Testimise mustrid ja võtted

```
public void transfer(Money money,
                     String fromAccount, String toAccount) {

    String fromCurrency = bankService.getAccountCurrency(fromAccount);
    Money creditAmountConverted = bankService.convert(money, fromCurrency);

    if (!bankService.hasSufficientFundsFor(creditAmountConverted))
        return;

    bankService.credit(creditAmountConverted, fromAccount);
```

Child Test

```
@Test
public void calculatorSupportsAddition() {
    RpnCalculator c = new RpnCalculator();
    c.setAccumulator(1);
    c.enter();
    c.setAccumulator(2);
    c.plus();
    assertThat(c.getAccumulator(), is(3));
}

public void calculatorSupportsEnter() {
    RpnCalculator c = new RpnCalculator();
    c.setAccumulator(1);
    c.enter();
    assertThat(c.stack.get(0), is(1));
    assertThat(c.getAccumulator(), is(1));
}
```

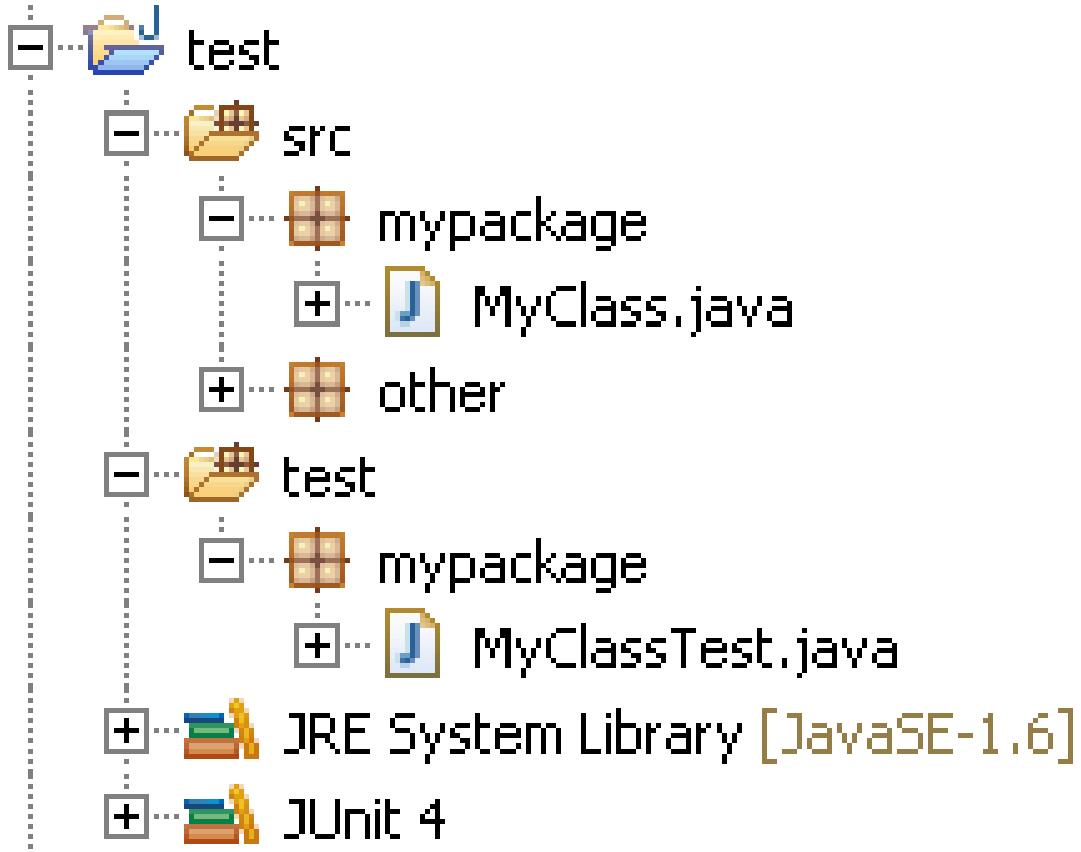
Learning Test

```
@Test
public void learingScanner() {
    Scanner s = new Scanner("abc 123");

    assertTrue(s.hasNext());
    assertFalse(s.hasNextInt());

    assertThat(s.next(), is("abc"));

    assertTrue(s.hasNextInt());

    assertThat(s.nextInt(), is(123));

    s.close();
}
```

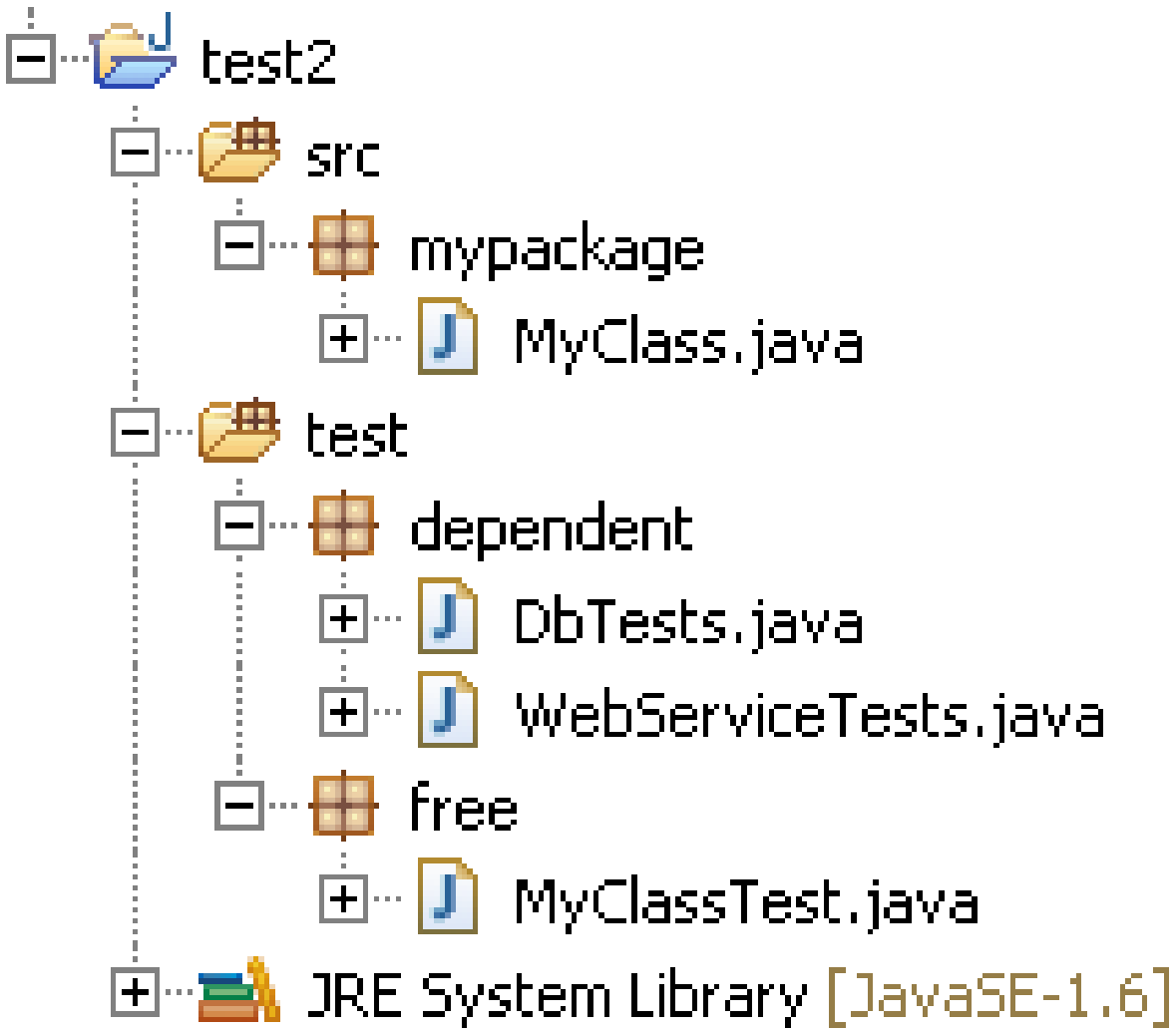
Test List

- $5 + 10 CHF = $10 if rate is 2:1
- ~~$5 * 2 = $10~~
- Make “amount” private
- Dollar side-effects?
- Money rounding?

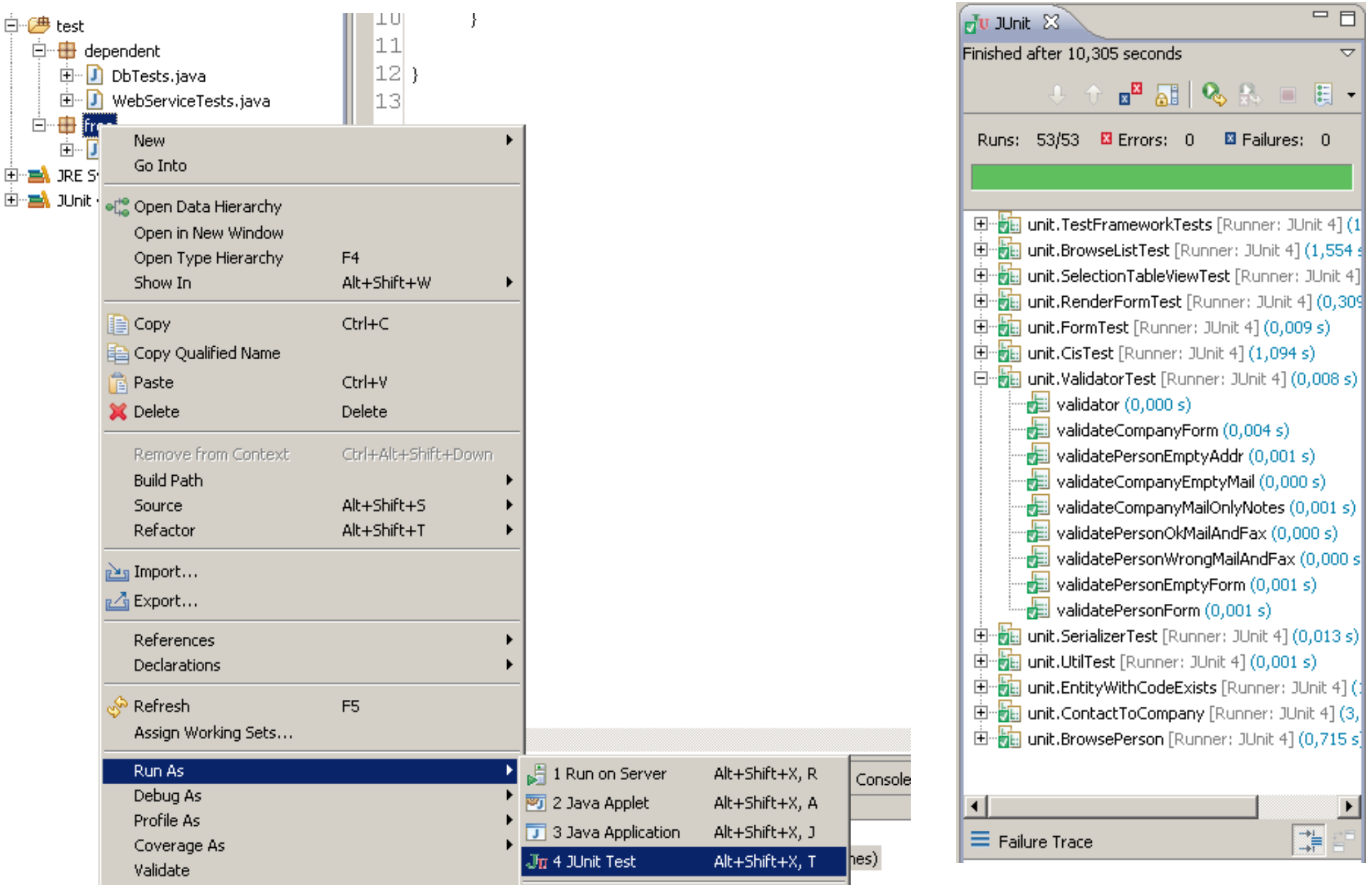
Testide paigutus



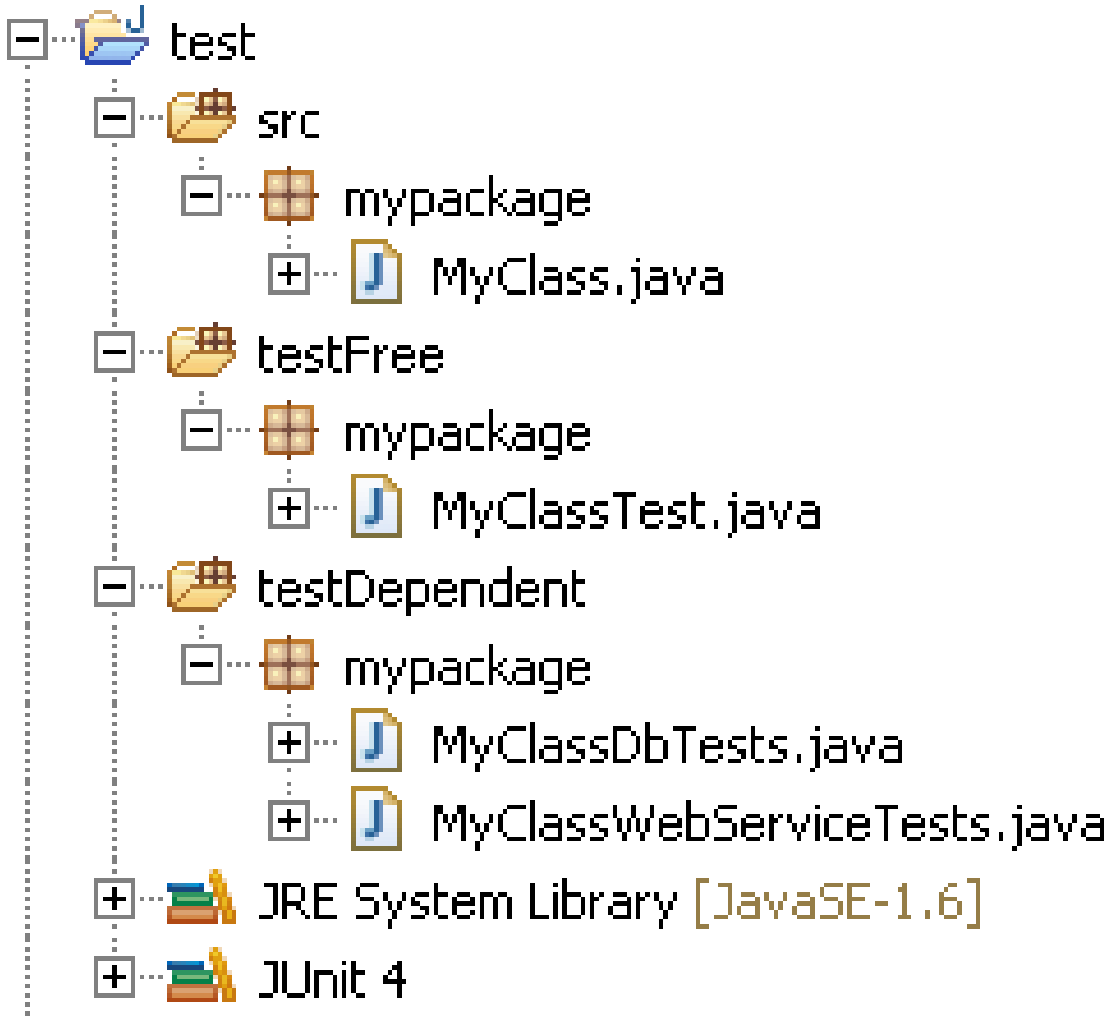
Testide paigutus



Testide paigutus



Testide paigutus



Arrange Act Assert

```
@Test
public void pushIncreasesSize() {

    Stack stack = getStack();

    stack.push(1);

    assertThat(stack.getSize(), is(1));
}
```

Privaatmeetodite testimine

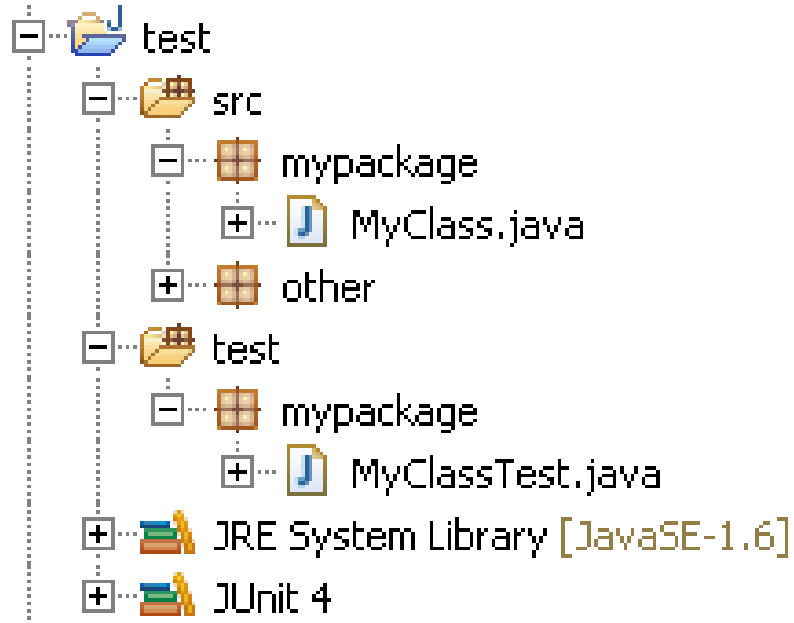
Reflection

```
Method method = targetClass.getDeclaredMethod("someMethod", argClasses);

method.setAccessible(true);

method.invoke(targetObject, argObjects);
```

Nähtavuse muutmine

private -> public, protected, “package private” (vaikimisi)

```
private int calculate() {}

public int calculate() {}

protected int calculate() {}

int calculate() {}
```



Nähtavuse muutmise probleemid

- Avalik liides
- Turvalisus

@deprecated märge

DeprecatedMethodCaller.java | DeprecatedMethods.java

```
1  package test;
2
3  public class DeprecatedMethods {
4
5      public void calculateSth() {
6          submethod();
7      }
8
9      /**
10      * @deprecated
11      */
12     void submethod() {
13
14     }
15 }
```

DeprecatedMethodCaller.java | DeprecatedMethods.java

```
1  package test;
2
3  public class DeprecatedMethodCaller {
4
5      public void callsDeprecatedMethod() {
6          new DeprecatedMethods().submethod();
7      }
8  }
9
```

Se | Ou | JU | An

- test
- DeprecatedMethods
  - calculateSth() : void
  - submethod() : void

Problems | Javadoc | Declaration | Search | Console | Call Hierarchy

0 errors, 1 warning, 0 others

Description	Resource	Path	Location	Type
Warnings (1 item)				
The method submethod() from the type Depre	Deprecate...	/test/src/test	line 6	Java Problem

*Utils klassid

Abiklass

```
public class DepositUseCase {
    public boolean deposit() {
        // ..
        // use many sub methods
        return false;
    }

    private int getBalance() {
        // do some work
    }
}
```


```
public class DepositUseCaseHelper {
    public int getBalance(/* arguments */) {
        // do some work
    }
}
```

Millist eelistada?

- Testida ainult läbi avaliku liidese
- Muuta nähtavust (vajadusel @deprecated märge)

Dependency Injection (DI)

```
public Money getOrderTotal(String orderId) {
  Dao dao = new DbDao();

  List<Money> orderAmounts = dao.getOrderAmounts(orderId);

  Money total = new Money(0, "EUR");

  for (Money amount : orderAmounts) {
    total = total.plus(amount);
  }

  return total;
}
```

Parameter injection

```
public Money getOrderTotal(Dao dao, String orderId) {

  List<Money> orderAmounts = dao.getOrderAmounts(orderId);

  Money total = new Money(0, "EUR");

  for (Money amount : orderAmounts) {
    total = total.plus(amount);
  }
  return total;
}

OrderService orderService = new OrderService();
orderService.getOrderTotal(new TestDao(), "123");
```

Setter injection

```
public class OrderService {

  private Dao dao;

  public void setDao(Dao dao) {
    this.dao = dao;
  }

  public Money getOrderTotal(String orderId) {
    List<Money> orderAmounts = dao.getOrderAmounts(orderId);
    ...
  }
}

OrderService orderService = new OrderService();
orderService.setDao(new TestDao());
orderService.getOrderTotal("123");
```

```
public class PolicyServiceImpl implements PolicyService {

    @Resource
    protected PolicyDao policyDao;
    @Resource
    private TimeDao timeDao;
    @Resource
    private SysParameterService sysParameterService;
    @Resource
    private InsuredObjectService insuredObjectService;
    @Resource
    private PolicyCustomerService policyCustomerService;
    @Resource
    private UserService userService;
    @Resource
    private CustomerService customerService;
    @Resource
    private KnowledgeBaseService knowledgeBaseService;
    @Resource
    private InvoiceRowService invoiceRowService;
    @Resource
    private ObjectFactory<InvoiceRowSynchronizer> invoiceRowSynchronizerFactory;
    @Resource
    private SpringBeanInjector springBeanInjector;
    @Resource
    private DepositBalanceService depositBalanceService;
    @Resource
    private InvoiceService invoiceService;
    @Resource
    private AuthenticationService authenticationService;
    @Resource
    private PolicyWorkbenchService policyWorkbenchService;
    @Resource
    private DivisionService divisionService;
    @Resource
    private VehicleService vehicleService;
```

DI raamistikud (Spring)

```
public class OrderService {

  @Resource
  private Dao dao;

  public Money getOrderTotal(String orderId) {
    List<Money> orderAmounts = dao.getOrderAmounts(orderId);
    ...
  }
}
```

```
OrderService service = new OrderService ();
service.getOrderTotal("123");
```

Vs.

```
OrderService s = (OrderService) context.getBean("orderService");
service.getOrderTotal("123");
```

DI raamistikud

```
@Test
public void testOrderTotal() throws Exception {
  OrderService service = new OrderService();
  service.dao = new TestDao();  // compilation error

  service.getOrderTotal(orderId);

  ...
}
```

Reflection

```
import org.apache.commons.lang3.reflect.FieldUtils;


...


OrderService service = new OrderService();
FieldUtils.writeField(service, "dao", new TestDao());
```

Beaninject

```
Inject.field("field").of(target).with("value");

Inject.property("propertyName").of(target).with("value");

Inject.bean(target).with("value");

Inject.bean(bankService).with(bankDao);
```

https://github.com/lkoskela/beaninject

Ülesanne 8: injection

```
@Test
public void totalIncome1() {
    Report report = new TestableReport(new TestableBank());

    Money total = report.getTotalIncomeBetween(null, null);

    assertThat(total, is(new Money(2, "EUR")));
}
```

Delegated setup

```
@Test
public void applyTenPrecentDiscountCreatesDiscountRow()
      throws Exception {

  Order order = getSmallOrder();

  ...
```

Custom assertion

```
@Test
public void applyTenPrecentDiscountCreatesDiscountRow()
    throws Exception {

  Order order = getOrder();

  order.applyDiscount();

  assertContainsItemWithPrice("pen", 1.5, order);
}
```

Sisend või oodatav tulemus failist

```
@Test
public void htmlToJasper() throws Exception {
    String html = "<p>hello<br /><strong>one <em>two </em>three<br /></strong></p>"
                + "<p><span style=\"text-decoration:underline;\">second </span>paragraph</p>"
                + "<p></p>"
                + "<ul><li>item 1</li></ul>";

    String expected = "hello<br />"
                + "<b>one <i>two </i>three<br /></b><br />"
                + "<u>second </u>paragraph<br /><br />"
                + "<li>item 1</li>";
    assertThat(new Converter().htmlToJasper(html), is(expected));
}
```

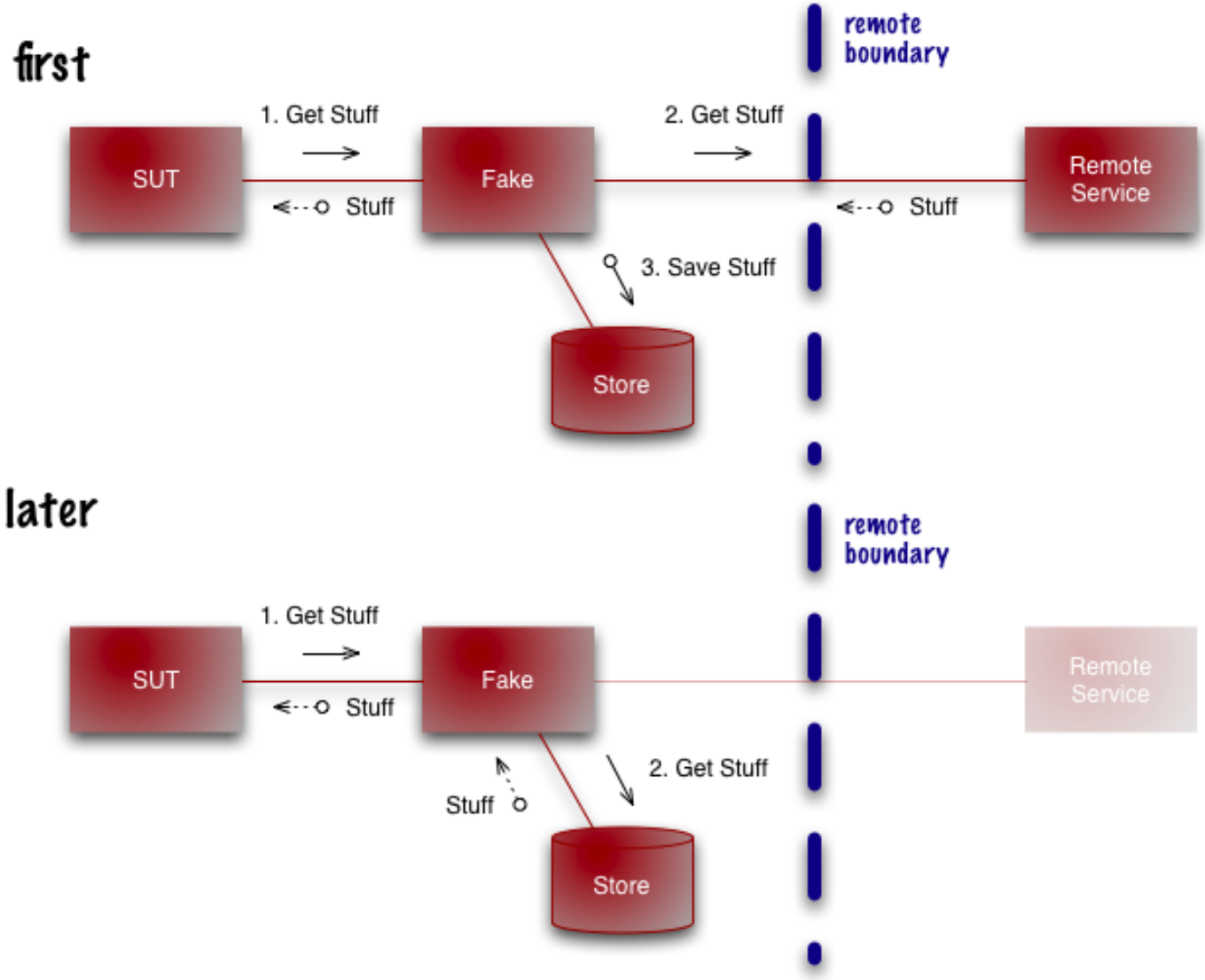
```
String html = TestUtils.getFile("c:/workspace/proj1/input.html");
```

Sisend või oodatav tulemus failist

```
String html = readFromFile("input.html");
```


```
public String readFromFile() {
    String filePath = getClass().getResource("input.html").getFile();

    ...
}
```

Self Initializing Fake



Tulemuse teisendamine stringiks

```
@Test
public void producesRightOrders() throws Exception {
  Order2 order1 = new Order2();
  order1.setName("pen");
  order1.setQuantity(2);

  Order2 order2 = new Order2();
  order2.setName("paper");
  order2.setQuantity(5);

  assertThat(getOrders(),
             is(Arrays.asList(order1, order2)));
}
```

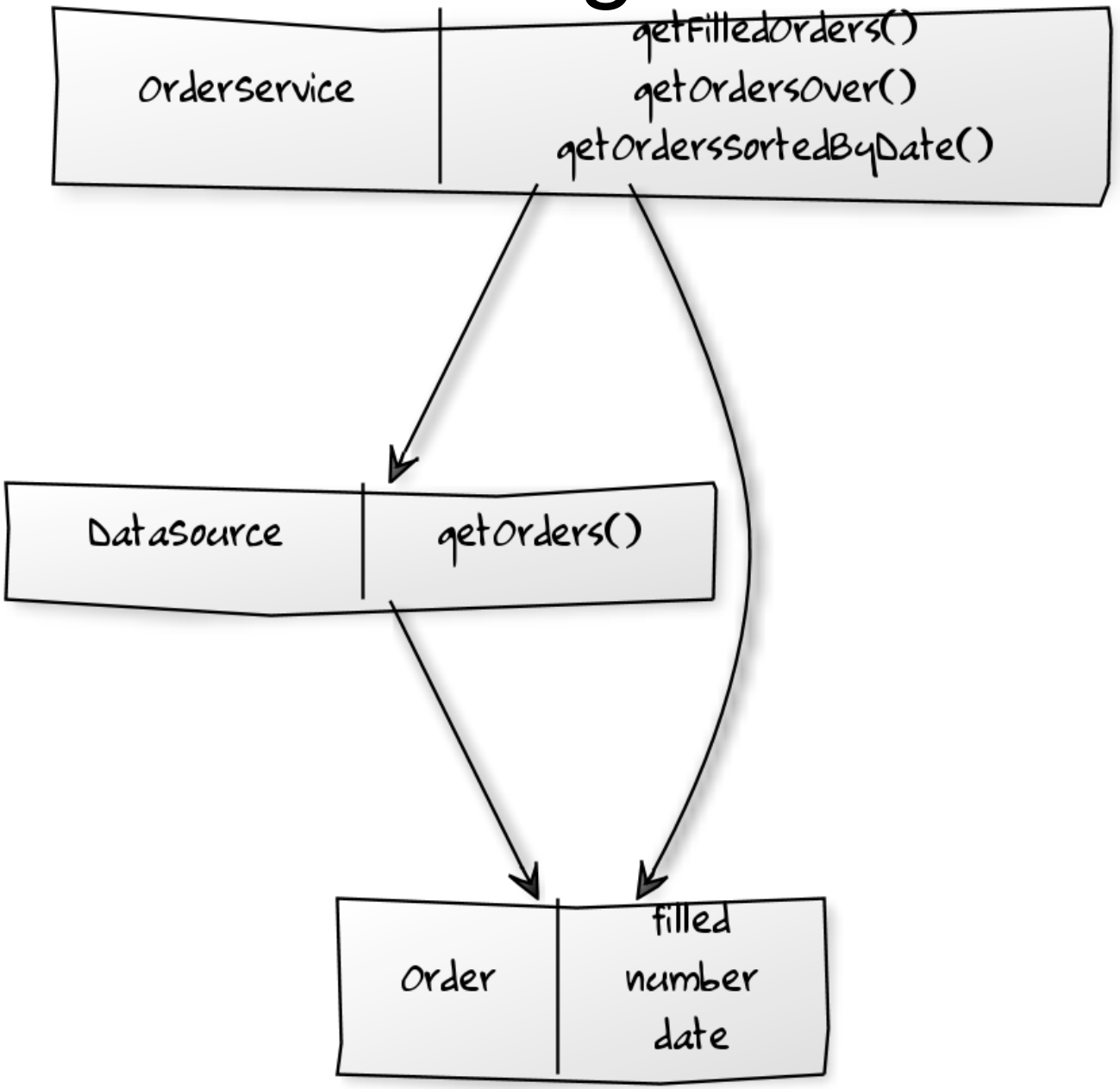
Tulemuse teisendamine stringiks

```
@Test
public void rightOrderItems() {
  assertThat(getOrders().toString(), is("[pen(2), paper(5)]"));
}


System.out.println(Arrays.asList(1, 2, 3).toString());
// [1, 2, 3]

System.out.println(Arrays.asList(order1, order2).toString());
// [pen(2), paper(5)]


@Test
public void rightOrderItems2() {
  assertEquals("[pen(2) 2011-10-01, paper(5) 2011-10-05]",
            getOrders().toString());
}
```

Tulemuse teisendamine stringiks

```
@Test
public void rightOrderArticles() throws Exception {
  assertThat(articleNamesAsString(getOrders()), is("pen, paper"));
}

@Test
public void rightOrderArticleDates() throws Exception {
  assertThat(articleDatesAsString(getOrders()), is("01.10, 05.10"));
}

private Object articleNamesAsString(List<Order2> orders) {
  List<String> names = new ArrayList<String>();
  for (Order2 order : orders) {
    names.add(order.getName());
  }
  return StringUtils.join(names, ", ");
}
```

Ülesanne 9: tulemuste teisendamine stringiks



```
@Test
public void getFilledOrders() {
    OrderService service = getOrderService();

    assertThat(orderNumbersAsString(service.getFilledOrders()),
               is("..."));
}
```

Testable Object

```
@Test
public void testReportTotalIncome() {
    Bank bank = mock(Bank.class);
    Dao dao = mock(Dao.class);
    AccountingService accountingService =
                          mock(AccountingService.class);
    Report report = new Report();
    Inject.bean(report).with(bank, dao, accountingService);

    report.generate();

    verify(dao).save(anyObject());
}
```

Testable Object

```
@Test
public void testReportTotalIncome() {
    Report report = getReport();

    report.generate();

    verify(dao).save(anyObject());
}
```

Testable Object

```
private Bank bank;
private Dao dao;
private AccountingService accountingService;

private Report getReport() {
    bank = mock(Bank.class);
    dao = mock(Dao.class);
    accountingService = mock(AccountingService.class);
    Report report = new Report();
    report.setDao(dao);


    return report;
}
```

Testable Object

```
class TestableReport extends Report {

    public Bank bank;

    public Dao dao;

    public AccountingService accountingService;

    @Override
    public void setDao(Dao dao) {
        super.setDao(dao);
        this.dao = dao;
    }
}
```

```
verify(report.dao).save(anyObject());
```

Object Mother

```
ObjectMother objectMother = new ObjectMother();

Order order = objectMother.simpleOrder();

orderProcessor.process(order);



objectMother.setCustomerAddress(..
```

Object Mother

```
objectMother.customerWith1MdollarsSharesAndDrivesAPorsche()

objectMother.wealthyCustomer()
```

Personas

Bob is 52 years old and works as a mechanic with an organisation offering road service to customers when their car breaks down. He has worked in the job for the past 12 years and knows it well.

...

```
objectMother.getCustomerBob()
```

Builder

```
@Test
public void fixtureBuilderExample() throws Exception {
  Person p = new Person("Tiit", 20);
  Address addr = new Address();
  addr.setTown("Tallinn");
  p.setAddress(addr);
  p.setMail("test@test.ee");
  ...

  Person person = getPerson();
  person.setAge(21);
  Address address = new Address();
  addr.setTown("Tallinn");
  person.setAddress(address);
}
```

Builder

```
@Test
public void fixtureBuilderExample1() throws Exception {
  Person person = new PersonBuilder()
    .withTown("Tallinn")
    .withAge(18)
    .build();




  Person person1 = aPerson().withTown("Tallinn").build();
  Person person2 = aPerson().withTown("Tartu").build();
  Person person3 = aPerson().withTown("Narva").build();
}
```

```
public class PersonBuilder {
  private String name = "Default name";
  private String address = "Default address";
  private String mail = "Default mail";
  private int age = 21;

  public PersonBuilder withName(String name) {
    this.name = name;
    return this;
  }

  public PersonBuilder withAge(int age) {
    this.age = age;
    return this;
  }

  public Person build() {
        Person p = new Person(name, age);
        p.setAddress(address);
        p.setMail(mail);
        return p;
  }
}
```

Ülesanne 10: Builder muster

Java EE

Servlet-i testimine

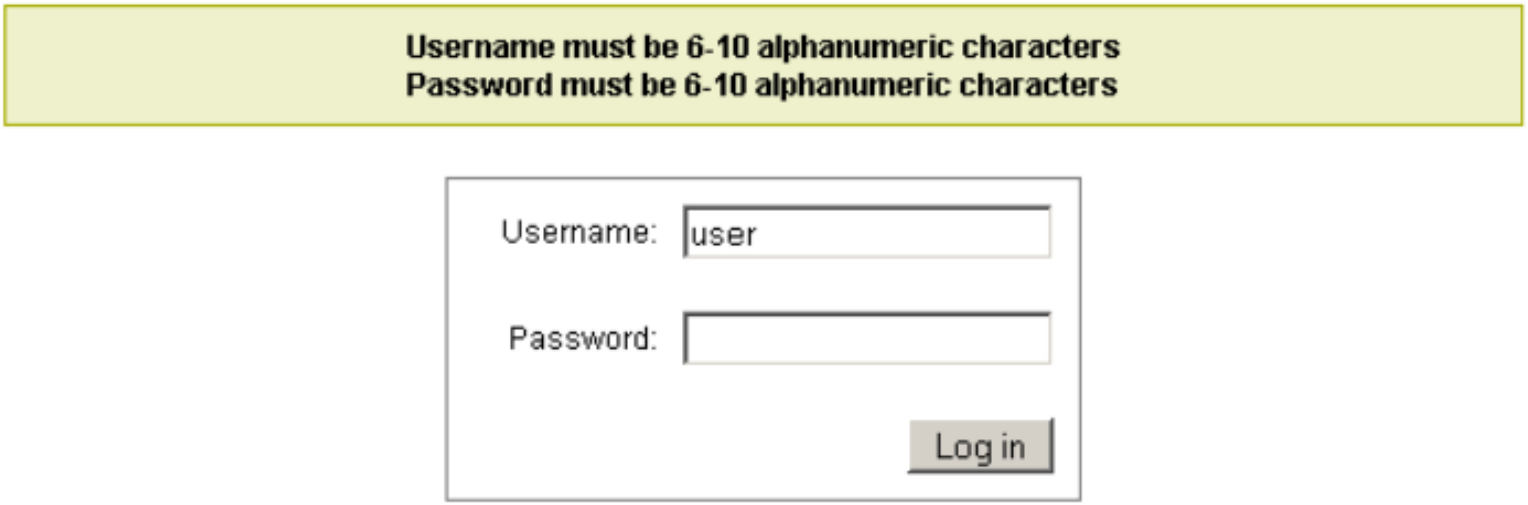
```
public class HelloServlet extends HttpServlet {

  @Override
  public void doGet(HttpServletRequest request,
      HttpServletResponse response) throws IOException {

    response.getWriter().println("Hello!");
  }
}
```

Servlet-i testimine

```
@Test
public void testDoGet() throws Exception {
  HttpServletResponse responseMock =
                          mock(HttpServletResponse.class);

  StringWriter stringWriter = new StringWriter();
  PrintWriter printWriter = new PrintWriter(stringWriter);

  when(servletResponseMock.getWriter()).thenReturn(printWriter);

  new HelloServlet().doGet(null, servletResponseMock);

  assertThat(stringWriter.toString(), is("Hello!"));
}
```

Spring MVC

Testimise raamistik



Testimise raamistik

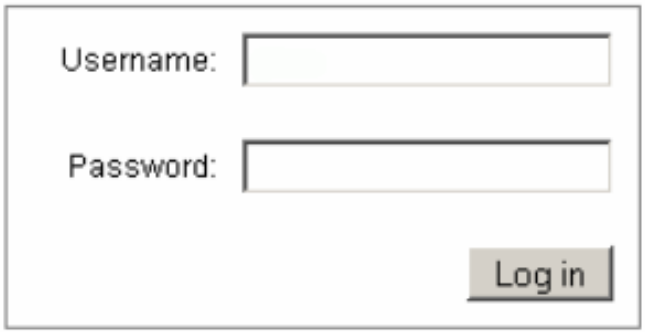
```
@Test
public void presentForm() {
  Result r = ct.execute("loginPage"); // login.html?..
  assertThat(r.getViewName(), is("loginForm"));

  assertThat(r.div("loginForm"), contains("login.label.username"));
  assertThat(r.div("loginForm"), contains("login.label.password"));
  assertThat(r.div("loginForm"), contains("login.label.login"));

  print(r);
  preview(r); // import org.eclipse.swt.browser.Browser;
}
```

<st:message key="login.label.username" />

Username

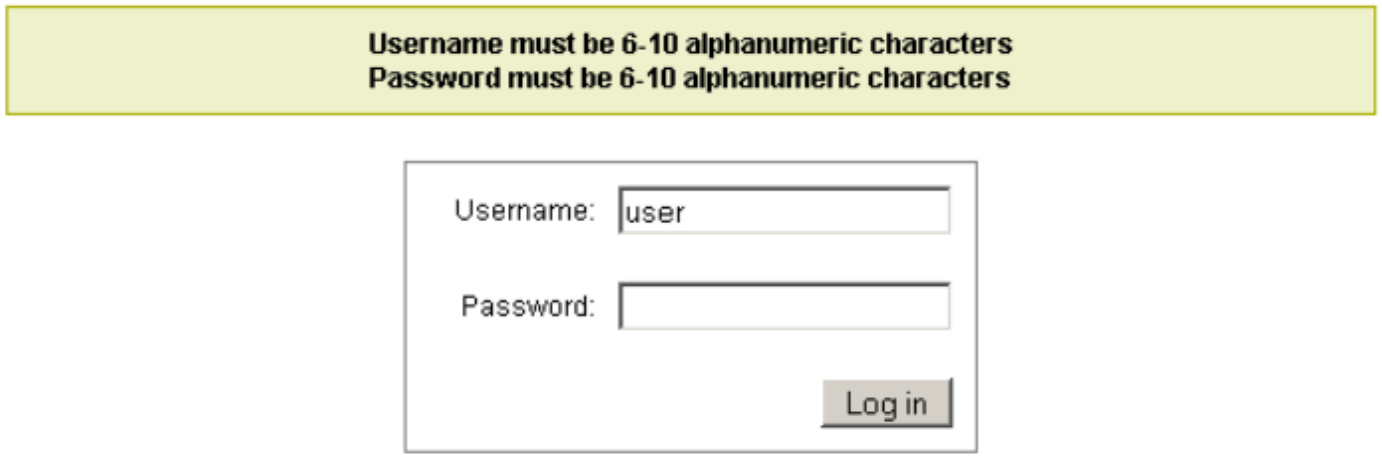
Vs.

Username



Testimise raamistik

```
@Test
public void submitInvalidUserOrPassword() {
  ct.addParam("username", "user"); // login.html?username=user&..
  ct.addParam("password", "123");
  ct.buttonPressed("loginButton");

  Result r = ct.execute("loginPage");

  assertThat(r.div("messageBox"), contains("login.message.invalid.user"));
  assertThat(r.div("messageBox"), contains("login.message.invalid.pass"));
}
```



Testimise raamistik

- http://code.google.com/p/controllertester/

Võimalik arenduskäik

```
public List<StatementRow> generateStatement(Parameters params) {

      System.out.println(params.getStartDate());

      // get data from db
      // get some configuration parameters
      // do some calculations and transformations
      // ...

      return Arrays.asList(new StatementRow());
}
```

```
private void getOrders() {
    return Arrays.asList(new Order("Pen", 1),
            new Order("Paper", 2));
}
```

Võimalik arenduskäik

- Saab:
  - käivitada väljaspool serverit (aeg, mugavus);
  - valmis töölaud.
- Jääb ära:
  - parem kontroll koodi osade üle;
  - kõrgem kvaliteet

Testimise tööriistad

Powermock

```
public int calculateDiscount() {
    return calculateDiscountSub();
}

private int calculateDiscountSub() {
    return 0;
}
```

```
@Test
public void mainpulatePrivateMethods() throws Exception {
    OrderService orderService = spy(new OrderService());
    when(orderService, "calculateDiscountSub").thenReturn(1);
    assertThat(orderService.calculateDiscount(), is(1));
}
```

Powermock

```
public static int calculateDiscount(String orderId) {
    throw new IllegalStateException("not implemented");
}
```

```
@Test
public void testMockStaticWithExpectations() throws Exception {
    mockStatic(OrderService.class);

    when(OrderService.calculateDiscount(null)).thenReturn(1);

    assertThat(OrderService.calculateDiscount(null), is(1));
}
```

Powermock

```
public Money getOrderTotal(String orderId) {
    DbDao dao = new DbDao();

    List<Money> orderAmounts = dao.getOrderAmounts(orderId);
    Money total = new Money(0, "EUR");
    for (Money amount : orderAmounts) {
        total = total.plus(amount);
    }
    return total;
}

@Test
public void replaceNew() throws Exception {
    OrderService orderService = new OrderService();
    DbDao dbDaoMock = mock(DbDao.class);

    whenNew(DbDao.class).withNoArguments().thenReturn(dbDaoMock);
    when(dbDaoMock.getOrderAmounts(null)).thenReturn(
        Arrays.asList(new Money(1, "EUR")));

    assertThat(orderService.getOrderTotal(null), is(new Money(1, "EUR")))
}
```

Powermock

```
@RunWith(PowerMockRunner.class)
@PrepareForTest({ DbDao.class, OrderService.class })
public class PowerMockTests {

}
```

Agitar

- http://www.agitar.com/downloads/demos/junit_gen/junit_generation_skin.swf

Agitar

```
public void testSort() throws Throwable {
    int[] num = new int[0];
    BubbleSort.sort(num);
    assertEquals("num.length", 0, num.length);
}

public void testSort1() throws Throwable {
    int[] num = new int[3];
    num[1] = 1;
    BubbleSort.sort(num);
    assertEquals("num[0]", 1, num[0]);
}

public void testSortThrowsNullPointerException() { ...
```

Agitar

```
public void testConstructor() throws Throwable {
    Stack stack = new Stack(100);
    assertNotNull("stack.stack",
       getPrivateField(stack, "stack"));
    assertEquals("stack.getSize()", 0, stack.getSize());
}

public void testEnsureStackIsNotEmpty() throws Throwable {
    Stack stack = new Stack(100);
    stack.push(100);
    stack.push(1000);
    callPrivateMethod("mrt.Stack",
       "ensureStackIsNotEmpty", new Class[] {}, stack,
       new Object[] {});

    assertEquals("stack.getSize()", 2, stack.getSize());
}
```

Agitar

```
public void testTransferMoney1() throws Throwable {

    BankService bankService = (BankService)
       Mockingbird.getProxyObject(BankService.class);

    TransferService transferService = new TransferService(bankService);

    Mockingbird.enterRecordingMode();
    Mockingbird.setReturnValue(bankService.getBalance("\n"), 70);
    bankService.transfer(1, "\n", "");
    Mockingbird.setNormalReturnForVoid();
    Mockingbird.enterTestMode(TransferService.class);

    transferService.transferMoney(1, "\n", "");

    assertSame("transferService.bankService",
          bankService,
          getPrivateField(transferService, "bankService"));
}
```

Matcher-id

```
// 2 + 1 on 3
assertEquals(3, 2 + 1);
```

Võrdub

```
assertEquals(3, 2 + 1);


public void assertEquals(int a, int b) {
    if (a != b) throw new AssertionError("not equal");
}
```

Suurem kui

```
// 2 > 1
assertTrue(2 > 1); // teade

assertIsGreaterThan(2, 1);
```

```
assertThat(actual,
        new IsGreaterThanMatcher(expected));

assertThat(actual, isGreaterThan(expected));

assertThat(2, isGreaterThan(1));
```

```
assertThat(2, new IsGreaterThanMatcher(1));


public class IsGreaterThanMatcher
                    extends TypeSafeMatcher<Integer> {

    private Integer expected;

    public IsGreaterThanMatcher(Integer expected) {
        this.expected = expected;
    }

    @Override
    protected boolean matches(Integer actual) {
        return matchedValue > expected;
    }
}
```

```
assertThat(2, isGreaterThan(4));

public static <T> void assertThat(
        int actual, Matcher<? super T> matcher) {


    if (!matcher.matches(x))
        throw new AssertionError("does not match");
}



assertThat(1 + 2, is(3));
```

```
@Override
public void describeTo(Description description) {
   description.appendText("greater than " + expected);
}
```

Expected: greater than 4
but: was <2>

Mockito

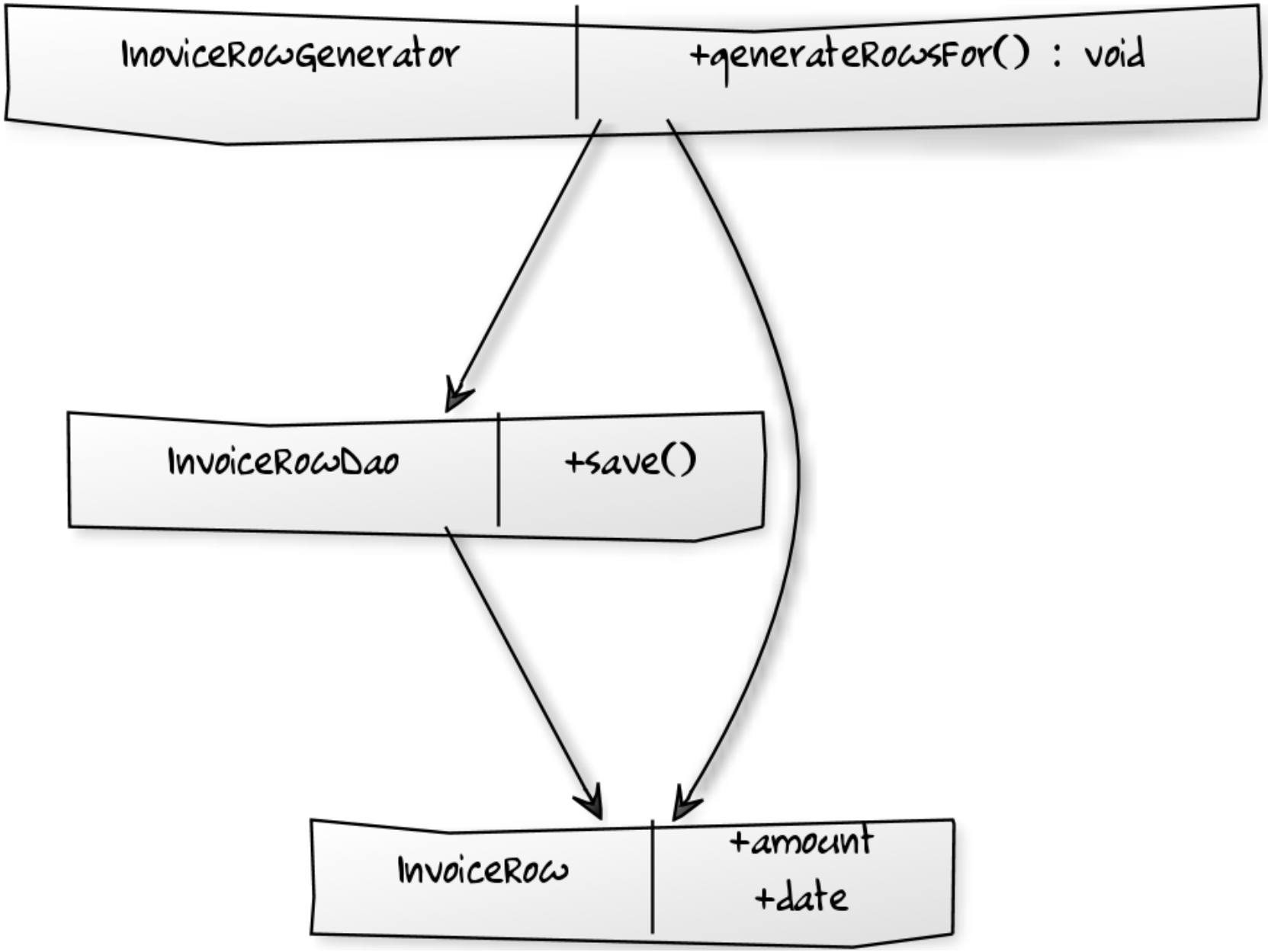
```
when(bankService.getAccountCurrency("123"))...

when(bankService.getAccountCurrency(anyString()))...

when(bankService.getAccountCurrency(startsWith("prefix"))).
```

Mockito

```
Invoice invoice = new Invoice();
invoice.addRow(...);
...

dao.save(invoice)
```

```
verify(dao).save(...);
verify(dao).save(invoiceWithTotalOf(10));
verify(dao).save(invoiceWithDate(...));
```

Ülesanne 11: Mockito matcher



```
@Test
public void amountsAreCorrect() throws Exception {

    InvoiceRowDao dao = mock(InvoiceRowDao.class);
    InoviceRowGenerator generator = new InoviceRowGenerator();
    Inject.bean(generator).with(dao);

    generator.generateRowsFor(new BigDecimal(10),
        asDate("2012-02-15"), asDate("2012-04-02"));

    verify(dao, times(2)).save(argThat(hasAmountOf(3)));
    verify(dao).save(argThat(hasAmountOf(4)));


    verifyNoMoreInteractions(dao);
}
```

```
public void generateRowsFor(BigDecimal amount,
                            Date periodStart, Date periodEnd) {

    ...


    for (Payment payment : payments) {
        dao.save(new InvoiceRow(payment.amount, payment.date));
    }
}
```

Kokkuvõtteks