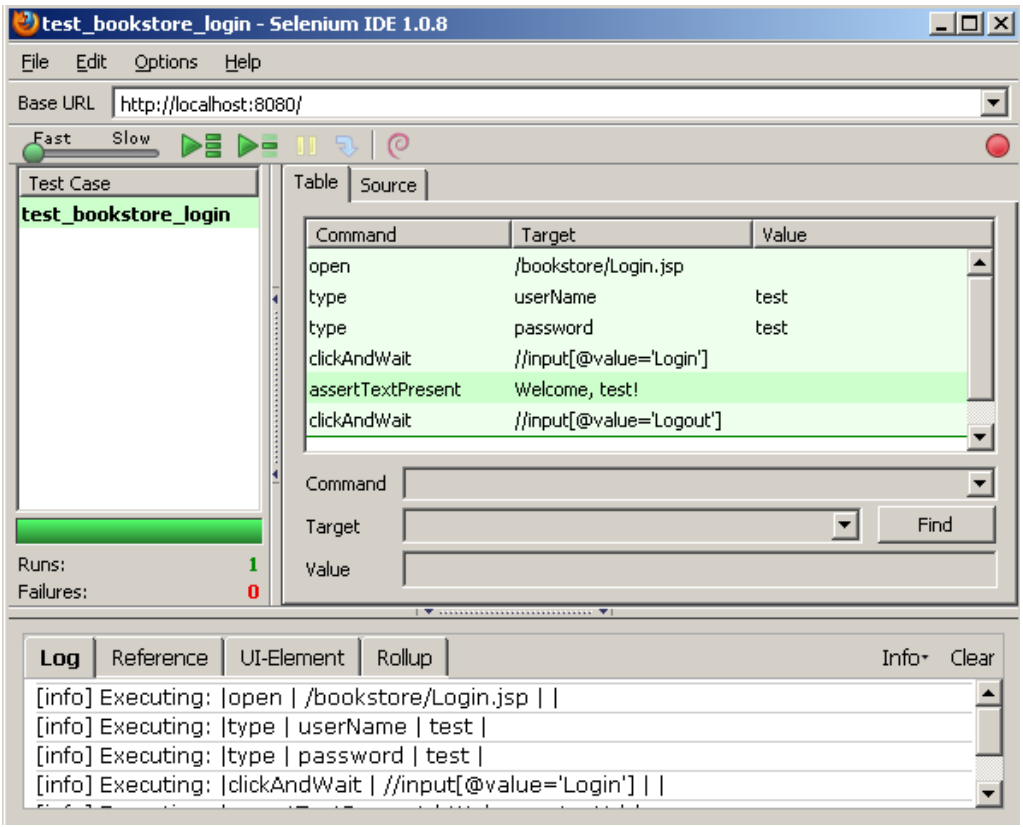
5. gyakorlat: GUI tesztelés és profiling

GUI tesztelés

A gyakorlat első felében webes GUI-k tesztelésére nézünk egy módszert. Egy *record & replay* típusú eszközt, a Seleniumot[1] próbáljuk ki. A tesztek rögzítésére és a helyi gépen való visszajátszására a *Selenium IDE* komponens szolgál, ami egy Firefox add-on.

A Selenium IDE rövid bemutatása

A Selenium IDE a feltelepítés után a *Tools / Selenium IDE* menüpontból érhető el.



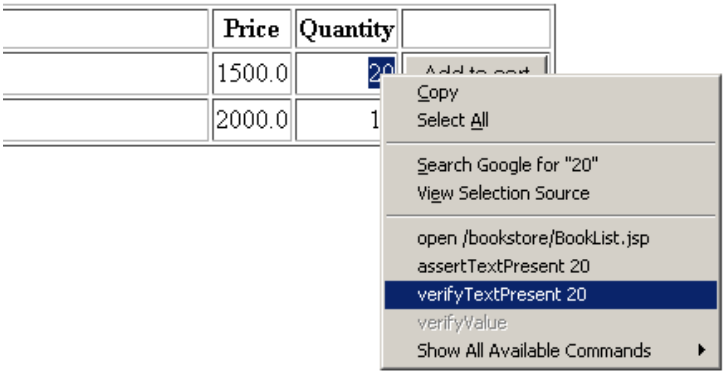
1. ábra: Teszteset a Selenium IDE eszközben

A Selenium IDE elindítása után rögtön felvétel üzemmódba vált, ilyenkor rögzíti a böngészőben lévő kattintásokat és gépeléseket. Ezeket a középső táblázatban látható parancsokban rögzíti. A parancsok szerkezete mindig (parancs, cél, érték).

Több tesztet is lehet rögzíteni, ezeket *tesztkészletekbe* (test suite) lehet rögzíteni. A tesztek utána visszajátszhatóak, az eredmény az alsó *Log* ablakban látszik. Ha valamelyik teszt sikertelen, akkor érdemes a visszajátszás sebességét lejjebb venni, és akkor követhető a böngésző ablakban a végrehajtás is.

[1] http://seleniumhq.org/

Az ábrán szereplő tesztesetben látszanak a leggyakrabban használt parancsok: *open*, *type*, *clickAndWait*, valamint a szöveg ellenőrzésére szolgáló *assertTextPresent*. A további parancsok használatához érdemes körülnézni a Selenium által felkínált menüpontokban, amik a weboldal vizsgálni kívánt elemén jobb gombbal kattintva jelennek meg.



2. ábra: További Selenium parancsok

A Selenium IDE-ben a *Command* melletti legördülő menüben látható, hogy a Seleniumban rengeteg további beépített parancs van. Egy parancs kiválasztása esetén alul a *Reference* részen olvasható annak a leírása.

Feladatok

1. Egy webes könyváruházat fogunk vizsgálni a gyakorlat során, ezt indítsuk el az Asztalon lévő parancsikonnal. A weboldal ezután a http://localhost:8080/bookstore/Login.jsp oldalon érhető el.
2. Rögzítsük a fenti ábrán is látható sikeres bejelentkezést vizsgáló tesztesetet.
3. Rögzítsük egy új tesztesetet, mely egy sikertelen bejelentkezést vizsgál.
4. Vegyünk fel egy olyan tesztesetet, mely a kosárba helyezést ellenőrzi. Ha számításokat szeretnénk végezni, akkor ahhoz a *storeEval* parancsot használhatjuk, ami egy JavaScript részlet eredményét menti el.

Profiling

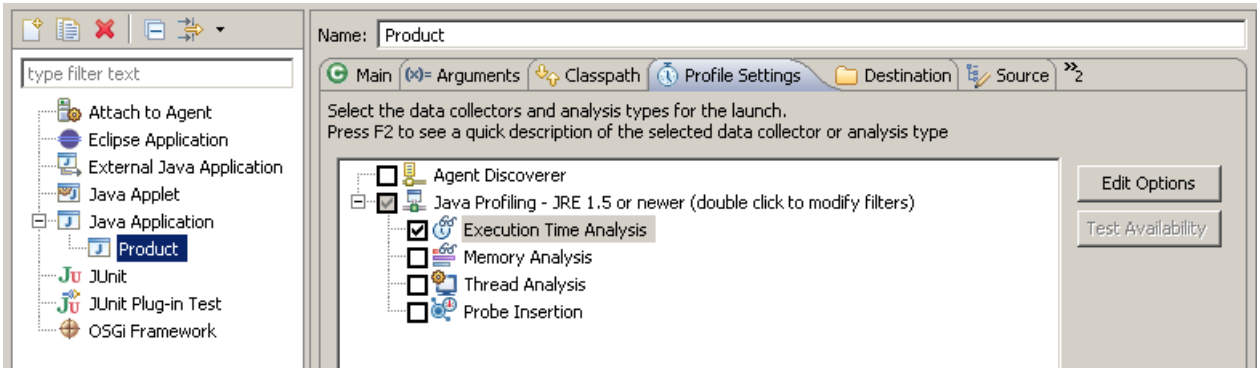
A gyakorlat második felében az *Eclipse TPTP*[2] profilert fogjuk használni végrehajtási idők és memóriafoglalások vizsgálatára.

TPTP bemutatása

A TPTP feltelepítése után megjelennek a *Profile* futtatási lehetőségek.

3. ábra: Futtatás profiler segítségével

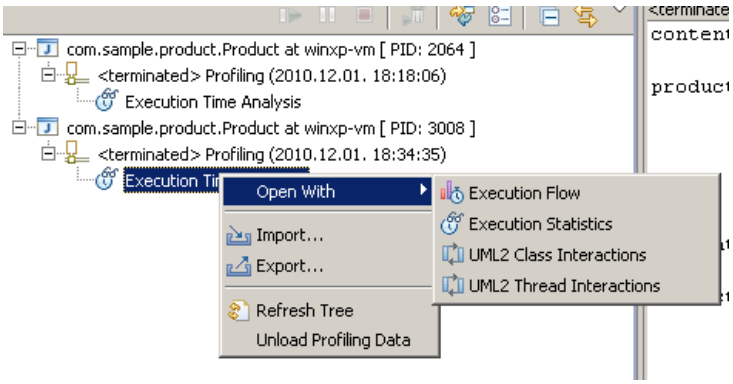
A megfelelő *Run configuration* beállításainál meg kell adni, hogy milyen információkat szeretnénk rögzíteni.



4. ábra: Nyomkövetési lehetőségek

A *Java Profiling* elemet kiválasztva az *Edit Options* résznél tudjuk megadni hogy milyen csomagokat szűrjön ki vagy rögzítsen a profiler.

Az alkalmazás elindítása után átvált a *Profiling and Logging* perspektívába.



5. ábra: Rögzített adatok megjelenítési lehetőségei

[2] http://www.eclipse.org/tptp/

Itt utána látszanak a korábbi és az aktuális futtatások, a rögzített adatok (5. ábra). A jobb gombos menüben adhatjuk meg, hogy milyen formában szeretnénk az adatokat megjeleníteni.

Végrehajtási idő rögzítése esetén ezek a következőek:

- *Execution Flow*: egy viszonylag nehezen értelmezhető grafikus forma.
- *Execution Statistics*: táblázatos forma az átlagos és összesített végrehajtási időkről. A csomagok és osztályok mentén összesíthetjük az eredményeket.
- *UML2 Class és Thread Interactions*: UML szekvencia diagramok formájában jeleníti meg a kommunikációt az objektumok és/vagy szálak között.

Az *Execution Statistics* nézetben a fontosabb oszlopok jelentése a következő:

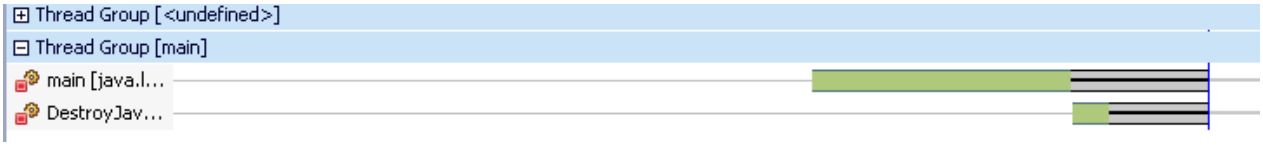
- *Base Time*: az adott típusú hívás kiszolgálására fordított összes idő, a hívásból indított további hívások ideje nélkül.
- *Avarage Base Time*: `Base Time / Calls`
- *Cumulative Time*: az adott hívásban és a belőle induló hívásokban töltött összes idő. Ha egy hívásban nem tartalmaz további hívásokat, akkor a *Base Time* értékkel egyezik meg.
- *Calls*: az adott típusú hívás hányszor került végrehajtásra.

Feladatok

A TPTP egyik mintaalkalmazását fogjuk használni, mely XML fájlokból olvassa fel termékek adatait, majd azokat megjeleníti a képernyőn.

1. Állítsuk be a profiling beállításainál, hogy végrehajtási időket mérjen.
2. Futtassuk az alkalmazást, majd elemezzük a kapott eredményeket!
    a. Melyik osztályban töltöttük el a legtöbb időt?
    b. A *Call Tree* nézet alapján melyik metódusban töltöttük el a legtöbb időt?
    c. Hogyan lehetne javítani ennek a metódusnak a hatékonyságán?
3. Nézzük meg az *UML2 Class Interactions* nézetet is.
    a. Rejtsük el a SecuritySupport példányokat, hogy kicsit áttekinthetőbb legyen a kép.
    b. A hívási hierarchia így nagyjából követhető, és át lehet tekinteni az alkalmazás működését.
    c. A baloldalon található rózsaszín csík jelzi a végrehajtási időket, minél sötétebb egy adott terület, annál hosszabb ideig tartott végrehajtani. Vannak-e úgynevezett hotspotok az alkalmazásban (ahol a végrehajtási idő nagy része tömörül)?
    d. Az elrejtett osztályokat az ablak jobb felső sarkában lévő lefelé nyilacska (View Menu) *Hide/Display Patterns* menüjében tudjuk visszarakni.

4. Állítsuk át, hogy memória adatokat rögzítsen a profiler.
    a. Melyik osztályból keletkezett a legtöbb példány?
    b. Részletesebb elemezésekre használható a különálló Memory Analyzer (MAT)[3] eszköz például.
5. Többszálú alkalmazások vizsgálatához lenne hasznos a Thread Analysis.



[3] http://www.eclipse.org/mat/