, melyik csapatnak megy a válasz. Ez azért van, mert az üzenetet automatikusan az a csapat kapja meg, amelyik azt a kérdést írta, amelyik alatti linkre kattintva ugrott fel ez a lap.

A csapatok által használt oldalak

Ahogy már írtam, a versenyző csapatok és a zsűri ugyanazt a bejelentkező képernyőt használja a rendszerbe való belépéshez. Vannak még lapok, amelyek szinte teljesen megegyeznek valamelyik eddig megismerttel, de ezek ismertetése előtt a csapatok által leggyakrabban használt, legtöbb funkciót nyújtó oldalt mutatom be.

A „homepage”



Az oldal funkcionálisan két részre bontható. A felső részen egy kis táblázat mutatja, hogy az adott csapat hogyan áll a problémák megoldásával. A fenti mintapéldát úgy dolgoztam ki, hogy egy probléma összes lehetséges állapotát lefedje. Ezek az állapotok:

- „Accepted”: a csapat beküldte a probléma megoldását

- „Compile error”: a versenyzők beküldtek egy programot, de az fordítási hibát okozott. Bár kissé különös lenne, ha valaki úgy küldene be kódot, hogy még csak meg sem próbálta lefordítani, de elviekben lehetséges, sőt később látni fogjuk, hogy fordítási hiba más miatt is előállhat, tehát fel kellett készülni erre az eshetőségre is
- „Wrong output”: a beküldés ebben az esetben is megtörtént, sőt a kód le is fordult, de nem ugyanazt a kimenetet adta futáskor, mint az előre elkészített tesztkimenet. Mivel a szoftver karakterről karakterre hasonlít, ezért előfordulhat olyan eset is, hogy bár az algoritmus megoldása a problémának, de a kimenet előállításának kódolásakor a csapat formai hibát vétett. A legtipikusabb formai hiba az extra szóköz, vagy szóközök az output-ban, hiszen ezt a legnehezebb észrevenni. Erre tehát érdemes különös figyelmet fordítani a versenyzőknek.
- „Time limit exceeded”: ezt a feliratot akkor látja a csapat egy feladat neve mellett, ha találtak ugyan egy megoldást a problémára, de a kimeneti adatok előállítása túl lassan történik. Sajnos nem kapnak pontos adatot arról, hogy mennyi ideig is futott a program. Ez azért hátrány, mert előfordul, hogy csupán tizedmásodpercekkel lépte túl a futási idő a limitet, vagyis a kód gyorsításával helyes megoldást állíthat elő a csapat. Ha viszont az algoritmus időbonyolultsága is rossz, akkor a versenyzők sokat veszítenek azzal, ha apróságokkal próbálja meg gyorsítani a programot.
- „Missing”: a legegyszerűbb szituáció, a csapat még nem küldött be kódot.

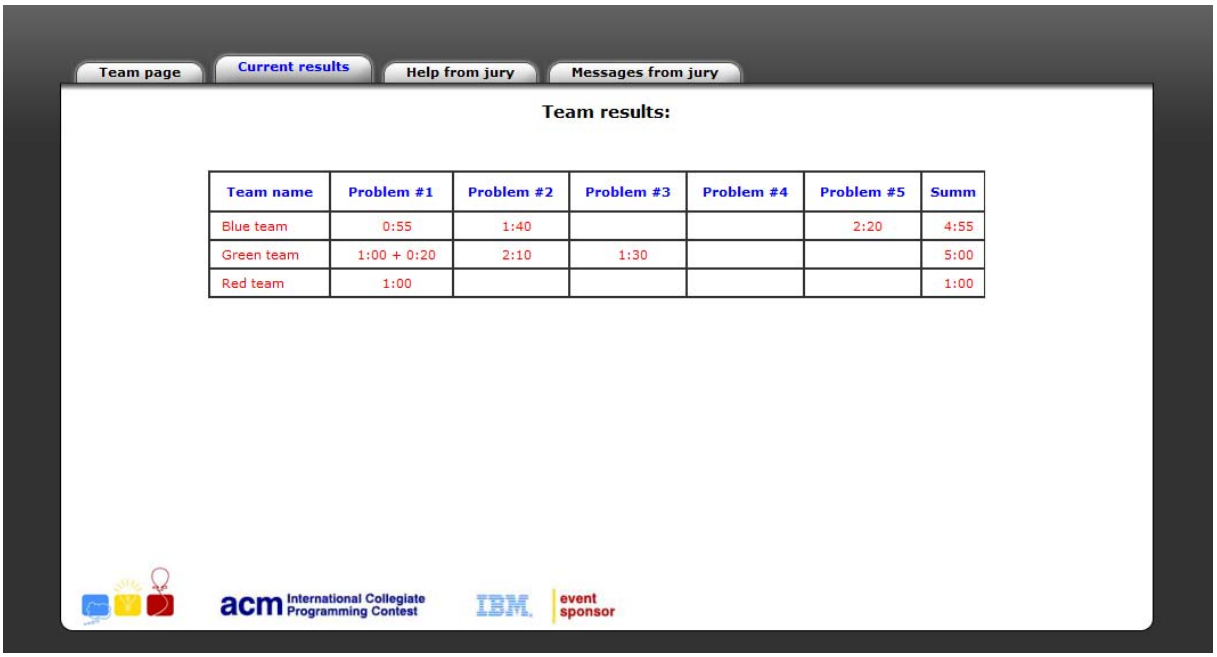
A táblázat nem tartalmaz információkat a beküldés időpontjáról, vagy az időbüntetésekről, de ezek a dolgok itt nem is fontosak. Egy feladat beküldésekor a szoftver elindítja a kód feldolgozását és rögtön visszatér erre az oldalra, hogy a csapat láthassa, miként bírálta el a rendszer az algoritmusukat. Sajnos ez az oldal bizonyos – lentebb megfogalmazott – megfontolások miatt nem töltődik újra automatikusan, ezért a beküldés után manuálisan kell frissíteni ahhoz, hogy megkapjuk az eredményt.

Az oldal nagyobbik részét a forráskódok beküldéséhez használandó űrlap foglalja el. Először is egy legördülő menüből ki kell választani a megoldott probléma sorszámát. Ezután a megoldás előállításához használt programozási nyelvet kell ugyanilyen módon megadni, innen fogja tudni a szoftver, hogy milyen fordítót kell használnia. (Ha itt félrekattint a versenyző, bizonyára meglepve fogja tapasztalni, hogy a fenti

táblázatocskában a probléma neve mellett a „Compile Error” felirat fog díszelegni, holott ő letesztelte a programját, ami nemcsak lefordult, de le is futott.) . Végül a nagy szövegbeviteli mezőbe kell bemásolni a kódot, majd a „Submit” gombra kattintva kész is egy program beküldése.

Ez az űrlap az oka annak, hogy az oldal nincs automatikus frissítésre állítva. Frissítéskor ugyanis az űrlapok már kitöltött mezőinek értéke elvész. Bár ez itt komoly problémákat ritkán okozhatna, hiszen a kódolás nem az oldalon zajlik, vagyis adatot valószínűleg nem veszítenének a felhasználók, azért fontoljuk meg az automatikus frissítés hatásait: az újratöltődést néhány másodperces gyakoriságra állítva a csapatoknak csak ennyi idejük lenne egy program elkészültekkor annak beküldésére. Elég mókás jelenet lenne, amint a versenyzők feszülten figyelik, mikor töltődik újra az oldal, majd villámgyors mozdulatokkal próbálnák kitölteni az űrlapot, hogy még azelőtt a „Submit” gombra kattinthassanak, mielőtt az újbóli frissülés ki nem üríti a mezőket.
Az sem jelentene megoldást, ha két újratöltődés között percek telnének el. Bár egy kód elküldése így kényelmesen megoldható lenne, de utána túl sokat kellene várni a visszajelzésre, ami feszültté tehetné a versenyzőket és felboríthatná a stratégiájukat.
Marad tehát a manuális frissítés.

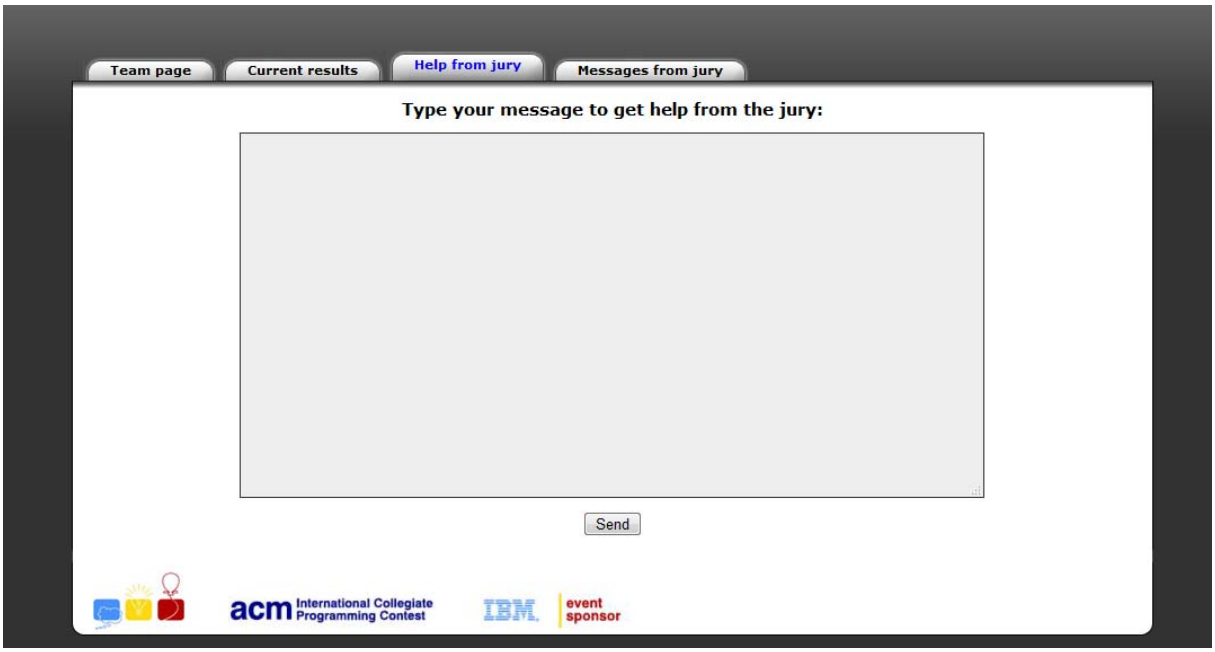
A verseny állása



A csapatok egy pontosan ugyanolyan táblázatban látják a verseny pillanatnyi összesített állását, mint amilyet a zsűri által használt oldalaknál már láthattunk. Tulajdonképpen az egész oldal csupán a fenti lapfülek által takart linkekben különbözik a már bemutatottól. Ugyanolyan logikát követve tartalmazza az beküldések időpontját és esetleges időbüntetéseket. Az automatikus frissítés itt nem okozhat semmilyen problémát, az oldal tehát néhány másodpercenként újratöltődik – egy darabig. Ugyanis a verseny utolsó fél órájában a csapatok már nem láthatják az összesített állást – így rendelkezik a verseny szabályzata. Ekkor tehát az automatikus újratöltődés már nem működik és a manuális frissítést használva sem fognak a versenyzők semmi változást látni a táblázatban.

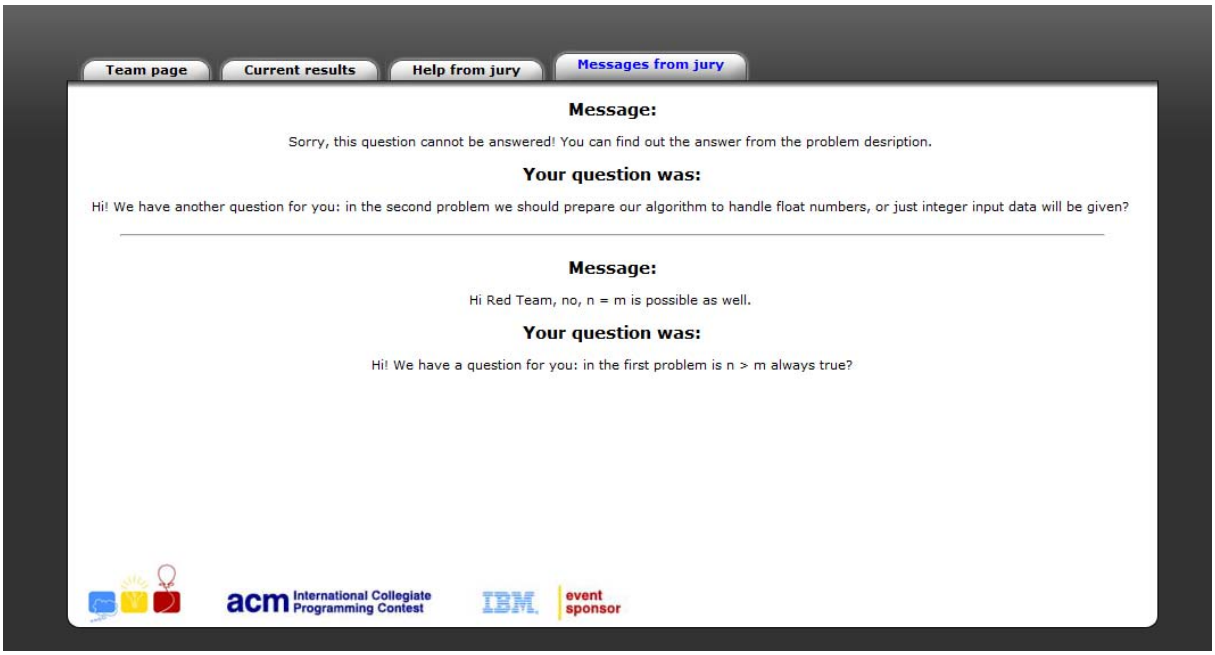
Egyelőre a csapatoknak meg kell keresniük saját magukat a táblázatban, ami 30-40 induló esetén már kicsit kellemetlen. Ezért a programom tökéletesítése során tervezem az aktuális csapat információit tároló sor kiemelését.

Üzenet a zsűrinek



Már többször is esett szó arról, hogy a versenyzők alkalomadtán segítséget kérhetnek a zsűritől. Ehhez a fent látható roppant egyszerű űrlapot használhatják, ami pontosan úgy működik, mint a zsűri „levélküldő” űrlapja. Egyszerűen begépelik az üzenetet, amit a „Send” gombra kattintva tudnak elküldeni.

Válasz a zsűritől



A csapat a zsűrinek küldött kérdésekre kapott reagálásokat a fent látható oldalon olvashatja. Több üzenetváltás esetén az összes megjelenik a lapon vízszintes vonalakkal elválasztva időrendi sorrendben. Nem csak a válasz, hanem a kérdés is olvasható, megkönnyítve a visszakeresést.

A program fejlesztői szemmel

A témavezetőmtől a munka kezdetekor csupán egyetlen szóból álló korlátozást kaptam a program megvalósítását illetően: java.
Ez úton is szeretném megköszönni témavezetőmnek, Kósa Márknak, hogy szabad kezet kaptam és nem lettem korlátok közé szorítva – amit soha nem szerettem.
Először egy szerver-kliens modellen alapuló, csupán a klasszikus J2SE lehetőségeit kihasználó szoftver jutott az eszembe, de ezt szinte azonnal el is vetettem és tekintetem a számomra mindeddig teljesen ismeretlen J2EE felé fordítottam. Olyan eszközöket kerestem, amelyek leveszik a vállamról a hálózati kommunikáció megvalósításának terhét, emellett nem kell klasszikus GUI tervezésével sem fáradnom. Hamar rájöttem, hogy egy HTTP protokollt használó szerveroldali alkalmazás írása lesz a legcélszerűbb, hiszen rengeteg előnye van az első ötletemmel szemben:

- Nincs szükség kliens készítésére, továbbá arról sem kell gondoskodni, hogy a kliens minden versenyzésre használt számítógépen telepítve és megfelelően beállítva legyen, hiszen web-böngésző ma minden gépen megtalálható. Ez a verseny szervezőinek is jócskán megkönnyíti az életét.
  Az is mindegy, hogy a csapatok milyen operációs rendszert futtató gépet használva vesznek részt a versenyben.
- A szerver a klienseket HTTP kérések feldolgozásával szolgálja ki, melynek megvalósítása sokkal magasabb szinten zajlik, mint pl. egy socket alapú kommunikáció implementálása. Ez nem csupán megkönnyíti a fejlesztést, hanem a hibalehetőségek számát is drasztikusan csökkenti.
- Szintén rengeteg munkát és lehetséges hibaforrást jelentett volna a szerver felkészítése arra, hogy egyszerre több klienssel tartsa a kapcsolatot. A J2EE viszont egyszerű API hívásokat biztosít a programozó számára a kliensek és munkamenetek kezelését lehetővé tevő kiforrott, letisztult, régóta használatban lévő technikák használatára.
- A felhasználói felület weboldalakból áll. HTML-CSS párosítással sokkal egyszerűbb tetszetős felhasználói felületet létrehozni, mint klasszikus GUI

elemeket felhasználva. Ezen kívül a kód változtatása nélkül, csupán egy új CSS állomány készítésével néhány óra alatt teljesen át lehet alakítani az alkalmazás megjelenését.

Egyetlen hátránya ennek a megoldásnak az, hogy a versenyzésre használt számítógépeknek csatlakoznia kell az internetre, nincs lehetőség szeparált versenyzési környezet kialakítására, ahogy azt a 2007-es közép európai döntőn láttam Prágában: a gépek nem csak hogy nem voltak csatlakoztatva a világhálóra, de néhány fajta szövegszerkesztőn, a fordítókon és a lebonyolítást végző szoftveren kívül nem lehetett más alkalmazást elindítani.

A használt technológiák

A szoftver írásához a J2EE egyik legfiatalabb és legígéretesebb eszközét, a JSP-t választottam, a továbbiakban ezzel szeretném kicsit megismertetni az Olvasót. Ehhez szükségesnek tartok ejteni pár mondatot a Java szervletekről is. Fontos megjegyeznem, hogy a most következő leírások még a tárgyalt technológiák kisebb komponenseit tekintve sem teljes körűek, csupán arra elegendőek, hogy a dolgozat tárgyát képező szoftver működését az is átlássa, aki soha nem találkozott még az általam használt eszközökkel.

A Java szervletek

A szervletek olyan Java nyelven íródott szerveroldali szoftverkomponensek, amelyek Java programokat futtatni képes HTTP szerverekbe ágyazva szolgáltatásokat nyújthatnak a klienseknek. A szervletek futtatására lehetőséget nyújtó HTTP szervereket szervlet-konténereknek nevezik. A szervletek felhasználási lehetőségei olyan óriásiak és szerteágazóak, hogy egy önálló dolgozat tárgyát is képezhetnék.

Nyelvi szempontból vizsgálva minden szervlet a`javax.servlet.GenericServlet` absztrakt osztályból származik. Ez az osztály olyan nélkülözhetetlen metódusokat tartalmaz, amelyek végigkísérik a szervlet egész életciklusát. Egy szervletet író programozó számára a feladat gyakorlatilag az egyetlen absztrakt metódus, a

`public abstract void service (ServletRequest req, ServletResponse resp)` implementálása az üzleti logika szerint. A szervlet a `req` objektumban fogja megkapni a klienstől érkező kérést és a válaszát a `resp` objektumba fogja beépíteni.

Az szervletek egyik legtipikusabb és jelen dolgozat szempontjából legfontosabb felhasználási területe a HTML űrlapok feldolgozása és HTML oldalak dinamikus előállítása (Pontosan erre volt szükségem, a programom megvalósítását először tisztán szervletek segítségével terveztem, amíg meg nem ismertem a JSP-t.). Ehhez egy speciális osztályt, a `javax.servlet.GenericServlet` leszármazottját, a `javax.servlet.http.HttpServlet`-et kínálja fel számunkra a J2EE.

Egy HTML űrlap adatainak elküldése kétféleképpen is történhet (GET és POST módok), ezeknek a feldolgozására a HttpServlet osztály a `protected void doGet(HttpServletReqest req,HttpServletRespond resp)` és a `protected void doPost(HttpServletReqest req,HttpServletRespond resp)` metódusokat használja, a `GenericServlet`-től örökölt service metódus feladata csupán annyi, hogy a kérés típusától függően továbbítsa a kérést reprezentáló objektumot a megfelelő metódusnak.

A kérés minden adata a `HttpServletReqest` osztály egy objektumában van eltárolva. Az osztály legfontosabb metódusa a `String[] getParameterValues(String name)`, ami a HTTP űrlap `name` nevű beviteli mezőjének az értékeit adja vissza. Ez tehát az alapja az űrlap feldolgozásának. Fontos megemlíteni még a `HttpSession getSession(boolean create)` metódust, ami a munkameneteket és a kliensek megkülönböztetését megvalósító `HttpSession` objektummal tér vissza.

A szervlet által előállított választ egy `HttpServletRespond` objektum reprezentál. Az osztály metódusai lehetőséget kínálnak többek közt a válasz számtalan tulajdonságának beállítására, viszont a legfontosabb köztük a `PrintWriter getWriter()`, ami a válasz saját `PrintWriter` objektumával tér vissza. Ezt az objektumot (a szokásos módon) használva egy olyan pufferbe írhatunk, aminek a tartalma aztán – megfelelő tartalomtípus használata esetén – a kliens számítógép böngészőjében fog megjelenni, vagyis ezzel állítjuk elő a válasz HTML oldalt. Ez azt

jelenti, hogy minél nagyobb oldalt szeretnénk válaszképpen elküldeni, a szervletünk kódjában annál többször szerepel a `PrintWriter` objektum valamelyik író metódusának hívása. Ráadásul mivel a (HTTP) szervlet lényege a dinamikus tartalomelőállítás, egy idő után a kód valósággal hemzsegni fog az előbb említett hívásoktól. Az egész szituációt még bonyolítja, ha a válaszként küldött HTML oldalnak nem csak a tartalma, de a megjelenése is fontos. Tetten értük tehát a szervletek legnagyobb gyengeségét. A J2EE fejlesztőinek ki kellett találniuk egy olyan eszközt, amely ellensúlyozza ezt a gyengeséget és nem csupán kiegészítője lehet a szervleteknek, hanem bizonyos típusú feladat esetén hatékony önálló fejlesztőeszköz is lehet. Így született meg a **J**ava **S**erver **P**ages, röviden a JSP.

Java Server Pages

A JSP oldalak gyakorlatilag pontosan ellentétesen néznek ki, mint a szervletek. Míg egy szervletben a java kód sorai között jelennek meg a kimenetre írandó szövegek, addig egy JSP oldalon a konstans kimeneti szöveg sorai közé ékelődnek be java kódrészletek. Röviden úgy lehetne mondani, hogy a JSP oldal tulajdonképpen egy HTML oldal java kódokkal kiegészítve. (Vagy nem kiegészítve. Ugyanis ha egy tetszőleges HTML állomány kiterjesztését „.jsp”-re változtatjuk, szabályos JSP állományt kapunk). A két eszköznek nem csupán a szerkezete ellentétes, hanem a felhasználási területe is: szervleteket nagy számolásigényű, bonyolult algoritmust megvalósító, sok kódot tartalmazó, viszont szegényes kimenetet produkáló alkalmazás írásakor érdemes használni, JSP-t viszont pont akkor, amikor a megjelenés fontos, az üzleti logika viszont kevés kóddal megvalósítható. Lehetőség van a két eszköz együttes használatára is! Nagyméretű, az MVC architektúrán alapuló alkalmazások felépülhetnek úgy is, hogy az üzleti logikát megvalósító kód oroszlánrésze szervletekben jelenik meg, míg a JSP oldalak feladata leginkább a megjelenítés.
Annak ellenére, hogy ilyen markánsan szembe lehet állítani őket egymással, a szervletek és a JSP oldalak között nagyon is szoros kapcsolat van.
Hogy mennyire, azt jól szemlélteti az, ami egy JSP oldalra történő első hivatkozáskor történik:

- A kliens elküldi a kérést a szervernek

- A web- vagy alkalmazásszerver a kiterjesztés alapján felismeri, hogy JSP oldalról van szó és továbbítja azt a JSP konténernek
- Mivel az oldalra először történik hivatkozás, meghívódik rá a JSP fordító (ami a JSP konténer része)
- A fordító legenerálja egy szervlet osztály kódját
- A JSP-ben szereplő Java kódokat szinte változtatás nélkül bemásolja a generált osztály `jspService` metódusába (ami megfelel a kézzel írt szervletek `service` metódusának)
- A JSP-ben szereplő konstans sorokat átadja a generált szervlet által használt `HttpServletResponse` objektum `PrintWriter` objekuma megfelelő írómetódusának paraméterül
- Egyszerű Java fordítóval lefordítja a szervlet osztályt, majd példányosítja
- Végül lefuttatja a szervletet, ami kiszolgálja a kliens kérését (és ez az utolsó lépés hajtódik végre minden további hivatkozáskor a JSP oldalra)

A JSP oldalakból tehát szervlet készül, vagyis a JSP a szervlet kiterjesztése, absztrakciója. Kifejlesztésének fő célja az volt, hogy segítségével külön lehessen választani a tartalom előállítását a megjelenés megvalósításától és a Java programozásban kevésbé jártasak is hatékonyan kezelhessenek szerveroldali komponenseket.

A JSP elemei

A JSP gyakorlatilag három fajta nyelvi elemmel rendelkezik: direktívák, script-elemek és akcióelemek. Ezek – néhány kivételtől eltekintve – bárhol elhelyezhetők egy oldal kódjában. Ha egy karaktersorozatot a JSP fordító nem ismer fel nyelvi elemként, akkor az változtatás nélkül, a whitespace karaktereket is megőrizve bekerül az előállított oldal szövegébe.
A nyelvi elemek egy része XML-szerű tag, egy része viszont a könnyebb gépelhetőség kedvéért az XML-nél egyszerűbb jelölést használ. A JSP oldalak tehát nem XML dokumentumok, de létezik ekvivalens XML formájuk.

Megjegyzés

<%- Így néz ki egy JSP megjegyzés. A megjegyzés csupán a kód dokumentálására szolgál, nem kerül bele az előállított oldal szövegébe -%>

Direktívák

A direktívák a JSP konténernek szóló utasítások, nem módosítják az előállított oldal szövegét. Alakjuk:

```
<%@ <direktíva-név> <attr1>=”<érték1>” <attr2>=”<érték2>”
... %>
```

A leggyakrabban használt direktíva a `page` direktíva, amivel az egész oldalra vonatkozó jellemzőket állíthatunk be. Az `extends` attribútuma pl. azt mondja meg, hogy a oldalból generált osztály melyik osztályból származzon; az `import` attribútum értékeivel generált szervlet import listáját egészíthetjük ki.

Az `include` direktíva használatával más (nem feltétlenül JSP) fájlt tudunk beilleszteni az oldal kódjába, ezzel nagyméretű oldalakat több részletben, áttekinthetőbben tudunk fejleszteni (ezzel arra is lehetőség van, hogy egy oldal kódolását párhuzamosan több programozó végezze). A beillesztés itt statikus, de később látni fogjuk, hogy van lehetőség dinamikus beillesztésre is.

Scriptelemek

A JSP háromféle scriptelemet ismer: script-részlet, deklaráció és kifejezés. Egy JSP oldalban nem csak a Java használható programozásra, de jelenleg ez a legelterjedtebb, így a továbbiakban én is csak erre fogok koncentrálni.

Script-részletek

Java kódot a <% és a %> karakterpárok közé lehet beilleszteni oldalba. Ahogy már említettem, a nyelvi elemek és a nyomtatandó sorok tetszőlegesen keveredhetnek az oldal szövegében, ez teszi lehetővé pl. a következő kis trükk alkalmazását:

```
<%@ page import="java.util.Calendar" >
<% if (Calendar.getInstance().get(Calendar.AM_PM) ==
Calendar.AM %>
Kellemes délelőttöt!
<% else %>
Kellemes délutánt!
<% %>
```

A JSP fordító a fenti 6 sorból egy majdnem 100 soros szervlet osztályt generál. Az import listát kiegészíti a `java.util.Calendar` osztállyal a page direktíva miatt. A további sorokból pedig a következő Java kódot készíti és helyezi azt el a `jspService` metódusba (az `out` a válasz `PrintWriter` objektumának neve):

```
if (Calendar.getInstance().get(Calendar.AM_PM) ==
Calendar.AM
   out.println(„Kellemes délelőttöt!");
else
   out.println(„Kellemes délutánt!");
```

Ez a hasznos kis trükk csupán a jéghegy csúcsa a JSP előnyeit tekintve.

Deklarációk

A <%! és a %> karaktersorozatok közé helyezhetünk el Java deklarációkat. Mivel változókat a srcipt-részletekben is deklarálhatunk, felmerülhet a kérdés, hogy erre miért is van szükség. A válasz pedig abban rejlik, hogy míg egy script-részletet a fordító teljes egészében a `jspService` metódusba épít be, addig a deklarációra használt scriptelembeli eszközöket a `jspService`-en kívül helyezi el. Ezzel lehetőség nyílik változókon kívül metódusok deklarálására, sőt, a szervlet generált metódusainak felülírására is.

Kifejezések

A kifejezések olyan <%= és %> közé írt Java kifejezések, amelyeket a fordító String típusúra konvertálva közvetlenül beépít a válasz oldal szövegébe.

Implicit objektumok

A programozók munkáját megkönnyítendő a szkript részletekben lehetőség van a generált szervlet által használt objektumok használatára. Így elérhetjük pl. a kérést és a választ reprezentáló `HttpRequest` és `HttpResponse` objektumokat `request` és `response` néven, vagy akár a válasz objektum `PrintWriter`-ét `out` néven. Természetesen az implicit objektumok neve nem definiálható felül.

Akciók

Bár a JSP az eddig bemutatott lehetőségei alapján egy érdekes és apró alkalmazások készítésénél hasznos eszköznek bizonyulhat, egyelőre az Olvasó biztosan nem fedezett fel olyan átütő erejű újítást, ami alapján a JSP-t igazán életképes technológiának tarthatná. Ez nem is csoda, hiszen a JSP igazi erejét a következőkben tárgyalt akciók adják. Az akciók lehetőséget biztosítanak arra, hogy az oldal írója egyszerűsített formában használjon fel bonyolult, nehezen megvalósítható funkciókat. A JSP az akciókkal valósítja meg létrejöttének azon célját, hogy a programozásban kevésbé jártasak is hatékonyan kezelhessenek szerveroldali komponenseket. Először is be szeretnék mutatni néhányat a beépített akció közül.

A <jsp:useBean> akció

Ez a JSP leghasznosabb és legbonyolultabban használható beépített akciója, én is többször használtam fel a szoftver írásához. Alakja a következő:

```
<jsp:useBean
        id="name"
        scope="page | request | session | application"
        {
            class="package.class" |
            type="package.class" |
            class="package.class" type="package.class"
      } />
```

Az `name` egy tetszőleges szabályos azonosító. A `class` és a `type` értéke egy Java osztály lehet, a kettő közül legalább egyet kötelezően meg kell adni. A `scope` fent látható lehetséges értékei lefedik egy szervlet-JSP alapú web-alkalmazás névtereit.
A useBean akció (kissé egyszerűsített) hatásmechanizmusa a következő:

- `id` néven deklarál egy változót. Típusa `type` lesz, amennyiben meg van adva, ellenkező esetben a `class` értékének megfelelő típus
- A `scope` által megadott névtérben megpróbálja megtalálni az id által azonosított objektumot
- Ha megtalálta, értékül adja a deklarált változónak
- Ha nem találta meg és a `class`-ban szereplő osztálynak van paraméter nélküli konstruktora, akkor példányosítja és értékül adja a deklarált változónak
- Ha a fenti lépések nem hajthatóak végre, akkor kivétel váltódik ki

Egészen egyszerűen mondva, ha létezik `id` nevű, `class` (`type`) típusú objektum a névtérben, akkor a rendszer elérhetővé teszi, ha nem létezik, akkor létrehozza.

A <jsp:setProperty> akció

Vagy a scriptelemekben létrehozott, de leginkább a <jsp:useBean> akcióval elérhetővé tett objektumok tulajdonságainak beállítására a <jsp:setProperty> akció használható. Legegyszerűbb alakja:

```
<jsp:setProperty name="objektumnév"
property="tulajdonságnév">
```

Egyszerűen megadjuk az objektum és a tulajdonság nevét, a rendszer pedig a request implicit objektumban tárolt megfelelő névhez kapcsolódód értékre állítja a tulajdonságot. Ehez vagy meghívja az objektum osztályának megfelelő set... metódusát, vagy a `PropertyEditor osztály setAsText(String value)` metódusát. A következő alakot használva kézzel is megadhatjuk a tulajdonság új értékét:

```
<jsp:setProperty name="objektumnév"
property="tulajdonságnév" value="érték">
```

Ebben az esetben a tulajdonság új értéke akár futásidőben is kikalkulálható!

A <jsp:getProperty> akció

Ez az akció a <jsp:setProperty> párja. Egy adott objektum adott nevű tulajdonsága kérdezhető le vele. Használata:

```
<jsp:getProperty name="objektumnév"
property="tulajdonságnév"
```

A fenti három akció működését végiggondolva világossá válik a JSP nagyszerűsége. Hiszen gondoljon bele a kedves Olvasó: a useBean akcióval óriási objektumok válhatnak elérhetővé, a setProperty-vel egyszerűen állíthatjuk be ezen objektumok tulajdonságait közvetlenül űrlapelemből, majd egy metódushívással tetszőleges bonyolultságú algoritmust megvalósító kódot indíthatunk el, aminek a végeredménye, vagy végeredményei lekérdezhetőek a getProperty-vel. És mindezt úgy, hogy a JSP oldal készítője – aki tegyük fel pl., hogy egy webdesigner és a Java programozáshoz csupán alapszinten ért – „bepötyög” pár sort, miközben mit sem tud az üzleti logikát megvalósító kódról, aminek a fejlesztéséért akár programozó csapatok tucatjai is felelősek lehetnek.

A <jsp:forward> akció

Az utolsó beépített akció, amit be szeretnék mutatni, a <jsp:forward>. Alakja:
`<jsp:forward page="erőforrás-url">`
Segítségével a JSP oldal a kérést átirányíthatja egy másik JSP oldalnak, vagy szervletnek. Az előnye ennek nem csak a feladatmegosztás, hanem az hogy az átirányítás dinamikusan, feltételekhez kötve is megvalósítható.

Saját akcióelemek

Ez a JSP legújabb és legígéretesebb területe. A programozó a beépítettekhez hasonló saját akcióelemeket definiálhat, amelyek mögött egy-egy Java osztály áll. Ezek az osztályok az akcióelemek paraméterein keresztül kaphatnak bemeneti adatokat, továbbá elérik a korábban tárgyalt implicit objektumokat. Ezeket felhasználva végeznek el valamilyen feladatot, írhatnak akár a kimeneti pufferbe is, vagy új script-változókat hozhatnak létre. Mivel én ezt az eszközt nem használtam fel a szoftverem készítésekor, nem részletezném a használatát és ezzel a JSP tárgyalását is lezárnám. Annyit szeretnék még ehhez a részhez hozzátenni, hogy a felhasználói oldalak megjelenéséért egy hétköznapi CSS állomány a felelős, amely minden megjelenített JSP oldal elején csatolódik be, a HTML állományoknál megismert módon.

A fejlesztőkörnyezet

A fejlesztéshez a NetBeans IDE 6.0.1 fejlesztőkörnyezetet használtam, ami ingyenesen letölthető a www.netbeans.org címről. Én a lehető legbővebb változatot töltöttem le, ami tartalmaz mindent, ami J2EE alapú webalkalmazások fejlesztéséhez szükséges: rengeteg template áll benne rendelkezésre szervletekhez, JSP oldalakhoz, sőt, még HTML űrlapokhoz is; képes együttműködni web- és alkalmazásszerverekkel, automatikusan végzi el ezek konfigurálását és elindítását. A Netbeans alá 1.6-os Java fordítót telepítettem.

A webszerver

A munkám során a kész szoftverkomponensek kipróbálásához szükségem volt egy szervlet- illetve JSP konténerre, ami nem más, mint egy HTTP szerver, ami tartalmaz Java fordítót, virtuális gépet és JSP fordítót. Én az Apache Software Foundation Tomcat webszerverét használtam, ami a Netbeans-hez hasonlóan szintén ingyenes, ennek ellenére nagyon megbízható, nagy teljesítményű és igen elterjedt. Bővebb információk a szerverről: www.apache.org

Az operációs rendszer

A szoftver fejlesztése Windows Vista Home Premium operációs rendszer alatt zajlott. Bár a Java nyelv egyik legnagyobb előnye a platformfüggetlenség, a programom nem nevezhető annak, mivel több helyen is igénybe veszi az operációs rendszer szolgáltatásait. Mindenképpen tervezek egy olyan telepítőt írni, amely képes lesz valamilyen általános interfészt készíteni a program számára, amin keresztül egységesen lehet operációs rendszer-szintű szolgáltatásokat hívni, miközben a tényleges rendszerhívás a platformtól függően történik.

A dolgozatom utolsó részében a szoftver megvalósításának konkrét részleteire szeretnék kitérni.

A program megvalósítása

A program tehát tisztán JSP felhasználásával készült, amihez kettő darab - J2SE-vel is megvalósítható - Java osztály társul a komplexebb algoritmusok megvalósításáért. Ahol külön nem mutatok rá ezen osztályok használatára, ott a Java kódok a JSP oldal belsejében, szkriptelemekben jelennek meg.

Bejelentkezés, kliensek kezelése

A felhasználás természetesen a zsűri bejelentkezésével indul. A „Log in” gombra való kattintáskor a szoftver először is ellenőrzi, hogy a csapatnév a „jury”, vagy más. Előbbi esetben megnyit egy `jpwd` nevű egyszerű szöveges állományt, amely a JSP fájlok könyvtárában kell, hogy elhelyezkedjen, és amely a zsűri jelszavát tartalmazza. Amennyiben a bejelentkező űrlap `pwd` nevű beviteli mezőjében található karaktersorozat nem egyezik meg a zsűri jelszavával, újra a bejelentkező képernyő jelenik meg, ellenkező esetben az új csapat regisztrálására szolgáló. Ezt a <jsp:forward> akcióval oldottam meg, a korábban bemutatott if-else trükkel párosítva:

```
<% if (!((request.getParameterValues
("pwd"))[0].equals(jpwd)))  %>
    <jsp:forward page="index.jsp" />
<% else %>
    <jsp:forward page="j_newteam.jsp" />
<%  %>
```

Jól látható, hogyan is kell gyakorlatban egy HTML űrlap mezőjének értékét elérni. Mivel egy ilyen mezőnek több értéke is lehet, a `request.getParameterValues` metódus egy tömbbel tér vissza.

A j_newteam.jsp ugyanúgy működik, mint a bejelentkező oldal. Ő a jsp_newteam_reg.jsp oldalnak továbbítja az űrlapjába beírt értékeket. Ez az oldal nem szolgál megjelenítésre, csak JSP kódsorok vannak benne, konstans szöveg nem. A megkapott csapatnevet és jelszót eltárolja egy térképben, majd azonnal visszanavigál a j_newteam.jsp-re. Ahhoz hogy legyen egy térkép, amit használni tud, szükség van a megfelelő osztály importálására

és példányosítására, továbbá a kész térképobjektum elérhetővé tételére. A korábban leírtak alapján ez igen egyszerű:

```
<% @page import=”java.util.HashMap” %>
<jsp:useBean id=”team_pwd” class=”HashMap”
scope=”application”>
```

A `scope` értéke természetesen „application” hiszen a csapatok bejelentkezéséhez szükség lesz erre a térképre – teljesen máshol. Első hivatkozáskor tehát létrejön a térképobjektum, az összes többi esetben csak elérhetővé válik az oldal számára.

Az összes csapat rögzítése után indulhat a verseny. A csapatok bejelentkezését ugyanaz az oldal validálja, mint a zsűriét. Ez az oldal tartalmazza ugyanazt a useBean sort, mint a fenti, vagyis hozzáfér a csapatnév-jelszó párokat tartalmazó térképhez. Egy csapat bejelentkezése sikeres, ha a térképben létezik a csapat nevével megegyező kulcs és a hozzátartozó érték a megadott jelszóval egyenlő. Ha egy csapat bejelentkezése sikeres volt, példányosítódik az egyik általam írt osztály, a Team, majd az új objektum bekerül egy közönséges vektorba. A vektorra több oldal is hivatkozik majd, tehát ezekben mindben szerepelni fog a következő két sor:

```
<% @page import=”java.util.Vector” %>
<jsp:useBean id=”teams” class=”Vector” scope=”application”>
```

Miután a csapatot elhelyezte a rendszer a vektorban, létrehoz egy Session objektumot, ami egyértelműen azonosítani fogja a csapatot, méghozzá a nevük alapján. A bejelentkezéskor megadott csapatnevet a következőképpen helyezzük el a Session objektumban:

```
request.getSession(true).setAttribute("teamId",request.getPa
rameterValues("name"));
```

Ettől a pillanattól fogva a munkamenetnek megfelelő csapatobjektum a következőképpen érhető el:

```
((Team) teams.get(teams.indexOf(new Team(((String[] )
request.getSession().getAttribute("teamId"))))))
```

A Team osztály

A Team osztály először is tehát rendelkezik egy `name` adattaggal, ami alapján egyértelműen azonosítani lehet a Team objektumokat. Az osztály equals metódusa is csupán ezen alapszik és a konstruktor is csak ezt kapja meg paraméterül.
A osztály rendelkezik ezen kívül három tömbbel: `status, time, penalty`. Mindhárom tömb mérete a megoldandó problémák számával egyenlő és rendre a következő információkat tartalmazzák: az „indexedik" probléma státuszát (ahogy azt már tárgyaltuk az „A homepage" fejezetben), a legutóbb beküldött program beküldési idejét, mint Calendar objektumot és az időbüntetéseket, ami egy egész szám.
Az osztály metódusai:

- `public boolean equals(Object o)` : már tárgyaltuk
- `public String getStatus(int index)` : a `status` tömb index-edik elemét adja vissza
- `public Calendar getTime(int index)` : a `time` tömb index-edik elemét adja vissza, amennyiben a `status` tömb `index`-edik elem „accepted", egyébként null-t
- `public int getPenalty(int index)` : a `penalty` tömb index-edik elemét adja vissza, amennyiben a `status` tömb index-edik elem „accepted", egyébként (-1)-et
- `public int compareTo(Object o)` : a klasszikus célú compareTo metódus. Az „accepted" státuszú problémák száma, azon belül pedig az idő + büntetés alapján egyértelmű sorrendet állít fel két csapat között
- `public void setStatus(int index, String status)` : az `index`-edik probléma státuszát `status`-ra állítja, `time` tömb `index`-edik elemét beállítja a verseny óta eltelt időre, végül a `penalty` tömb `index`-edik elemét megnöveli 20-al, amennyiben a `status.equals(„accepted")` nem teljesül. Mindezek után „accepted" státusz rendezi a csapatokat tartalmazó vektort.

Program beküldése, értékelés

A program használata során láttuk, milyen űrlapot kell kitölteni egy program beküldéséhez. A homepage az űrlapot egy compile_run.jsp nevű oldalnak továbbítja feldolgozásra. Mivel a fordítást és a futtatást végző kód JSP viszonylatokat nézve hosszú és bonyolult, elfogadhatatlanul csúnya megoldás lett volna bezsúfolni azt az oldal szövegébe. Ezért fordultam újra a <jsp:useBean> akcióhoz, de ezúttal saját osztály betöltésére használtam.

```
<jsp:useBean id="compile_run" class="Compile_Run"
scope="page" />
```

Látható, hogy a scope értéke page, vagyis a létrehozott objektum az oldal lefutása után meghal. Ez azt jelenti, hogy minden beküldött forráskódot külön új objektum dolgoz fel.

A Compile_Run osztály

Az osztálynak adattagjai: `teamName (String), code (String), status (String), language (String) és problem (int).` Konstruktora üres, ez teszi lehetővé a <jsp:useBean> akció használatát. Metódusai:

- `public void setProblem(int problem)` : beállítja a problem változót. Meghívása a `<jsp:setProperty name="compile_run" property="problem" />` akcióval történik, közvetlenül a feladat beküldésére használt űrlap megfelelő mezőjéből
- `public void setLanguage(String language)` : beállítja a team_name változót. Meghívása a `<jsp:setProperty name="compile_run" property="language" />` akcióval történik, közvetlenül a feladat beküldésére használt űrlap megfelelő mezőjéből
- `public void setCode(String code)` : beállítja a code változót. Meghívása a `<jsp:setProperty name="compile_run" property="code" />` akcióval történik, közvetlenül a feladat beküldésére használt űrlap megfelelő mezőjéből
- `public void setTeamName(String teamName)` : beállítja a teamName változót. Meghívása a

`<jsp:setProperty name="compile_run" property="teamName" value= request.getSession().getAttribute("teamId")/>` akcióval történik

- `public String getStatus()` : ez a metódus végzi a munkát. A `teamName` és a `problem` változók alapján összeállít egy fájlnevet, a kiterjesztés a `language` változó értéke lesz. Az állományba belemásolja a code-ot. A `language` alapján megpróbálja lefordítani, majd lefuttatni az így elkészített forrásfájlt. Ez utóbbi lépéseket a `Runtime` osztály egy példányának segítségével végzi el, amely képes külső programok futtatására az `exec` metódusával. Fordításhoz csupán meghívja a megfelelő fordítót az összeállított fájlnevet átadva paraméterként, futtatáshoz viszont használnia kell az operációs rendszer shell-jét, mégpedig egy ciklus belsejében, hiszen a programnak több száz tesztadatra kell lefutnia.
  Ha a fordítást végző `exec` hibával tér vissza, a visszatérési érték „compile error” lesz. Amennyiben a fordítás és a futtatás is sikeres volt, akkor a program kimeneteit egyenként összehasonlítja azokkal a szöveges állományokkal, amelyek a minta kimeneteket tartalmazzák. Ha valamelyik nem egyezik, akkor a visszatérési érték „wrong output” lesz. Ellenkező esetben még egy utolsó ellenőrzés marad hátra: az első `exec` futása előtt és a második futása után létrejön egy-egy időbélyeg. Ha a kettő különbsége nagyobb, mint egy előre beállított érték, akkor „time limit exceeded” szöveggel tér vissza a metódus. Amennyiben minden ellenőrzés sikeres volt, a visszatérési érték „accepted”.

A compile_run.jsp a fent említett <jsp:setProperty> akciók futtatása után végrehajt egy `<jsp:getProperty name="compile_run" property="status" />` alakút is. Röviden tehát elküldi az objektumnak a szükséges értékeket, majd elindítja az a get-metódust, ami elvégzi a kért feladatot. A visszakapott státusszal meghívja az aktuális csapatobjektum setStatus metódusát, ami nem csupán beállítja a megfelelő adatokat a csapatnak, de ha a beküldött kód megoldás volt, akkor – ahogy már korábban említettem – rendezi a csapatokat tároló vektort, tehát előállítja a pillanatnyi állást.

Az eredmények megjelenítése

Tudjuk, hogy a zsűri és a csapatok egy hasonló táblázatban látják a versenyeredményeket. Ezt a táblázatot a JSP oldal úgy állítja elő, hogy végigmegy a csapatok vektorán és minden csapatobjektumtól lekérdezi az összes problémához tartozó beküldési időt és büntetést. Ha az Olvasó visszalapoz a Team osztály leírásához, látja, hogy a megfelelő get-metódusok csak akkor térnek vissza értékelhető eredménnyel, ha a probléma státusza „accepted”, egyébként a visszatérési értékeke objektum esetén null, egész érték esetén -1. Ezt felhasználva tudja az oldal, hogy mikor kell megjeleníteni időt és büntetést a táblázatban és mikor nem.
Mivel a vektor rendezett, ezért az elejétől a vége felé bejárva és így megszerkesztve a táblázatot tényleg a pillanatnyi sorrendet látják majd a táblázatot tanulmányozó zsűritagok vagy versenyzők a képernyőn.

Kommunikáció a zsűri és a csapatok között

A rendszer a csapatok és a zsűri számára fenntart egy-egy szöveges állományt a kommunikáció tárolására, amelyek neve megegyezik a csapat nevével. Amikor valaki – akár a zsűri, akár valamelyik csapat – elküld egy üzenetet, a szoftver hozzáfűzi azt (és néhány további információt) a címzett állományához. Minden alkalommal, amikor az üzeneteket megjelenítő JSP oldalra hivatkozás történik (akár linken keresztül, akár az automatikus frissítés segítségével), az újra megnyitja az állományt olvasásra, ez biztosítja, hogy minden üzenet a képernyőre kerül.

Összegzés

A kész dolgozatomat utólag átolvasva úgy érzem, sikerült elérnem a célom: az Olvasó nem csupán egy szoftver használatát és működését ismerheti meg, hanem egy mindazokat a körülményeket, amelyek hatással voltak az elkészültére: azt, hogy hol, mire és kik használnak hasonló szoftvereket és azt, hogy teljesen idegen technológiákkal való ismerkedés során hogyan tud körvonalazódni azok felhasználási terve. Ám sajnos – nem titkolom – a szoftver kivitelezése bonyolultabb volt, mint azt először gondoltam. A webes folyamatok – főképp a munkamenetek – még mindig tudnak számomra meglepetést okozni. Különös nehézséget okoz a tesztelés: egyedül nehezen tudok tesztelni egy olyan szoftvert, amit akár 20-30 csapat is használhat majd egyszerre. Végül van még számos olyan apróság, amivel nem vagyok elégedett – néhányat meg is említettem a dolgozatban. Ezen okoknál fogva tehát a program fejlesztése nem ér véget azzal a ponttal, amit a dolgozat utolsó mondatának végére teszek hamarosan, de remélem, egyszer egy tényleg hasznos szoftverrel tudom megajándékozni az ACM verseny helyi szervezőt.

Irodalomjegyzék

- www.acm.org (Az ACM társaság hivatalos honlapja)
- wiki.hup.hu/index.php/ACM (ACM HupWiki)
- www-304.ibm.com/jct09002c/university/students/contests/acm/index.html (Az ACM ICPC hivatalos weboldala)
- en.wikipedia.org/wiki/ACM_International_Collegiate_Programming_Contest (Wikipedia – ACM ICPC)
- J2EE Útikalauz Java programozóknak, ELTE TTK Hallgatói Alapítvány, 2002
- java.sun.com/j2ee/1.4/docs/tutorial/doc/Servlets.html#wp69957 (Sun – szervlet tutorial)
- java.sun.com/j2ee/1.4/docs/tutorial/doc/JSPIntro.html#wp100465 (Sun – JSP tutorial)
- Angster Erzsébet : Objektumorientált tervezés és programozás, Akadémiai Kiadó, 2003.