Testning av Javaprogram

Objektorienterad programmering
Laboration 4

Syfte

Att ge en praktisk övning i testning av Javaprogram.

Mål

Efter övningen skall du kunna

- enhets- och regressionstesta klasser med JUnit i BlueJ.
- använda avlusaren i BlueJ.

Utvecklingsmiljö

BlueJ på valfri plattform.

Litteratur

Kursboken kap 1-6. För att klara uppgifterna krävs att du behärskar avsnitt 6.1-4.

Färdig programkod

Given programkod finns på kursens hemsida i **Laborationer->Programkod för labbarna ->bluej_projects.zip**. Exemplet ligger **chapter03/mail-system**.

Labbgrupper

Arbetet genomförs i grupper om **två** personer.

Redovisning

Senaste redovisningsdag, se kurs-PM samt Fire.
Ladda upp filerna `MailServer`, `MailServerTest`, `MailItem`, `MailItemTest`, samt `MailClient` med dina lösningar i, samt `README.txt` med allmänna kommentarer om lösningen, i Fire. Redovisa slutversionen av resp. klass, alltså som den ser ut efter att du löst samtliga 8 uppgifter.

Glöm inte att trycka på **SUBMIT***!*

Problembeskrivning

Uppgifterna går ut på att bygga ut e-posthanteringssystemet i avsnitten 3.12-13 med spamfiltrering, samt att konstruera ett antal testfall för de inblandade klasserna. Testfallen finns sammanfattade i tabellform sist i detta PM - *använd namngivningen där för att underlätta redovisningen! Fel namngivning ger retur.*

Uppvärmning

Ställ in BlueJ så att testverktygen syns i fönstret:
Kryssa Tools->Preferences->Miscellaneous->Show unit testing tools.

Uppgift 1 Ämnesfält i brevhuvudet

Inför begreppet subject i `MailItem` så att man kan ange ämne för sina brev. Även klassen `MailClient` behöver modifieras för att hantera tillägget.

Uppgift 2 Några enkla testfall för `MailItem`

Lägg först till en testklass genom att högerklicka på klassikonen för MailItem och välja Create Test Class. Gör en testmetod `testAccessors` som kontrollerar att respektive accessmetod returnerar rätt värde.
Ex. Om ett objekt skapas med `new MailItem("FROM","TO","SUBJECT","BODY")` så skall `getFrom()` för det objektet returnera `"FROM"`, och analogt för de tre övriga accessmetoderna. Ingen objektfixtur behövs utan skriv in testmetoden i testklassen direkt. Kör testet!

Uppgift 3 Kasta brev med tomt avsändar- eller mottagarfält

Ändra metoden `post()` i serverklassen så att brev med tom mottagare eller avsändare ignoreras. Inför en boolesk privat hjälpmetod som avgör om ett meddelande skall kastas eller ej.

Testning av serverklassen

Det är ganska meningslöst att testa klassen `MailClient` isolerat eftersom alla dess metoder är direkt beroende av ett serverobjekt. Vi koncentrerar oss därför på att testa serverklassen. För att underlätta konstruktionen av kommande testfall börjar vi att bygga upp en *objektfixtur* som kan användas av flera testmetoder. Faktarutan på nästa sida beskriver hur man gör, studera den innan du går vidare med uppgift 4.

Uppgift 4 Testfall för serverklassen

> Testning av servern kan göras mer eller mindre ambitiöst. Ex: en enkel kontroll av att ett brev kommer fram till en viss mottagare är att verifiera att antalet brev till mottagaren ökar med ett med metoden `howManyMessages`. Mer detaljerad information fås förstås genom att kontrollera att det skickade brevet verkligen är likadant som det som tas emot med `getNextMailItem`.

Inför först en parameterlös accessmetod `int howManyMessages()` som returnerar det totala antalet brev i servern. Metoden är användbar här men även i andra testfall.

För att testa servern behövs en testklass, ett serverobjekt och ett antal objekt av klassen `MailItem`. Se tabellen s. 7 för namngivning. Vi lägger objekten i en fixtur i testklassen.

1. Skapa först en testklass till serverklassen.
2. Skapa följande objekt på objektbänken (som skall vara tom när du startar):
    a. Ett serverobjekt

   b. Ett meddelandeobjekt med tom avsändare och icke tom mottagare (övriga fält kan lämnas tomma).
   c. Ett meddelandeobjekt med icke tom avsändare och tom mottagare (övriga fält kan lämnas tomma).
3. Spara objekten som en fixtur i testklassen.
4. Testfall 1: Skapa testmetoden `testEmptyFrom`. Spela in följande scenario genom lämpliga metodanrop på objekten i objektbänken:
   a. Posta meddelandet med tom avsändare.
   b. Kontrollera att antalet lagrade meddelanden i servern är noll.
5. Testfall 2: Skapa testmetoden `testEmptyTo` på analogt sätt.
6. Kör de två testen.

Fler testfall

Gå vidare med testfall 3 i tabellen s. 7. Resterande test kommer senare. I mer komplicerade fall krävs att du redigerar koden för testmetoderna eftersom inspelningsfaciliteten i BlueJ inte alltid är tillräckligt flexibel.

Hantering av objektfixtur

En samling objekt på objektbänken kan sparas som en s.k. *objektfixtur* i en eller flera testklasser. Olika testklasser kan ha olika fixturer. För att spara en viss objektuppsättning som en fixtur i en testklass högerklickar man på testklassens ikon och väljer Object Bench to Test Fixture. Varje exekvering av en testmetod sker med objekten i originaltillstånd. Testen kan alltså köras i valfri ordning eftersom de ej kan påverka varandra.

1. Skapa testklassen.
2. Skapa önskade objekt på objektbänken. Anropa ev. metoder för att ”ställa in” rätt grundtillstånd i objekten.
3. Spara objekten som en fixtur i testklassen.
4. Skapa testmetod genom högerklick på testklassens ikon och välj Create Test Method…
5. Spela in testfallet genom att anropa önskade metoder på objekten, beroende på vad testfallet handlar om.
6. Avsluta inspelningen med tryck på knappen End för att spara objektbänkens objekt som en fixtur i testklassen, eller avbryt med Cancel.
7. Exekvera en eller flera testmetoder. Högerklicka på testklassens ikon så kommer listan med testmetoder upp. TestAll kör alla testmetoder för aktuell testklass och presenterar resultaten samlade i en tabell. Objektfixturen behöver ej vara öppen när testen körs.

För att bygga ut objektfixturen öppnar man den på objektbänken med Test Fixture to Object Bench och lägger till önskade objekt, anropar metoder etc. Se upp så att du inte av misstag gör detta flera gånger i följd så att extra objekt skapas. Om detta händer måste testklassen städas från överflödiga testobjekt manuellt.

STOPP!

Gör klart och kör testfallen 1-3 innan du fortsätter!

Gå aldrig vidare till nästa punkt förrän serverklassen har passerat alla test.

Refaktorering (= ombyggnad) av serverklassen

Vi skall nu ändra datarepresentation och därefter bygga ut serverklassen. Alla tidigare test skall upprepas efter *varje* förändring (regressionstestning) för att kontrollera att ändringen inte förstört tidigare fungerande beteende.

Uppgift 5 Bättre datalagring för brev i serverklassen

I originalversionen ligger alla brev huller om buller i serverns interna lista. Ändra lagringen av brev i servern så att den istället har en `HashMap` som kopplar ihop varje användarnamn med en egen brevlåda i form av en lista med element av typen `MailItem`. Typen blir alltså något i stil med: `HashMap<String,`*`listtyp`*`<MailItem>>`.

Inför metoderna

```
/**
 * Create a mailbox for a single user
 * @param  the name of the new user
 * @return 1 if the creation succeeded
 *         0 if a mailbox with that name already existed
 */
public int createMailbox(String user)

/**
 * Create mailboxes for users
 * @param  an array containing the names of the new users
 * @return the number of successfully created mailboxes,
 *         this number will be less than the number of
 *         specified users if some mailboxes already existed
 */
public int createMailbox(String[] users)
```

Använd dem sedan för att skapa nödvändiga brevlådor innan du testar annan funktionalitet!

Breven skall hädanefter hanteras i omvänd tidsordning så att det senast skickade tas emot först av klienten. Välj den listklass som lämpar sig bäst för din lösningsmetod. Utvidgningen underlättas om du först definierar följande metod (använd den så ofta du kan):

```
/**
 * Retrieves the mailbox for a user
 *
 * @param   who the owner of the requested mailbox
 * @return  the mailbox belonging to who,
 *          null if it does not exist
 */
private listklass getMailbox(String who)
```

Regressionstesta serverklassen! Testfall 1-6. *Avlägsna ”död” kod innan du går vidare!*

Uppgift 6 Gruppmail

Förbättra systemet så att brev kan sändas till flera mottagare. Mottagarna avgränsas med kommatecken i adressfältet, t.ex. `”user1,user2,user3”`. Ändra metoden `getTo` i `MailItem` så att den returnerar ett fält av mottagarnamn (använd `split`):

```
String[] getTo();
```

Adressfältet bör inte vara känsligt för förekomst av blanktecken runt kommatecknen. Implementera och kör testfallet `testTwoReceivers` som skickar ett brev från `"user1"` till mottagarna `"user2"` och `"user3"`. Testet skall kontrollera att brevet finns i brevlådorna hos båda mottagarna.

Regressionstesta serverklassen! Testfall 1-7

Uppgift 7 ”Return to sender”

Mottagare som saknar brevlåda i servern betraktas som okänd. Alla brev som skickas till okända mottagare skall returneras till avsändaren. Exempel: Om Allan har försökt skicka ett meddelande till Elvis, som oturligt nog inte har någon brevlåda, så skall Allan få tillbaka meddelandet:

```
From:     Postmaster
To:       Allan
Subject:  Unknown receiver: Elvis
Message:
Transcript of original message follows:
---------------------------------------
> From:     Allan
> To:       Elvis
> Subject:  Javabok
> Message:
> Hej! Kan jag låna din javabok till i morgon? /Allan
```

För att underlätta inkopieringen av originalmeddelandet i returbrevet börjar vi med att införa metoden `toString` och ändrar `print` i klassen `MailItem` enligt följande:

```
/**
 * Print this mail message to the text terminal.
 */
public void print()
{
    System.out.println(toString());
}

/**
 * compile a text representation of this object
 */
public String toString()
{
    return
        "\nFrom:     " + from +
        "\nTo:       " + to +
        "\nSubject:  " + subject +
        "\nMessage:\n" + message;
}
```

Implementera nu metoden

```
/**
 * Return a message with unknown receiver to the sender
 *
 * @param the message to be returned
 */
private void returnToSender(MailItem message, String receiver)
```

och använd den i metoden `post`. Parametern `receiver` avser den okända mottagaren. Använd lämplig metod i strängklassen för att stoppa in >-tecknen som i exemplet ovan.
Anm. Lösningen skall givetvis beakta fallet att även *avsändaren* kan sakna brevlåda.
Regressionstesta serverklassen! Testfall 1-8.

Uppgift 8 Spamfiltrering i serverklassen

Servern får inte posta brev med misstänkt innehåll i någons brevlåda. Ett brev betraktas som spam om Subject innehåller ordet SPAM eller om meddelandekroppen någonstans innehåller texten Viagra, stavat med små eller stora bokstäver och ev. förvanskat genom instoppade blanktecken och upprepade bokstäver, som t.ex. i ”V i AA gggg RrR aA”. Sådana brev skall ignoreras helt av servern. Matchningen skall ske med strängklassens metod `matches()`. Studera denna metod i Java API, samt även länken ”regular expression”. Det finns sajter på nätet för att testa reguljära uttryck – vilket kan vara bra för att komma underfund med hur de fungerar. Sök t.ex. på ”regex tester java”.

Test av spamfiltret (Testfall 9-10)
Regressionstesta fallen 1-8 igen efter att du implementerat spamfiltret. Om alla test gick bra skall sviten utvidgas med följande två fall:
Lägg till två brevobjekt i objektfixturen för `MailServerTest:`

1. ett med teckenföljden `”xxxSPAMyyy”` i ämnesfältet.
2. ett med teckenföljden `"xxxV iiA gR a xxx"` någonstans i meddelandekroppen (minimikrav).

Filtreringen skall givetvis fungera generellt enligt beskrivningen i första stycket ovan och inte bara för exemplen i 1-2.
Konstruera två testmetoder (testfall 9-10) som kontrollerar att de två spambreven enl. ovan ej lagras av servern. Enklast är nog att posta dem och se om det totala antalet lagrade brev i servern ökar eller ej.

Regressionstesta serverklassen! Testfall 1-10

Uppgift X Frivilig Xtrauppgift

Hitta på egna tjänster/förbättringar i mailsystemet. Finns det viktiga egenskaper som inte testats ovan? Konstruera fler och bättre testfall!

Lycka Till!

Testfall för mailsystemet

Följande objekt och testmetoder skall konstrueras (minimikrav). *Namnen i den vänstra kolumnen skall användas i lösningen!*

Testklass: `MailItemTest`		
Objektfixtur		**Förklaring**
-		*objektet skapas i testmetoden*
Testmetod		
`testAccessors`		Kontrollerar att resp. accessmetod returnerar rätt värde.
Testklass: `MailServerTest`		
Objektfixtur		**Förklaring**
`mailServer`		Ett serverobjekt. Efter att du gjort uppg. 5 skall brevlådor för `"user1"`, `"user2"` och `"user3"` skapas.
`messEmptyFrom`		Ett brev med tomt avsändarfält.
`messEmptyTo`		Ett brev med tomt mottagarfält.
`mess1User1to2`		Ett brev med olika avsändar- och mottagarfält samt subject `"1"`.
`mess2User1to2`		Som `mess1User1to2` men med subject `"2"`.
`messTwoReceivers`		Ett brev med avsändare `"user1"` och mottagare `"user2,user3"`.
`messUnknownReceiver`		Ett brev med avsändare `"user1"` och mottagare `"foo"`.
`messSpamSubj`		Ett brev med `"xxxSPAMyyy"` i subjectfältet.
`messSpamBody`		Ett brev med `"xv  I aaa G Rrr ayy"` i meddelandekroppen.
Testmetoder		**testar att**
1	`testEmptyFrom`	brev med tomt avsändarfält ej lagras i servern.
2	`testEmtyTo`	brev med tomt mottagarfält ej lagras i servern.
3	`testMessageRemoval`	att brev som hämtats med `getNextMailItem` ej finns kvar i servern.
	utför följande test efter *ombyggnaden av servern (uppg. 5)*	
4	`testMessageNotReadableToOthers`	brev ej kan hämtas av fel mottagare.
5	`testCreateMailboxes`	ett stort antal brevlådor (100) kan skapas. *Kräver editering av testmetoden.*
6	`testReceiveOrder`	brev mottas i omvänd sändningsordning (LIFO).
7	`testTwoReceivers`	ett brev med två mottagare kommer fram till dessa.
8	`testReturnToSender`	ett brev med okänd mottagare sänds åter till avsändaren.
9	`testSpamInSubject`	brev med spam i subject-fältet ej lagras av servern.
10	`testSpamInBody`	brev med spam i meddelandekroppen ej lagras av servern.

Objektfixtur

Objektfixturen för serverklassen skall se ut så här när den är färdigbyggd:

```
private MailServer mailServer;
private MailItem messEmptyFrom;
private MailItem messEmptyTo;
private MailItem mess1User1to2;
private MailItem mess2User1to2;
private MailItem messTwoReceivers;
private MailItem messUnknownReceiver;
private MailItem messSpamSubj;
private MailItem messSpamBody;

...


/**
 * Sets up the test fixture.
 *
 * Called before every test case method.
 */
protected void setUp()
{
    mailServer = new MailServer();
    mailServer.createMailbox("user1");
    mailServer.createMailbox("user2");
    mailServer.createMailbox("user3");
    messEmptyFrom = new MailItem("", "foo", "", "");
    messEmptyTo = new MailItem("foo", "", "", "");
    mess1User1to2 = new MailItem("user1", "user2", "1", "");
    mess2User1to2 = new MailItem("user1", "user2", "2", "");
    messTwoReceivers = new MailItem("user1", "user2,user3", "", "");
    messUnknownReceiver =
        new MailItem("user1", "unknown", "foo", "bar");
    messSpamSubj = new MailItem("user1", "user2", "xxxSpAmyyy", "");
    messSpamBody =
        new MailItem("user1", "user2", "", "xxxvI  a Gr AA yyy");
}
```