8. nädala kodutöö

IDK0051, sügis 2015
Tähtaeg: 23:59:59 enne teie 10. nädala praktikumi

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

Kui oma seadistusi kontrolliv objekt peab ise otsustama, kust andmeid lugeda, kuhu andmeid kirjutada või milliseid alammooduleid kasutada, siis sõltuvuse sisestamise korral omab kontrolli objekti kasutaja.

Lihtsaim näide on andmete lugemine objektist. Selle asemel, et objekt looks ise ühenduse andmebaasiga, failiga või veebiteenusega, annab objekti kasutaja talle selle ühenduse ette.

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 pakkimise 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 klass, mille avalikus API-s on konstruktor ja 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:

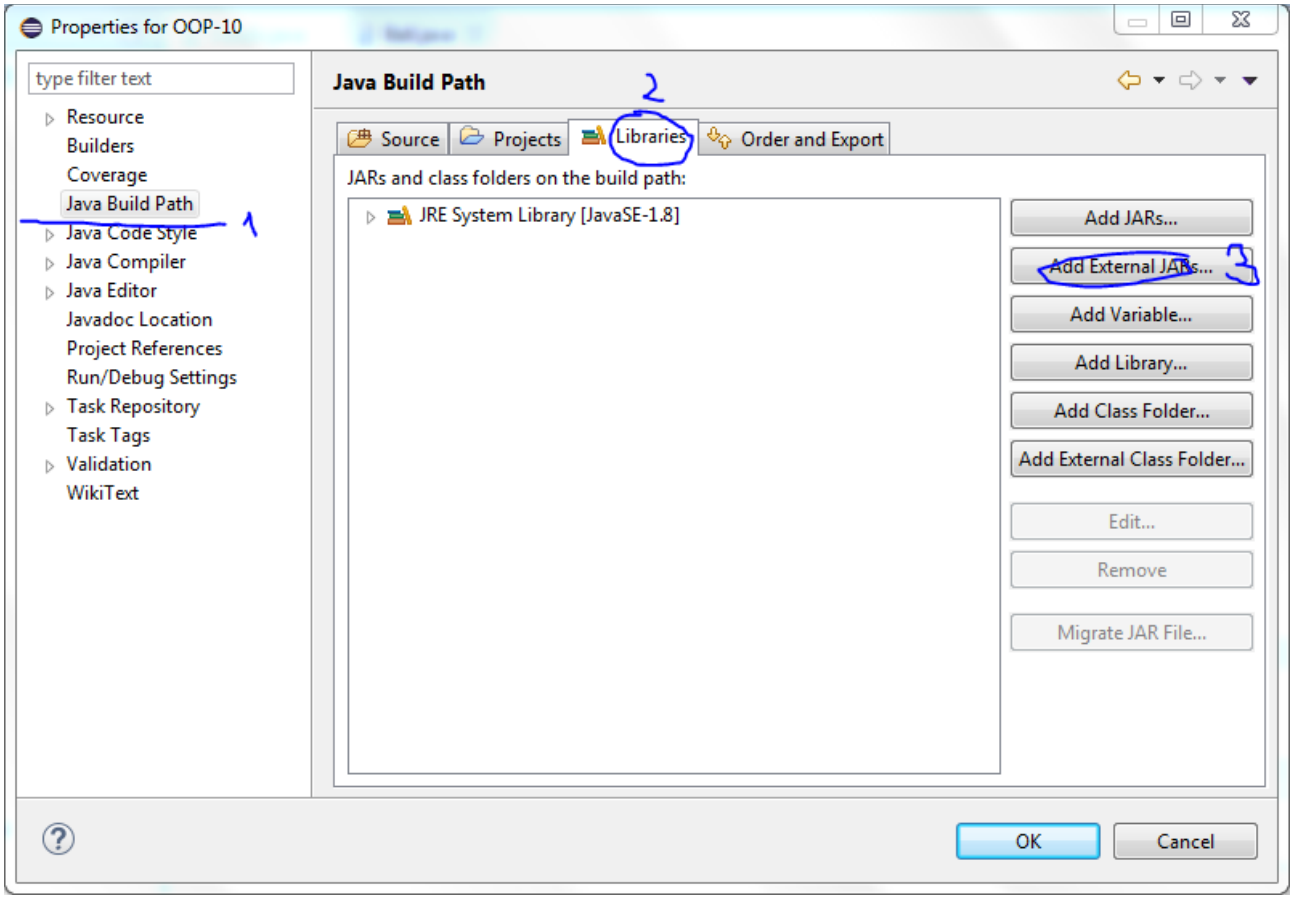
- te lisate andmesideühenduse objekil sõnumi „Nägin ahvi!”
- te lisate andmesideühenduse objekil sõnumi „Moskiito hammustas kolleegi!”
- kutsudes nüüd välja sõnumi saatmise, eeldate, et pakkimiseks saadetakse tekst „Nägin ahvi!;Moskiito hammustas kolleegi!”

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

Kuidas seadistada?

1. Esmalt laadige mokkimiseks alla Mockito raamistiku JAR: http://mockito.org/

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



2. 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. Vaadake ained.ttu.ee keskkonnas 8. praktikumi näidislahenduse testklassi mock ja verify kasutamise kohta
2. Uurige näidet http://docs.mockito.googlecode.com/hg/org/mockito/Mockito.html#1 Mockito dokumentatsioonist

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

Punktide jaotumine

Kokku saab kodutöö eest 3 punkti, mis jaguneb nii:

- Ülesanded 1 – 3 (kood ja pakkija unit-testid): kokku 1 punkt
- Ülesanne 4 (mockimine): 2 punkti