5. gyakorlat: Struktúra alapú tesztelés

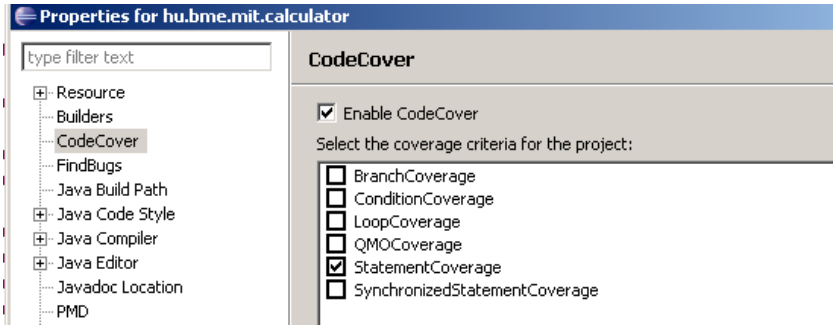
Kódfedés mérése

Az első feladat során egy meglévő tesztkészlet kódfedését fogjuk megvizsgálni különböző fedési kritériumok alapján, majd az azonosított hiányosságok alapján új teszteket implementálunk (*hu.bme.mit.calculator* projekt az SVN repositoryban: https://szet.inf.mit.bme.hu/svn/main_2011/trunk/gyak4/).

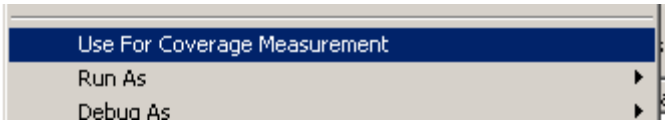
A kódfedés méréséhez a ***CodeCover*** eszközt használjuk, mely a következő URL-ről telepíthető az Eclipse-en belül: http://update.codecover.org/ (a kiadott virtuális gépen ezt már elvégeztük).

1. Váltsunk át a CodeCover perspektívára, és állítsuk be a CodeCovert!

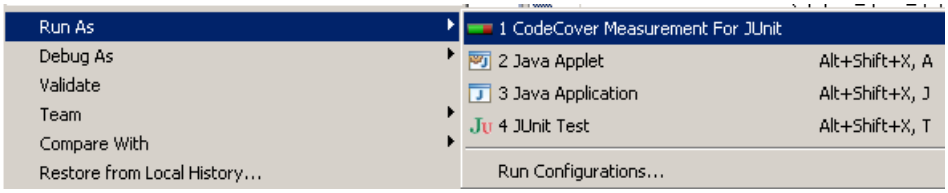
A CodeCover használatához először be kell kapcsolni a projekten, hogy milyen típusú fedést akarunk mérni:



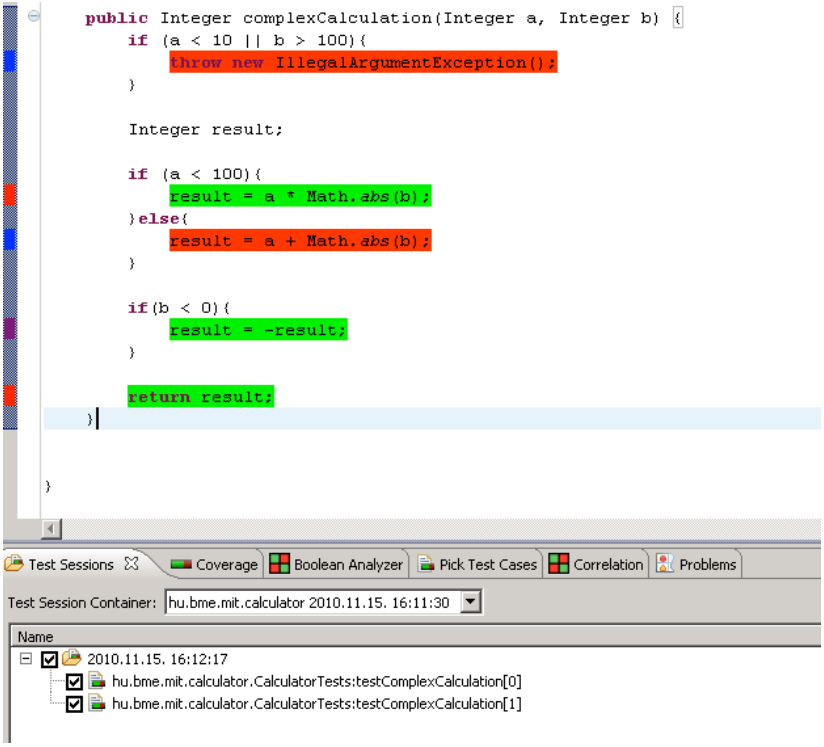
Ezután ki kell választani, hogy melyik forrásfájlokat akarjuk instrumentáltatni (jobb gomb az adott forrásfájlon, majd *Use For Coverage Measurement*):



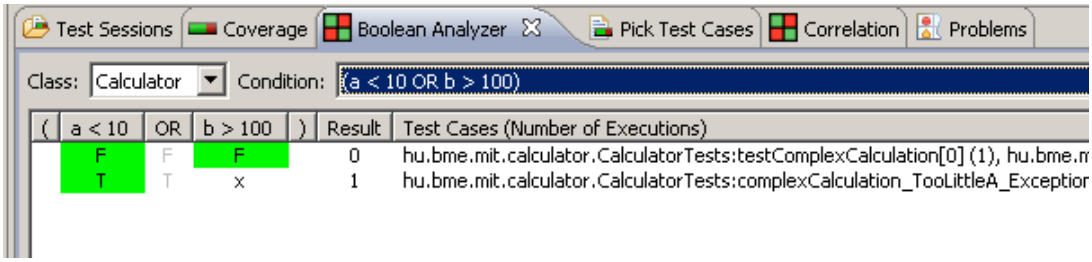
Majd a *CodeCover Measurement For JUnit* opcióval kell a teszteket indítani.



A tesztek lefutása után alul a *Test Sessions* nézetben ki kell választani, hogy melyik futtatások eredményét akarjuk megnézni. Ha kiválasztjuk valamelyiket, akkor a forráskódot megfelelően színezni fogja az eszköz.



2. Bővítsük a tesztkészletünket, hogy a complexCalculation metódusnál 100%-os utasítást lefedést érjünk el.
3. Állítsuk át, hogy csak döntés lefedettséget vizsgáljon az eszköz (BranchCoverage), majd így is egészítsük ki a tesztkészletet.
4. Végül állítsuk át, hogy csak feltétel lefedettséget vizsgáljon (ConditionCoverage), és így is egészítsük ki a tesztkészletet. A lefedetlen kombinációk azonosításához segítséget nyújt a *Boolean Analyzer* nézet.



(Figyelem! A CodeCover a következőt érti condition coverage alatt: *„Term coverage takes a closer look to the decision expression in if-statements. Each basic boolean term of the decision must at least once effect the overall result to true and to false. CodeCover implements the Ludewig term coverage, which subsumes MC/DC for boolean short circuit semantics.”*)

Modul izolációs tesztelés

Unit tesztelés esetén az egyik legfontosabb feladat annak megoldása, hogy a modul tényleg izoláltan fusson, és az összes függőségét valahogy helyettesítsük. Az irányítható és megfigyelhető csonkok kézi elkészítése helyett most a *Mockito*[1] nevű keretrendszert fogjuk használni ezek automatikus generálására.

Mockito bemutatása

A *Mockito* mind stubok és mock objektumok használatát lehetővé teszi.

- Állapot ellenőrzése:

```
// create a test double (there is no separate stub() call)
List mockedList = mock(List.class);

// specify how the test double should respond to calls from the SUT
when( mockedList.get(0) ).thenReturn( "first" );

// call the SUT as in any test
String r = sut.queryList(mockedList, 0);

// assert the state or the return value of the SUT to decide the outcome of the test
assertEquals("first", r);
```

- Interakciók ellenőrzése:

```
// create a test double
// it automatically records all the calls received
List mockedList = mock(List.class);
sut.setList(mockedList);

// call the SUT as in any test
sut.add(0);

// verify the mock and not the SUT
// check that the SUT sent the required interactions
verify(mockedList).add(0);
```

A fenti kód lefordulásához még importálni kell a Mockito statikus metódusait:

```
import static org.mockito.Mockito.*;
```

A Mockito használata során a kulcslépés egy általános *test double* létrehozása a mock hívással (pl. List mockedList = mock(List.class)). Lehet interfészt vagy egy konkrét osztályt is „mockolni”. Ennek eredményeképp:

- a mockedList-en hívható a List összes metódusa,
- minden külön nem definiált hívásra valami alapértéket ad vissza, pl. null, 0, üres lista,
- a mockedList-nek küldött hívásokat eltárolja, később azok ellenőrizhetőek.

A fenti alapokon kívül még a következő funkciók lehetnek hasznosak:

[1] http://mockito.org/

- Argument matcher: ha nem equals egyezést akarunk a paramétereknél (lásd a Matchers osztály metódusait)

```
when(mockedList.get(anyInt())).thenReturn("element");
```

- ArgumentCaptor: ellenőrzés során lehetséges a mock-nak átadott paraméter elkapása
- Adott számú meghívás ellenőrzése (atLeast(2), atMost(5), never() is használható)

```
verify(mockedList, times(2)).add("twice");
```

  - Kivétel dobása

```
when(mockedList.get(-1)).thenThrow(new Exception());
```

  - when szabályok felüldefiniálhatják egymást (legutolsó számít)

További leírást lásd a Mockito dokumentációjában:

http://docs.mockito.googlecode.com/hg/org/mockito/Mockito.html

Feladatok

Adott egy űrhajók működését modellező kódváz (hu.bme.mit.spaceship projekt az SVN repositoryban: https://szet.inf.mit.bme.hu/svn/main_2011/trunk/gyak4/). A feladat egy konkrét hajó torpedókilövő rutinjának a modul szintű tesztelése. A metódus definíciója a következő:

```
/**
* Tries to fire the torpedo stores of the ship.
*
* @param firingMode how many torpedo bays to fire
* SINGLE: fires only one of the bays.
*  - For the first time the primary store is fired.
*  - To give some cooling time to the torpedo stores,
     torpedo stores are fired alternating.
*  - But if the store next in line is empty the ship
     tries to fire the other store.
*  - If the fired store reports a failure, the ship
     does not try to fire the other one.
* ALL: tries to fire both of the torpedo stores.
*
* @return whether at least one torpedo was fired successfully
*/
public boolean fireTorpedos(FiringMode firingMode)
```

A unit teszteléshez az adott modult (GT4500 osztály) izoláltan kell tesztelni:

- Azonosítsuk, hogy milyen függőségei vannak a modulnak.
- Tudjuk-e minden függőségét befolyásolni a tesztek során, könnyen vezérelhető-e a tesztelendő modul? Ha nem, milyen megoldást javaslunk?
- Készítsünk a modul leírása alapján unit teszteket a fireTorpedos metódushoz, a függőségek helyettesítéséhez használjuk a Mockito keretrendszert.
- A tesztekben próbáljuk végiggondolni, hogy az állapot vagy interakciók ellenőrzése a célszerűbb, és használjuk a megfelelő módszert.