Institutionen för datavetenskap

9 december 2018

5DV135 (HT-18)
Applikationsutveckling i java
Obligatorisk uppgift 1

Teachers
Anders Broberg, Elias Åström, Jakob Lindqvist
Klas af Geijerstam Unger, William Viktorsson

Jonas Sjödin jonas@jonassjodin.com
https://github.com/JonasKop/app-java-ou1

Innehåll

1 Introduktion **1**

2 Användarhandledning **1**
2.1 Beskrivning 1
2.2 Kompilering och körning 1
2.3 Körning 1
2.4 Handledning i programmet 1
2.5 Testkörningar 2

3 Systembeskrivning **4**
3.1 MVC 4
3.1.1 Kontroller 4
3.1.2 Vy 4
3.1.3 Modell 5
3.2 Flödesbeskrivning 5
3.3 Skrivning av tester 5
3.4 UML-diagram 6

1 Introduktion

Denna rapport beskriver utförandet av den första obligatoriska uppgiften i kursen Applikationsutveckling i java (HT-18) vid Umeå Universitet. I denna uppgift skrivs ett grafiskt unittestningsprogram vid namn MyUnitTester med ett enkelt grafiskt gränssnitt som är skrivet i java med biblioteket Swing.

2 Användarhandledning

2.1 Beskrivning

Programmet består av två paneler. En panel högst upp som innehåller en testkörningsknapp, ett textfält och en rensningsknapp. Nedanför denna finns ett textutrymme där all utdata skrivs, dvs både felhantering och testresultat. När programfönstret expanderas ökar textfältet och textutrymmet i storlek medans knapparna har en satt statisk storlek. Efter testkörning skrivs resultatet ut i textutrymmet längs ner i programmet. Om utdatan är för stor för att rymmas i textutrymmet möjliggörs skrollningsfunktionen för användaren så att all data kan ses.

2.2 Kompilering och körning

För att bygga en .jar fil av detta program kan man göra på många olika byggsystem såsom Maven eller Ant. Jag har använt mig av Intellij som i sin tur använt sig av Ant. För att bygga programmet gör man följande:
Klicka på File, Project Structure. Där man skapar en artifact av en .jar fil där main klassen väljs. Sedan klickar man på Build, Build Artifacts. Sedan kan programmet köras från terminalen.

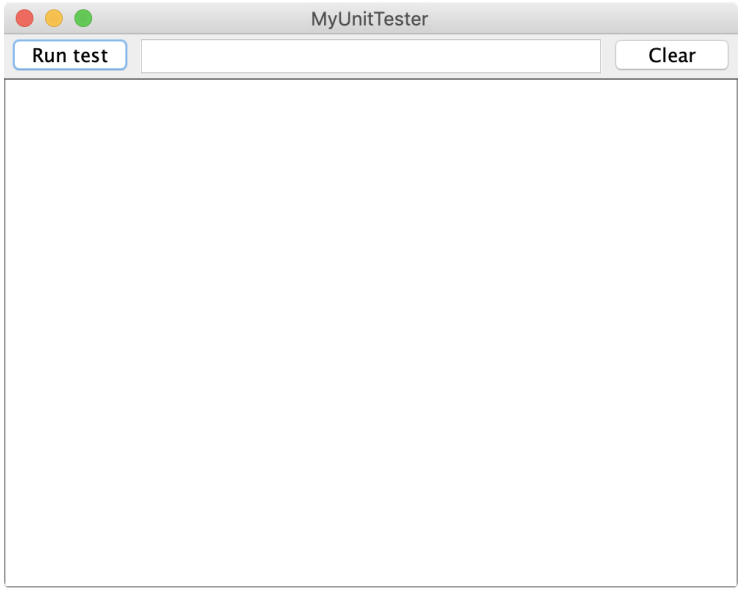
2.3 Körning

Programmet är färdigkompilerad java8 kod och komprimerat till en .jar fil. För att köra MyUnitTester körs kommandot:
`java -jar /YOUR_PATH/MyUnitTester.jar`

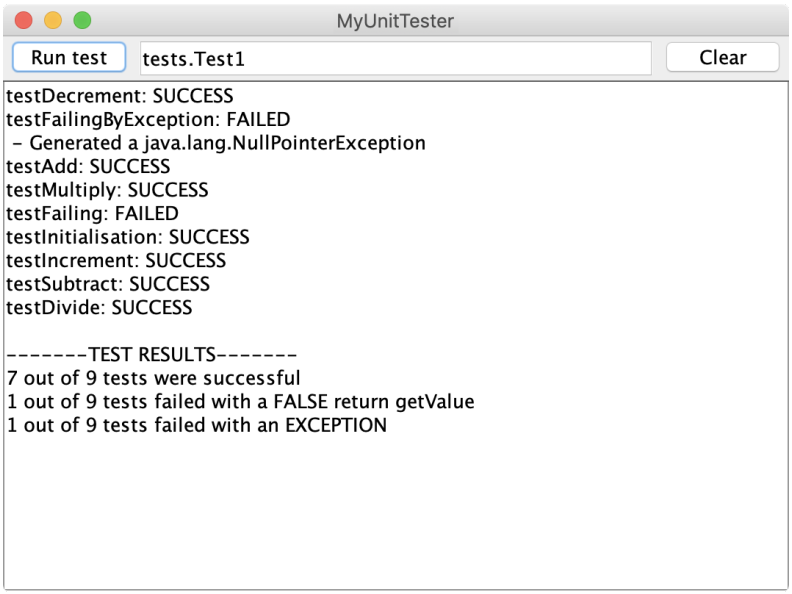
2.4 Handledning i programmet

När programmet är startat skriver användaren in sin testklass i textfältet högst upp och klickar sedan på Run test knappen för att köra programmet. All utdata inklusive fel och korrekta tester skrivs sedan ut till textutrymmet längst ner. Om testklassen är felaktig på något sätt så skrivs denna information ut i textutrymmet. För att rensa textrutan klickar användaren på Clear knappen.

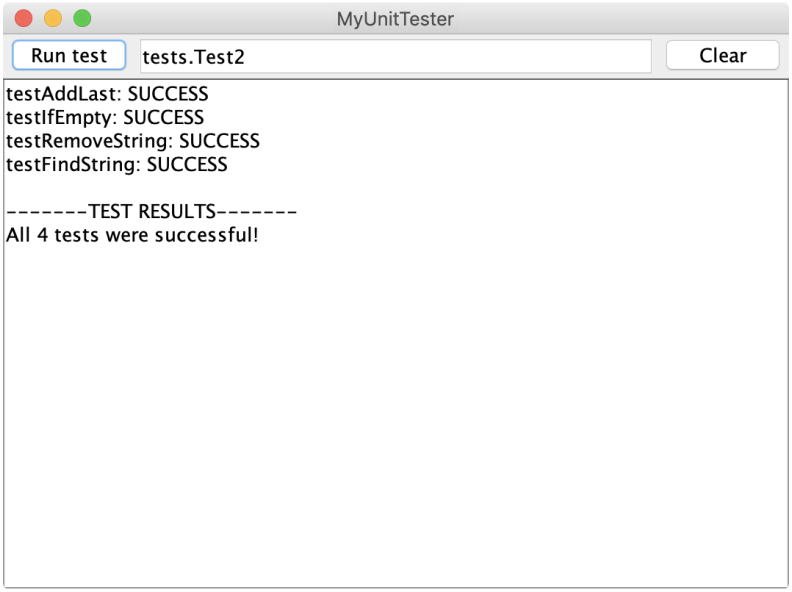
2.5 Testkörningar



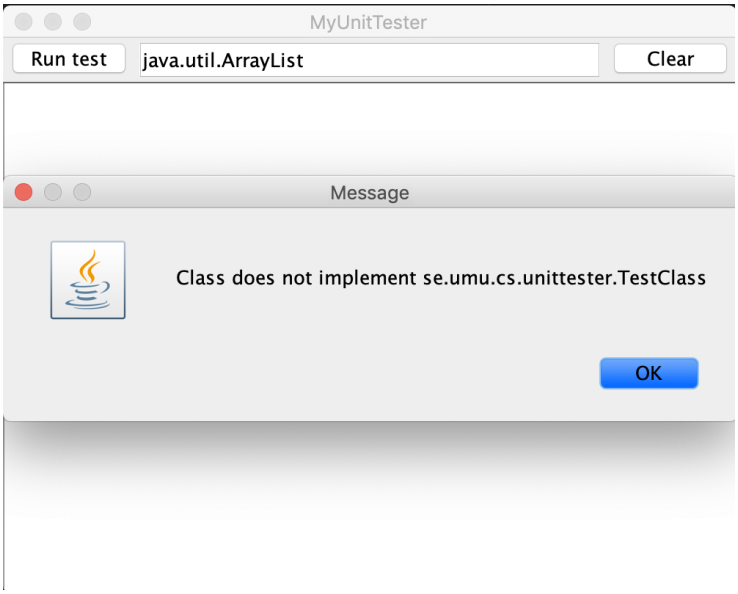
Figur 1 – Uppstart av programmet



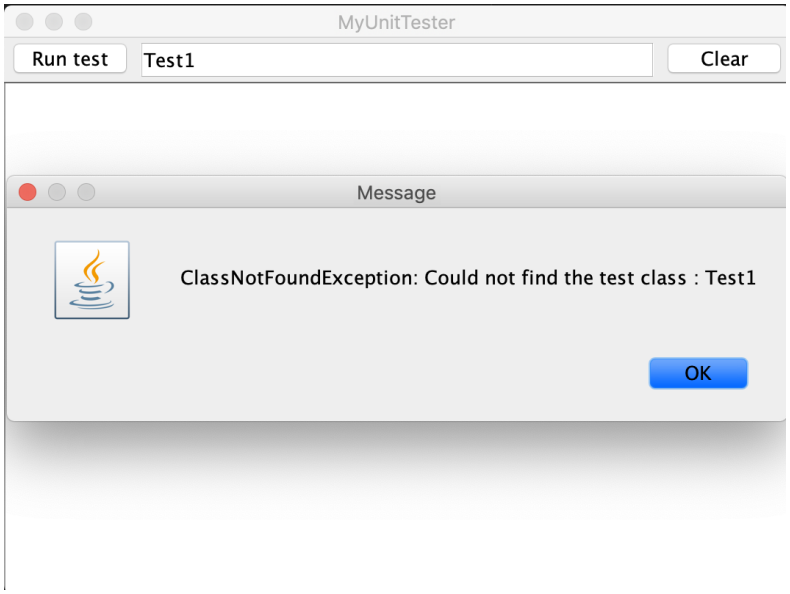
Figur 2 – Testkörning av som missyckas delvis



Figur 3 – Testkörning som lyckas



Figur 4 – Testkörning av klass som inte implementerar rätt interface



Figur 5 – Testkörning av ej existerande klass

3 Systembeskrivning

3.1 MVC

Koden använder tre huvudmoduler, vy, en kontroller och en testkörningsmodul där vyn och testkörningsmodulen kommunicerar med varandra genom kontrollern. Vyn och testkörningsmodulen är i mitt program interfaces som implementationerna måste implementera för att kunna användas i kontrollern.

3.1.1 Kontroller

Kontrollern har en väldigt enkel design och i min implementation skapar den endast vyn och sätter en action listener på kör-knappen. Action listenern på kör knappen ärver från en swing worker och vet både om testmodellen och vyn. Detta sätts via konstruktorn hos kontrollerklassen. Swing workern kör testprogrammet och sätter en callbackmetod som använder sig av vyns metod `displayString()` för att testet ska kunna skriva ut testresultatet i en `JTextArea` under körning. Om ett fel uppstår som gör att testen inte kan köras korrekt returnerar testprogrammet ett objekt, förslagsvis en sträng som gör skrivs ut i ett notifikationsfält för användaren.

3.1.2 Vy

Vyn skapas genom att skapa en klass som implementerar view interfacet. Sedan körs vyns `initialize()` metod för konfigurering av vyn och `display()` metoden för att visa fönstret. Data från vyn hämtas från `getInput()` metoden och utdata visas genom `displayString()` metoden. För att sätta en action listener för att köra tex testprogrammet används metoden`addActionListener()`. All kommunikation med vyn sker alltså genom dessa publika metoder.

3.1.3 Modell

Modellen implementerar `TestModel` interfacet och har en publik metod, metoden `run`. Denna metod tar en callback metod som argument som den kan skriva in strängar till under körning som sedan visas i vyn. Den tar även en inputsträng med inputdatan som ska köras in i detta fall, testprogrammet. Min implementation av testprogrammet skriver ut hur testerna gick under körning av testerna. Efter testerna körts klart skrivs en testsummering ut. Om ett fel uppstår under körning som gör att testet inte kan köras, tex om det är en ej korrekt testklass eller om klassen ej existerar så returneras ett felmeddelande som visas för användaren i en notifikation. Min implementation använder sig även av den statiska klassen `ExceptionHandler` för sin felhantering. Om en exception kastas i `run()` metoden skickas den hit och den genererar en sträng som innehåller exceptionnamnet en förklaring av felet och en eventuell feltext om den satts när undantaget kastades. Denna sträng returneras sedan till funktionen som kallade på den såsom `run()` i detta fall som returnerar den som felmeddelande.

3.2 Flödesbeskrivning

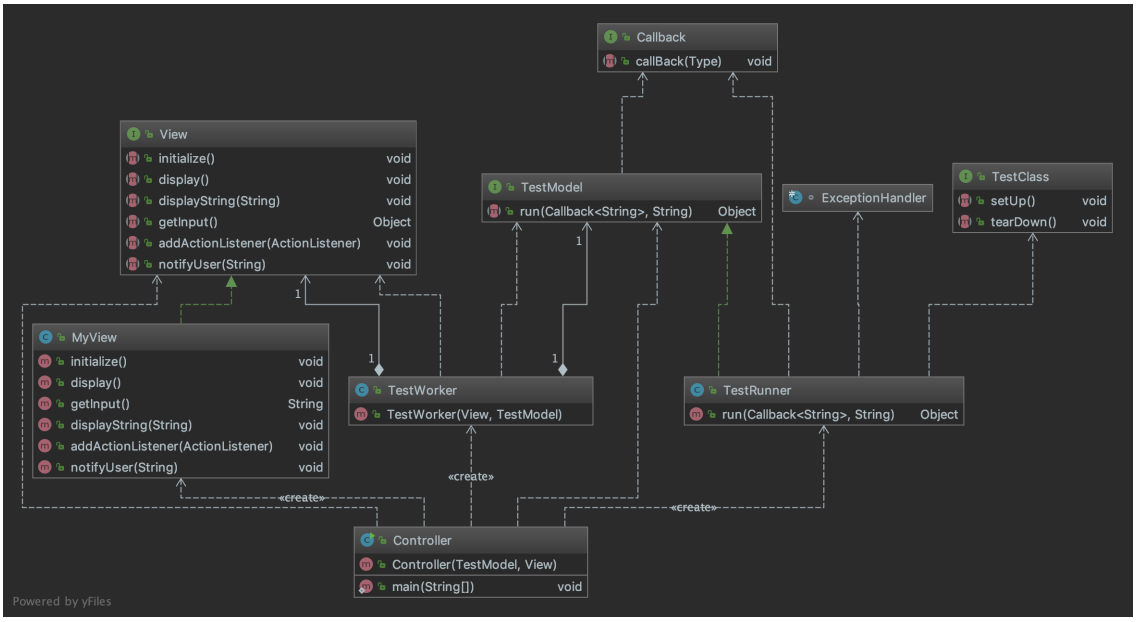
När användaren klickar på Run test-knappen kör en SwingWorker som kör testprogrammet. Den skickar in en callback metod som tar en sträng som argument som publicerar all textutdata som körs där i JTextArean i vyn. Den skickar även in en textsträng från JTextFielden i vyn som borde vara i form av sökvägen till plus testklassens namn. Tex `java.util.ArrayListTest`. Testkörningsmetoden tar sedan och läser in klassen och dess metoder. Sedan kollar den om `setUp()` och `tearDown()` finns och sparar ner dem isåfall. Sedan körs `setUp()`, testmetoden och `tearDown()` om de finns. Detta görs för alla testmetoder i klassen. För varje testmetod skrivs löpande under programmets körning ut testets resultat genom callback metoden till JTextArean. När testerna körts färdigt skrivs en summering ut av alla körda tester i callback metoden som skrivs ut till JTextArean.

3.3 Skrivning av tester

För att skriva ett test som kan köras av programmet måste testklassen implementera `se.umu.cs.unittester`. Sedan är det frivilligt att implementera `setUp()` och `tearDown()`. Om dessa finns körs först `setUp()` före varje test och `tearDown()` efter varje test. Alla testmetoder måste börja med `test*`, tillexempel `testInit` är ett giltigt testmetodnamn. Om metoderna heter någonting annat kommer de att kunna köras av programmet. Det är även viktigt att konstruktorn är tom och inte tar några argument. Själva initieringen av värdena ska ske i `setUp()` metoden.

3.4 UML-diagram

Nedan visas ett UML-diagram över programmet. Controller är huvudprogrammet där main-funktionen finns. Den skapar GUI-t på en tråd och kör sedan testerna på huvudtråden, dvs GUI:t ska inte frysa när testklasserna exekveras.



Figur 6 – UML-diagram över Programmet