8. nädala kodutöö

IDK0051, sügis 2017
Tähtaeg: teisipäeva varahommikul kl 02:00

Kodutöö eesmärgiks on realiseerida programm, mis kasutab *dependency injection* printsiipi. Sõltvuse sisestamise eesmärgiks on viia kontroll objekti loomise ja seadistamise üle objektist välja (*inversion of control*).

Lihtsaim näide on andmete lugemine objektist. Selle asemel, et objekt looks ise ühenduse andmebaasiga, failiga või veebiteenusega, annab objekti kasutaja talle andmebaasiühenduse ette. Näiteks Student objekti meetod save() kasutab sisestatud DbConnection-tüüpi objekti, või lausa üldist Output-tüüpi objekti.

Kindlasti märkate, et tehniliselt kasutab sõltuvuse sisestamine nii kapseldamise kui kompositsiooni printsiipe sõltuvuse salvestamiseks klassis – üks ei välista teist! Sõltuvuse sisestamise idee ei ole meie aines erakordselt uudne programmikood, vaid uudne loogika – kontrolli viimine objektist välja (*inversion of control*).

Ülesanne

Bioloogid käivad tihti ekspeditsioonidel sellistes paikades maailmas, kus on kasutada vaid väga madala läbilaskevõimega andmeside. Seetõttu on nad kasutusele võtnud andmete pakkimise, et piiratud kanalit kasutades ikkagi võimalikult palju andmeid Ülikoolis asuvasse teaduskeskusesse saata. Teie ülesandeks on luua programm, mis töötab bioloogide kaasaskantavas sidejaamas ja saadab tekstilisi andmeid teaduskeskusesse neid eelnevalt pakkides. Sõltuvuseks, mis sinna programmi sisestatakse, on pakkija objekt, mis reaalselt sõnumi ära pakib.

1. Realiseerige **TDD printsiibil** sõnumi pakkija (sõnumi suuruse vähendaja), mis:
    - eemaldab sõnumist kõik täishäälikud ja tagastab sõnumi. Nt sõnumist „Avastasime uue elevandiliigi, isendid on kollased ja oskavad laulda” jääb peale pakkimist järgi „vstsm lvndlg, sndd n kllsd j skvd lld”

2. Looge andmesideühenduse liides, mille avalikus API-s kaks avalikku meetodit:
    - sõnumi lisamiseks – seda meetodit võib enne saatmist mitu korda välja kutsuda, sel juhul lisatakse uus sõnum eelmisele, eraldades kaks sõnumit semikooloniga
    - sõnumi saatmiseks – pakib sõnumi ja saadab teaduskeskusesse.
        - Kuna meil vastuvõtvat teaduskeskust ei ole, siis realiseerige see nii, et sõnum kirjutatakse faili

3. Kui süsteemi käivitate, siis sõnumi pakkija sisestage sõltuvusena andmesideühenduse objekti.

Sõltuvuse kasutamise testimine

4. Kirjutage ühik-test selle kohta, kas andmesideühenduse klass kutsub pakkijat õigete argumentidega välja. Kasutage mock'imist. Teie eesmärk on testida, **kas andmesideühenduse klass saadab pakkijale pakkimiseks õige sõnumi** kui sõnumi saatmise meetodi välja kutsute. Kujutage ette näiteks sellist kasutusjuhtu:

- te lisate andmesideühenduse objekil sõnumi „Nägin ahvi!”
- te lisate andmesideühenduse objekil sõnumi „Moskiito hammustas kolleegi!”
- kutsudes nüüd välja sõnumi saatmise, eeldate, et pakkimiseks saadetakse tekst „Nägin ahvi!;Moskiito hammustas kolleegi!”

Kasutame Mockito raamistikku, kus on olemas mock() ja verify() meetodid. Idee on selles, et mock() abil saate luua objekti, mis käitub nagu argumendiks olev klass, kogu API on sama, kuid reaalselt need meetodi ei tee midagi. Näite:

Student student = mock(Student**.class**);
student.getName(); // meetodit getName() ei kutsuta välja päris Student-tüüpi objekti peal (seda meil siin koodinäites ei olegi), vaid mock() poolt loodud nö petuobjektil.

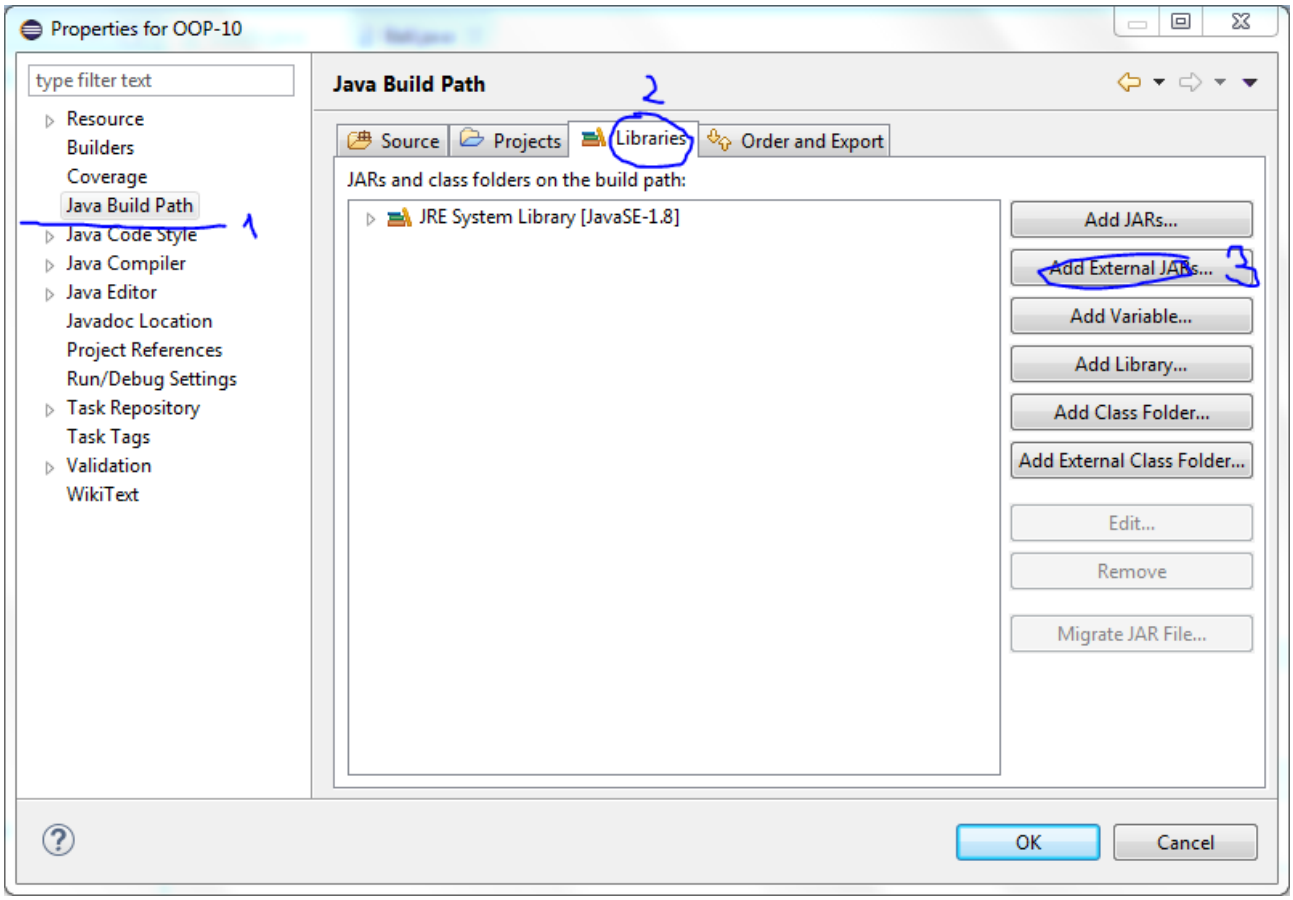
Mockimiseks tuleb teil ühiktestis luua pakkijast *mock*-objekt (kasutades meetodit mock()) ja sisestada see mock-objekt andmesideühenduse klassi päris objekti asemel ja seejärel lisada ja saata sõnumid. Nüüd saate *verify*() meetodiga mock objektil kontrollida, kas saadeti õige sõnum.

Ülesande sisu lõpp. Edasine siin failis on täiendav info.

Kuidas seadistada?

1. Esmalt laadige mokkimiseks alla Mockito raamistiku JAR: http://mockito.org/ (JAR olemas ka ained.ttu.ee)

Lisage see JAR Eclipses/IntelliJ oma projekti libraryte hulka (Project - Properties):



(IDE-d pakuvad võimalust Mockito ka automaatselt projekti lisada – palun googeldage).

2. Mockito omakorda kasutab teisi Java programme (nende public meetodeid), mistõttu tuleb projekti lisada ka sõltuvused bytebuddy, bytebuddy-agent ja objenesis (JAR-id ained.ttu.ee)

3. Nüüd saate oma testklassis staatiliselt importida Mockito meetodid, seda saab teha kahe reaga:

```
import static org.mockito.Mockito.mock;
import static org.mockito.Mockito.verify;
```

Kuidas mokkida?

1. Uurige näidet nr 1 http://site.mockito.org/mockito/docs/current/org/mockito/Mockito.html#1 Mockito dokumentatsioonist
2. Erinevad kasutusjuhud: http://www.baeldung.com/mockito-verify

mock() abil loodud objekti saate testis kasutada päris objekti asemel. Hiljem saate verify() meetodi abil kontrollida, kas mock-objektil kutsuti välja neid meetodeid, mida eeldasite.

Näiteks kui teil on mock objekt nimega *mockService* ja te tahate kontrollida, kas testi jooksul kutsuti välja mockService peale meetodit setMessage() argumendiga „tere”, siis vastav verify() väljakutse on:

```
verify(mockService).setMessage(„tere”);
```

Kodutöö hindamine

Jälgige, et (lisaks nõutud funktsionaalsusele) teie kood vastaks *clean code* nõuetele:

1) klassinimed suure algustähega, näiteks: Printer või MatrixPrinter

2) meetodite/muutujate nimed camelCase, näide: printMyName()

3) klassi/meetodite/muutujate nimed tähenduslikud. Hea näide: removeLastItemFromQueue()

Halb näide: strx2()

4) Taanete osas hästi vormindatud. Soovitav koodiformaatoriga üle kontrollida

Eclipe: Ctrl + Shift + F
IntelliJ Ctrl + Alt + L

5) testid ühe konkreetse funktsionaalsuse kohta, mitte ühes testis kogu funktsionaalsus