UMEÅ UNIVERSITET
Institution of Datavetenskap
Laborationsrapport

10 februari 2014

Laboration 1

Applikationsutveckling i Java 7.5hp, HT13

Enhetstester

Namn Adam Dahlgren
E-mail c12amn@cs.umu.se
Sökväg ~c12amn/edu/apjava/lab1

Handledare
Johan Eliasson, Eric Jönsson, Jan-Erik Moström

Innehåll

1 Problemspecifikation **1**
1.1 Exempellösning . . . 1
1.2 Syfte . . . 1
1.3 Originalspecifikation . . . 1

2 Åtkomst och användarhandledning **2**
2.1 Kompilering och körning . . . 2
2.2 Grafiskt användargränssnitt . . . 2
2.2.1 Exempelkörning . . . 4

3 Systembeskrivning **4**
3.1 En kortare beskrivning . . . 4
3.2 Model . . . 4
3.2.1 TestResult . . . 4
3.2.2 TestStatus . . . 5
3.2.3 UnitTester . . . 5
3.3 View . . . 5
3.3.1 TesterGUI . . . 5
3.4 Controller . . . 5
3.5 MyUnitTester . . . 6

4 Begränsningar **6**
4.1 Begränsningar . . . 6
4.1.1 Undantag . . . 6
4.1.2 Val av testklasser . . . 6
4.2 Funktionalitet . . . 6
4.2.1 Feedback . . . 6
4.2.2 Hur man väljer testklass . . . 6
4.2.3 Användarinteraktion och igenkänning . . . 6

5 Reflektioner **7**

6 Testkörningar **7**
6.1 Interface . . . 7
6.1.1 Testklass . . . 7
6.1.2 Resultat . . . 7
6.2 Konstruktor . . . 8
6.2.1 Testklass . . . 8
6.2.2 Resultat . . . 8
6.3 Interface . . . 9
6.3.1 Testklass . . . 9
6.3.2 Resultat . . . 9
6.4 Felfri testklass . . . 10
6.4.1 Testklass . . . 10
6.4.2 Resultat . . . 11

7 Diskussion **11**

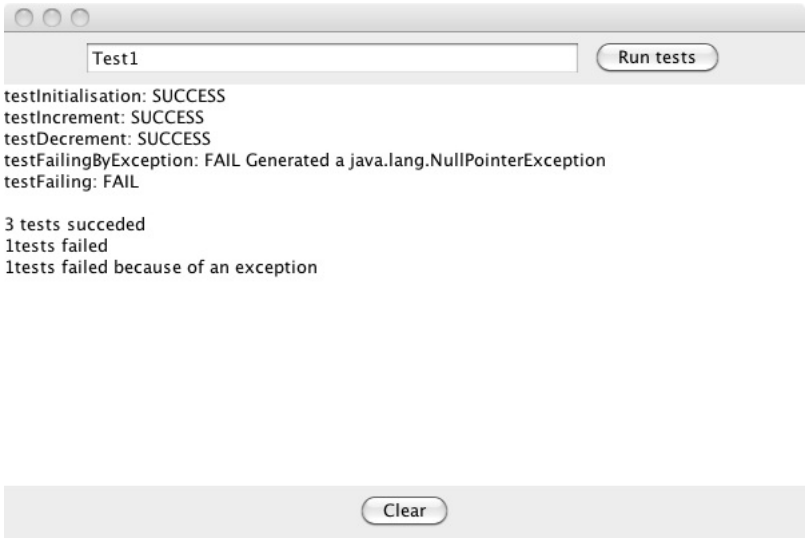
A Källkod **12**

1 Problemspecifikation

Uppgiften är att skapa ett program, *MyUnitTester*, som via ett grafiskt användargränssnitt utför tester i en specificerad testklass. Programmet ska sedan visa vilka tester som körs och resultatet. När alla tester är exekverade så skall användaren få en sammanfattning som summerar testresultaten.

När användaren har specificerat vilken testklass som ska utföras så måste programmet kontrollera att klassen är en giltig testklass. Detta uppnås genom att kontrollera att klassen implementerar ett interface *TestClass* samt att klassen har en parameterlös konstruktor. Uppfyller klassen dessa krav så ska alla metoder vars namn börjar med *test*, är parameterlösa samt returnerar en *boolean* exekveras. Finns metoderna *setUp* och *tearDown* så skall dessa exekveras före respektive efter varje testmetod. Ett test är lyckat om metoden returnerar *true*, annars misslyckat. Dessutom ska typen av eventuella undantag genererade av en testmetod synliggöras för användaren, till exempel genom utskrift.

1.1 Exempellösning



Figur 1: Exempellösning given i originalspecifikationen

1.2 Syfte

Syftet med laborationen är att bekanta sig med Java Reflections och öva på grafiska användargränssnitt. Rent kodmässigt så skall den följa Suns kodkonvention.

1.3 Originalspecifikation

Originalspecifikationen återfinns på `http://www8.cs.umu.se/kurser/5DV135/HT13/labbar/lab1/index.ht`

2 Åtkomst och användarhandledning

Programmet med källkod och denna rapport återfinns på Instutitionen för Datavetenskaps webservrar i katalogen `/home/c12/c12amn/edu/apjava/lab1`.

2.1 Kompilering och körning

Programmet startas från underkatalogen `bin` med kommandot:

```
:~/edu/apjava/lab1/bin/> java MyUnitTester
```

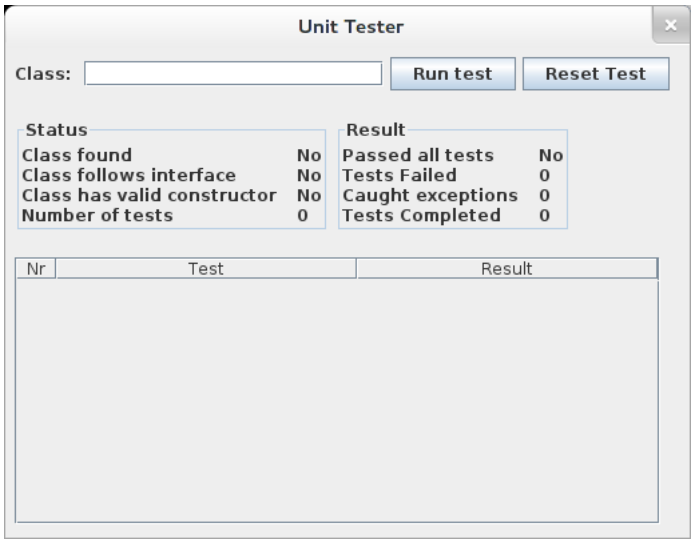
Programmet kan kompileras från underkatalogen `src` med kommandot:

```
:~/edu/apjava/lab1/src/> javac MyUnitTester.java
```

Detta gör att man från *src*-katalogen kan starta programmet med första kommandot.

2.2 Grafiskt användargränssnitt

När användaren startat programmet visas följande grafiska användargränssnitt.



Figur 2: Grafiskt användargränssnitt för lösningen

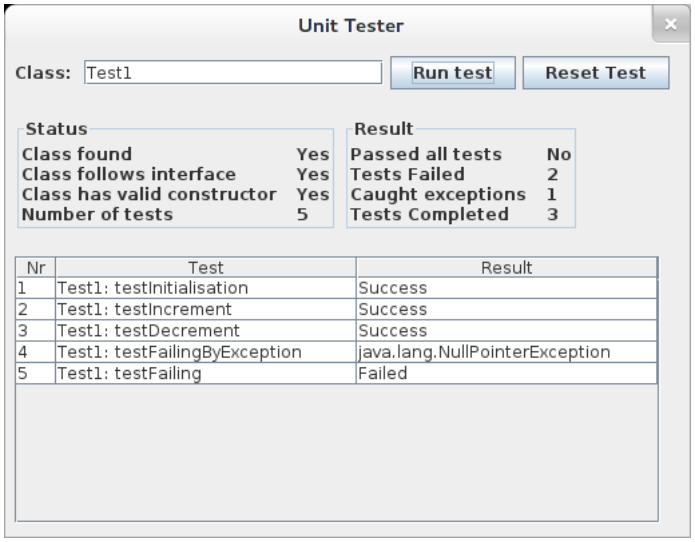
I användargränssnittet finns fyra huvudkomponenter:

- Textfältet för inmatning av testklass samt knappar för att kör tester eller tömma gränssnittet efter föregående test.
- Ett statusfält där information om den givna klassen och huruvida den finns, implementerar rätt interface och har rätt konstruktor. Även hur många test klassen innehåller finns med här.

- Resultatfältet innehåller information om en testkörning, hur många test som gick igenom respektive misslyckades och dylik information.
- En tabell över alla tester, med namn, nummer och dess resultat. Om ett test genererat ett undantag så visas detta i resultatkolumnen.

Viktigt när det gäller resultatfältet är att antalet fångade undantag enbart specificerar hur många av antalet misslyckade test som misslyckades på grund av ett undantag. Alltså är det summan av antalat lyckade test och antalet misslyckade test som motsvarar antalet test totalt.

Klassen som specificeras i textfältet av användaren måste ligga kompilerad i *lab1/bin*-katalogen. Om *Test1* är en sådan klass så anger man detta i textfältet på följande sätt:

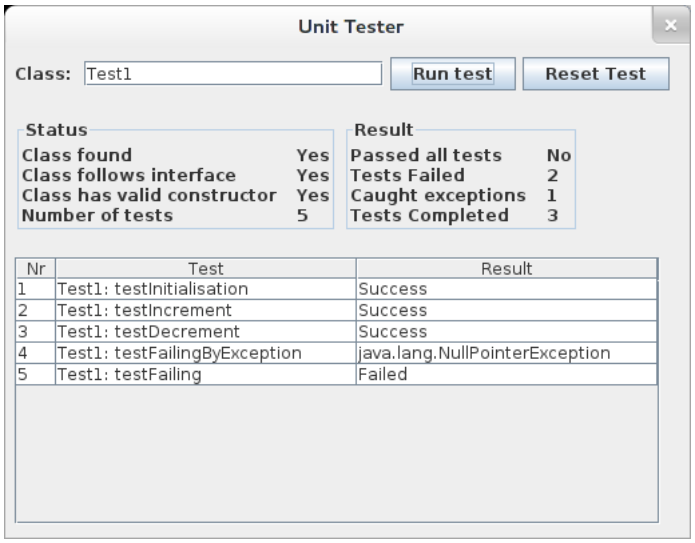


Figur 3: Inladdning av testklass

För att köra testet klickar man på **Run test** som även analyserar klassens validitet som testklass. **Reset test** återställer status- och resultatfälten samt tömmer tabellen.

2.2.1 Exempelkörning

Detta är en exempelkörning på en testklass med giltigt interface och giltig konstruktor som körts. Här demonstreras hur resultaten visas både i resultatfältet och tabellen.



Figur 4: En giltig testklass

3 Systembeskrivning

Då programmet är konstruerat efter Model-View-Controllermönstret så är den översiktliga beskrivningen mycket enkel.

3.1 En kortare beskrivning

När en användare anger en testklass i textfältet och klickar på **Run test** så ber det grafiska användargränssnittet styrenheten *mediator* att testklassen ska testas. *Mediator* vidarebefordrar detta till *UnitTester* som utför testet. För varje test som genomförts skickar *UnitTester* resultatet till tabellen idet grafiska användargränssnittet via *mediator*. När alla tester är genomförda så uppdateras både status och resultat som sedan visas i användargränssnittet.

3.2 Model

I modellpaketet återfinns två klasser som representerar en testklass status och testresultatet.

3.2.1 TestResult

Klassen är mycket enkel och innehåller attribut för antalet lyckade test, antalet misslyckade test och antalet undantag. I övrigt finns endast *setters* och *getters*.

3.2.2 TestStatus

Denna klass är även den simpel och innehåller attribut som representerar antalet test, om klassen finns och om den implementerar rätt interface samt har en giltig konstruktor.

3.2.3 UnitTester

Denna klass är själva enhetstestaren. Klassen använder sig utav en *mediator* för att kunna skicka ett testresultat till användargränssnittets tabell. Den har även en *TestStatus* och en *TestResult* som attribut.

Klassen har en huvudmetod som anropas med namnet på en testklass som parameter. Metoden använder sig sedan av *Java Reflection* för att ladda in klassen och kontrollera konstruktor- och interfacevillkoren. Om testklassen är giltig så exekverar metoden alla giltiga testmetoder och skickar resultatet till det grafiska användargränssnittets tabell. När alla testmetoder är exekverade så uppdateras både teststatusen och testresultatet som sedan förmedlas av *mediatorn* till det grafiska användargränssnittet. Metoden returnerar *true* om testningen genomfördes till fullo.

3.3 View

I viewpaketet ligger endast klassen för det grafiska användargränssnittet.

3.3.1 TesterGUI

Det grafiska användargränssnittet är uppbyggt av fyra huvudkomponenter som även nämndes i 2.2;

- Indatapanelen
- Statusfältet
- Resultatfältet
- Resultattabellen

TesterGUI är beroende av en *mediator* för att kunna skicka namnet på en testklass för testning i modellen. Status- och resultatfälten måste liksom tabellen uppdateras från *mediator*, de uppdateras ej från användargränssnittets sida.

I indatapanelen finns även en knapp för att nollställa all information i användargränssnittet. Detta påverkar ej den underliggande modellen utan är enbart en rensning i gränssnittet.

3.4 Controller

I kontroller återfinns endast styrenheten *mediator* som används för att förmedla interaktion mellan *View* och *Model*. *Mediator* innehåller tre viktiga metoder:

- **runTest** som ber en *UnitTester* utföra tester på en given klass. Är det ett lyckat test så uppdateras *View*.

- **addData** som vidarebefodrar ett resultat från *Model* till *View*.
- **updateGui** som hämtar status och resultat från en *UnitTester* och uppdaterar respektive fält i *View*.

3.5 MyUnitTester

Huvudklassen *MyUnitTester* som innehåller programmets *main*-metod sätter ihop alla delar. Den skapar en *UnitTester*, en *mediator* och en *TesterGUI*.

4 Begränsningar

Här beskrivs saknad funktionalitet och eventuella begränsningar.

4.1 Begränsningar

4.1.1 Undantag

Användaren kan ej få mer information om undantag annat än vilken typ av undantag som skett. Det borde vara möjligt att visa detta på något sätt i användargränssnittet då detta kan vara vital information för att hitta ett fel.

4.1.2 Val av testklasser

Klasserna som kan testas måste ligga i samma katalog som *MyUnitTester*.

4.2 Funktionalitet

4.2.1 Feedback

Lösningen har dålig feedback till användaren, framför allt visuellt. Först och främst borde visuell feedback ges i tabellen för att förstärka testresultat. Detta går att lösa genom att implementera en egen *TabelModel* som kan sätta färger på rader.

Det hade även varit trevligt med någon form av feedback för till exempel icke-existerande klasser och dylikt.

4.2.2 Hur man väljer testklass

Det är inte helt intuitivt hur en testklass ska väljas. Att det måste ligga en *.class*-fil i samma katalog som *MyUnitTester.class* framgår ej. Det hade varit mycket smidigare om användaren fick möjligheten att välja klass via en lista över tillgängliga klasser eller till exempel en *FileChooser*.

4.2.3 Användarinteraktion och igenkänning

Programmet har inget stöd för kortkommandon och dylikt. Till exempel bör användaren kunna trycka på tangentbordsknappen **Enter** istället för att trycka på **Run test**, så som det fungerar i de allra flesta applikationer. Lösningen är dock enkel, och det är att lämpliga *ActionListeners* läggs till som erbjuder denna funktionalitet.

5 Reflektioner

Programmet saknar funktionalitet för att till exempel ersätta Eclipse JUnit-vy. Men med lite utökning så skulle detta vara ett mycket trevligt alternativ som kan användas om Eclipse ej är tillgängligt.

Det har inte uppstått några egentliga problem annat än att strukturera interaktionen komponenter sinsemellan på ett smidigt och någorlunda logiskt sätt. MVC har varit ett mycket nyttigt verktyg som självklart kan användas på mer omfattande system.

6 Testkörningar

För en körning på en giltig testklass hänvisas läsaren till exemplet i 2.2.1.

6.1 Interface

Om en testklass inte implementerar rätt interface bör detta synas i det grafiska användargränssnittet och inga tester bör genomföras.

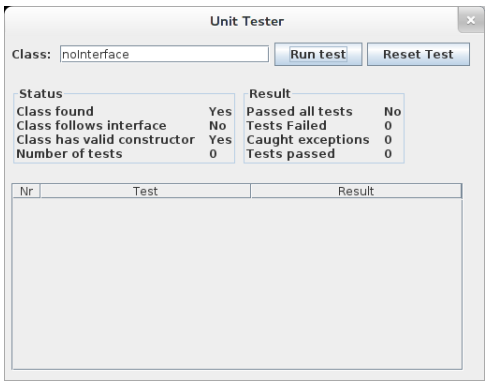
6.1.1 Testklass

Listing 1: Testklass som saknar interface

```
1  /* This testclass should pass constructor requirement
2   * but not the interface requirement */
3  /**
4   * This test class should pass constructor requirement
5   * but not the interface requirement
6   * @author c12amn
7   *
8   */
9  public class noInterface {
10
11      /**
12       * Empty constructor
13       */
14      public noInterface() {
15
16      }
17
18      /**
19       *  This test should not show or be counted
20       */
21      public boolean testAccessPrivateInteger() {
22     return true;
23      }
24 }
```

6.1.2 Resultat

Detta är ett korrekt beteende då det saknade interfacet upptäcks samt att *testAccessPrivateInteger()* inte visas.



Figur 5: Klass som ej implementerar interface

6.2 Konstruktor

Om en testklass inte har en konstruktor som ej tar några parametrar så skall detta synas i det grafiska användargränssnittet och eventuella tester ska ej genomföras.

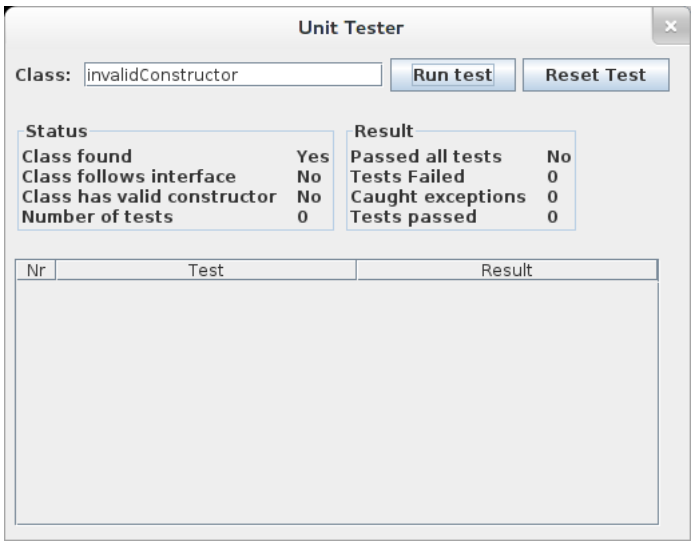
6.2.1 Testklass

Listing 2: Testklass som saknar giltig konstruktor

```
1 /* This testclass should pass interface requirement
2  * but not the constructor requirement */
3 /**
4  * This test class should pass interface requirement
5  * but not the constructor requirement
6  * @author c12amn
7  *
8  */
9 public class invalidConstructor {
10
11     /**
12      * Empty constructor.
13      * @param i
14      */
15     public invalidConstructor(Integer i) {
16
17     }
18
19     /**
20      *  This test should not show or be counted
21      */
22     public boolean testAccessPrivateInteger() {
23    return true;
24     }
25 }
```

6.2.2 Resultat

Användargränssnittet visar mycket riktigt att klassen inte har en giltig konstruktor och inga tester har körts.



Figur 6: Klass utan giltig konstruktor

6.3 Interface

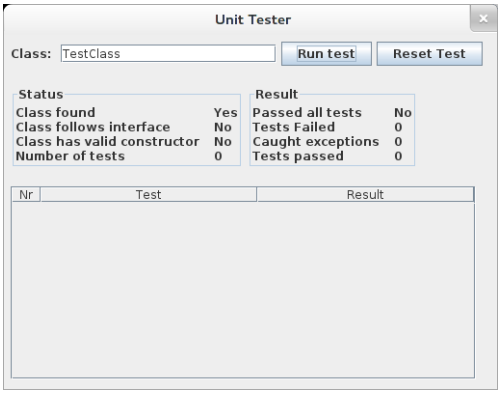
Här testas att skickas in *TestClass*-interfacet, vilket endast bör visa att klassen existerar.

6.3.1 Testklass

Listing 3: Testklass som saknar interface och giltig konstruktor

```
1
2
3 public interface TestClass {
4
5 }
```

6.3.2 Resultat



Figur 7: Interface som testklass

TestClass-interfacet ger väntat beteende hos användargränssnittet.

6.4 Felfri testklass

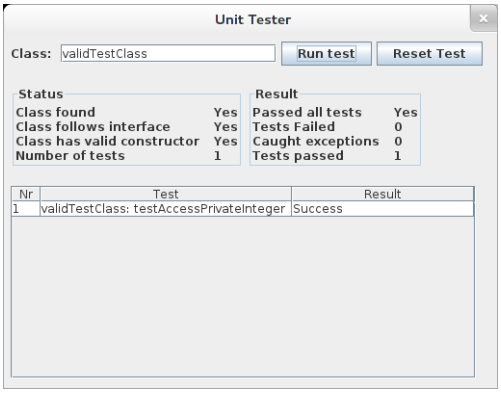
Här testas en felfri testklass. **Passed all tests** bör vara satt till **Yes** och inga test ska ha misslyckats.

6.4.1 Testklass

Listing 4: Felfri testklass

```
1  /**
2   * This test class should pass constructor requirement
3   * but not the interface requirement
4   * @author c12amn
5   *
6   */
7  public class validTestClass implements TestClass{
8
9      /**
10      * Empty constructor.
11      */
12     public validTestClass() {
13
14     }
15
16     /**
17      *  This test should not show or be counted
18      */
19     public boolean testAccessPrivateInteger() {
20    return true;
21     }
22 }
```

6.4.2 Resultat



Figur 8: Interface som testklass

Väntat beteende.

7 Diskussion

Att följa kodkonvention är otroligt viktigt för att koden ska vara läslig. Det ökar takten i vilken man kan läsa igenom kod för att felsöka eller när man återvänder till koden efter några dagar för att sätta sig in i den igen. En stor fördel är att all kod ser likadan ut var man än tittar vilket underlättar för förståelse. Dessutom måste man se till att namnge variabler och metoder på ett vettigt sätt vilket underlättar läsligheten för en själv såväl som för andra. Även om man brukar vara duktig på att namnge saker så är det bra att tvingas göra det överallt. Det är enkelt att få hjälp när kod följer en kodkonvention.

Nackdelen är väl att det tar mer tid när man ska lära sig en kodkonvention samtidigt som man börjar jobba på ett projekt. Även om man tjänar in på det mångfaldigt så är det ändå något som man måste ta hänsyn till och som kan ställa till det planeringsmässigt. I vissa fall måste man ge upp på sådana egenheter som man använder sig av i sin "egen"kodkonvention som man själv tycker om och som kanske är till stor hjälp.

A Källkod

feb 10, 14 17:11 **invalidConstructor.java** Page 1/1

```java
/* This testclass should pass interface requirement
 * but not the constructor requirement */
/**
 * This test class should pass interface requirement
 * but not the constructor requirement
 * @author c12amn
 *
 */
public class invalidConstructor {

    /**
     * Empty constructor.
     * @param i
     */
    public invalidConstructor(Integer i) {

    }

    /**
     *  This test should not show or be counted
     */
    public boolean testAccessPrivateInteger() {
        return true;
    }
}
```

feb 10, 14 16:54 **MyInt.java** Page 1/1

```java
/**
 * Class given by tutor
 * @author johane
 *
 */

public class MyInt {
        private int val;
        public MyInt() {
                val=0;
        }

        public void increment() {
                val++;
        }

        public void decrement() {
                val--;
        }

        public int value() {
                return val;
        }

}
```

feb 10, 14 16:58 **MyUnitTester.java** Page 1/1

```java
import javax.swing.SwingUtilities;
import model.unitTester.UnitTester;
import controller.Mediator;
import view.TesterGUI;

/**
 * Main class for MyUnitTester
 * @author c12amn
 *
 */
public class MyUnitTester {
    /**
     * Main method for program.
     */
    /* Must be declared out here to be accessible to the EDT */
    static TesterGUI gui;
    public static void main(String[] args) {
        /* Create mediator and tester */
        final Mediator med = new Mediator();
        UnitTester tester = new UnitTester(med);

        /* Set the tester for the mediator */
        med.setTester(tester);

        /* All GUI actions must be performed in the EDT */
        SwingUtilities.invokeLater(new Runnable() {

            @Override
            public void run() {
                /* Create, add to mediator and show GUI */
                gui = new TesterGUI(med);
                med.setGui(gui);
                gui.show();
            }
        });
    }
}
```

feb 10, 14 17:10 **noInterface.java** Page 1/1

```java
/* This testclass should pass constructor requirement
 * but not the interface requirement */
/**
 * This test class should pass constructor requirement
 * but not the interface requirement
 * @author c12amn
 *
 */
public class noInterface {

    /**
     * Empty constructor
     */
    public noInterface() {

    }

    /**
     *  This test should not show or be counted
     */
    public boolean testAccessPrivateInteger() {
        return true;
    }
}
```

```
/**
 * Given test class
 * @author johane
 *
 */
public class Test1 implements TestClass {
        private MyInt myInt;

        public Test1() {
        }

        public void setUp() {
           myInt=new MyInt();
        }

        public void tearDown() {
          myInt=null;
        }

        //Test that should succeed
        public boolean testInitialisation() {
           return myInt.value()==0;
        }

        //Test that should succeed
        public boolean testIncrement() {
           myInt.increment();
           myInt.increment();
           return myInt.value()==2;

        }

        //Test that should succeed
        public boolean testDecrement() {
                   myInt.increment();
                   myInt.decrement();
                   return myInt.value()==0;
        }

        //Test that should fail
        public boolean testFailingByException() {
                myInt=null;
                myInt.decrement();
                return true;

        }

        //Test that should fail
        public boolean testFailing() {
                return false;

        }

}
```



```
public interface TestClass {

}
```

```java
/**
 * This test class should pass constructor requirement
 * but not the interface requirement
 * @author c12amn
 *
 */
public class validTestClass implements TestClass{

    /**
     * Empty constructor.
     */
    public validTestClass() {

    }

    /**
     *  This test should not show or be counted
     */
    public boolean testAccessPrivateInteger() {
        return true;
    }
}
```



```java
package model;

/**
 * Container for a test result summary
 * @author c12amn
 *
 */
public class TestResultSummary {
    /* Variables to quantify results */
    private Integer nrOfPassed;
    private Integer nrOfFailed;
    private Integer nrOfExceptions;

    /**
     * Constructor initiating all variables to zero.
     */
    public TestResultSummary() {
        setNrOfPassed(0);
        setNrOfFailed(0);
        setNrOfExceptions(0);
    }

    /**
     * Getter for number of passed tests
     * @return nr of passed tests
     */
    public Integer getNrOfPassed() {
        return nrOfPassed;
    }

    /**
     * Setter for number of completed test
     * @param i
     */
    public void setNrOfPassed(Integer i) {
        if(i != null) {
            nrOfPassed = i;
        }
    }

    /**
     * Getter for number of failed tests
     * @return nr of failed tests
     */
    public Integer getNrOfFailed() {
        return nrOfFailed;
    }

    /**
     * Setter for number of failed tests
     * @param i
     */
    public void setNrOfFailed(Integer i) {
        if(i != null) {
            nrOfFailed = i;
        }
    }

    /**
     * Setter for number of exceptions
     * @param i
     */
```

```
    public void setNrOfExceptions(Integer i) {
        if(i != null) {
            nrOfExceptions = i;
        }
    }

    /**
     * Getter for number of caught exceptions
     * @return nr of exceptions
     */
    public Integer getNrOfExceptions() {
        return nrOfExceptions;
    }

    /**
     * Checks if all tests passed.
     * @return true if all tests passed, false if not or if there are no tests
     */
    public Boolean getPassedAllTests() {
        return nrOfFailed == 0 && nrOfPassed != 0;
    }
}
```



```
package model;

/**
 * Container class to hold information about the status of a tests.
 * @author c12amn
 *
 */
public class TestStatus {
    /* The status variables */
    private Integer nrOfTests;
    private Boolean classFound;
    private Boolean hasValidConstructor;
    private Boolean followsInterface;

    /**
     * Constructor for the tests status.
     */
    public TestStatus() {
        nrOfTests = 0;
        classFound = false;
        hasValidConstructor = false;
        followsInterface = false;
    }

    /**
     * Getter for the number of tests
     * @return
     */
    public Integer getNrOfTests() {
        return nrOfTests;
    }

    /**
     *
     * @return true if the class is found, else false
     */
    public Boolean classFound() {
        return classFound;
    }

    /**
     * Checks if the class did have a valid constructor
     * @return true if the class implements the correct interface
     */
    public Boolean hasValidConstructor() {
        return hasValidConstructor;
    }

    /**
     * Checks if the class did follow interface
     * @return
     */
    public Boolean followsInterface() {
        return followsInterface;
    }

    /**
     * Setter for the number of tests
     * @param i
     */
    public void setNrOfTests(Integer i) {
        if(i != null) {
```

```
            nrOfTests = i;
        }
    }

    /**
     * Setter for if the class found or not
     * @param b
     */
    public void setClassFound(Boolean b) {
        if(b != null) {
            classFound = b;
        }
    }

    /**
     * Setter for if the class has valid constructor
     * @param b
     */
    public void setHasValidConstructor(Boolean b) {
        if(b != null) {
            hasValidConstructor = b;
        }
    }

    /**
     * Setter for if the class follows the interface.
     * @param b
     */
    public void setFollowsInterface(Boolean b) {
        if(b != null) {
            followsInterface = b;
        }
    }

}
```



```
package model.unitTester;

import java.lang.reflect.Constructor;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import model.*;
import controller.Mediator;

/**
 * A unit testing class that uses Java Reflection to load a test class and
 * execute its test methods.
 *
 * A test method is defined as a method with no parameters, returning a boolean
 * and with a name starting with "test".
 *
 * @author c12amn
 *
 */
public class UnitTester {

    public static final String TEST_INTERFACE = "TestClass";

    /* The mediator is used to send test results to the GUI */
    private Mediator mediator;

    /* These two saves the status and summary of a unit test. */
    private TestStatus status;
    private TestResultSummary result;

    /**
     * Only constructor.
     *
     * @param med
     *            A mediator to communicate with the GUI
     */
    public UnitTester(Mediator med) {
        mediator = med;
        status = new TestStatus();
        result = new TestResultSummary();
    }

    /**
     * This method performs a unit test on a given test class. It checks if the
     * class is a valid test and then runs its test methods.
     *
     * A valid test class implements the interface TestClass and has a
     * constructor with no parameters.
     *
     * @param testClass
     * @return true if the unit test was successful, else false.
     * @see TestClass
     */
    public boolean performUnitTest(String testClass) {
        /* The result string and name of test sent to the GUI table */
        String result;
        String testName;

        /* Keeping track of the result of a test */
        boolean passed;

        /* Variables to indicate failed setup/teardown */
        boolean setUpFailed, tearDownFailed;
```

```
        /* Values for the validity of a test class */
        boolean classFound = true;
        boolean hasValidConstr = false;
        boolean followsInterface = false;

        /* Summary variables for the whole test */
        int nrOfTests = 0;
        int nrOfTestsPassed = 0;
        int nrOfFailed = 0;
        int nrOfExceptions = 0;

        /* Try to load the test class */
        Class<?> testclass = null;
        try {
            testclass = Class.forName(testClass);
        } catch (final ClassNotFoundException e) {
            /* The class was not found */
            classFound = false;
        }

        /* Check interface and constructor if class is found */
        if (classFound) {
            followsInterface = implementsInterface(testclass);
            hasValidConstr = checkConstructor(testclass);
        }

        /*
         * If the test class is valid
         */
        if (followsInterface && hasValidConstr && classFound) {

            /* Get the setUp/tearDown methods */
            Method setUp = getSetUp(testclass);
            Method tearDown = getTearDown(testclass);

            /* Create instance of test class */
            Object testObject = null;
            try {
                testObject = testclass.newInstance();
            } catch (InstantiationException e1) {
                /* Could not instantiate, should fail test */
                return false;
            } catch (IllegalAccessException e1) {
                /* Could not access constructor , should fail test */
                return false;
            }

            /* Loop through all methods */
            for (Method method : testclass.getMethods()) {
                /* If the method is a test method */
                if (method.getName().startsWith("test")
                        && method.getReturnType() == boolean.class
                        && method.getParameterTypes().length == 0) {

                    /* Another test found */
                    nrOfTests++;
                    /* Save the name of the test to send to the GUI */
                    testName = testClass + ": " + method.getName();

                    /* Default values */
                    result = "Failed";
```



```
                    passed = true;
                    setUpFailed = true;
                    tearDownFailed = true;

                    /* If there is a setUp method */
                    if (setUp != null) {
                        try {
                            /* Execute the setup method */
                            setUp.invoke(testObject, (Object[]) null);
                            /* Method succeeded */
                            setUpFailed = false;

                            /* All exceptions are reported back to the GUI */
                        } catch (IllegalArgumentException e) {
                            result = "SetUp: " + e.getCause().toString();
                        } catch (IllegalAccessException e) {
                            result = "SetUp: " + e.getCause().toString();
                        } catch (InvocationTargetException e) {
                            result = "SetUp: " + e.getCause().toString();
                        }
                    }

                    if (!setUpFailed) {
                        /* Execute the test method and save the result */
                        try {
                            passed = (Boolean) method.invoke(testObject,
                                    (Object[]) null);

                        } catch (IllegalArgumentException e) {
                            result = e.getCause().toString();
                            passed = false;
                        } catch (IllegalAccessException e) {
                            result = e.getCause().toString();
                            passed = false;
                        } catch (InvocationTargetException e) {
                            /* Save exception from test method */
                            result = e.getCause().toString();
                            /* Test failed */
                            passed = false;
                            nrOfExceptions++;
                        }

                        if (tearDown != null) {
                            /* Execute teardown method if available */
                            try {
                                tearDown.invoke(testObject, (Object[]) null);
                                tearDownFailed = false;
                                /*
                                 * Test is considered failed if teardown failed
                                 */
                                passed = false;

                                /* Save the exception type */
                            } catch (IllegalArgumentException e) {
                                result = "TearDown: " + e.getCause().toString();
                            } catch (IllegalAccessException e) {
                                result = "TearDown: " + e.getCause().toString();
                            } catch (InvocationTargetException e) {
                                result = "TearDown: " + e.getCause().toString();
                            }
                        }
```

```
                    }

                    if (passed && !tearDownFailed) {
                        /* Report success and count */
                        result = "Success";
                        nrOfTestsPassed++;
                    } else {
                        /* Default result is "Failed", only count */
                        nrOfFailed++;
                    }

                    /* Send data to GUI table. */
                    mediator.addData(testName, result);
                }

            }
        }

        /* Update the status */
        status.setClassFound(classFound);
        status.setFollowsInterface(followsInterface);
        status.setHasValidConstructor(hasValidConstr);
        status.setNrOfTests(nrOfTests);

        /* Update the result summary */
        this.result.setNrOfPassed(nrOfTestsPassed);
        this.result.setNrOfFailed(nrOfFailed);
        this.result.setNrOfExceptions(nrOfExceptions);

        return true;
    }

    /**
     * Loads the method called setup for the given class
     *
     * @param testclass
     * @return A method if the method setUp() exists, else null.
     */
    public static Method getSetUp(Class<?> testclass) {
        Method setUp;
        /* Try to get method called setUp with no parameters */
        try {
            setUp = testclass.getMethod("setUp", (Class<?>[]) null);
        } catch (NoSuchMethodException e) {
            /* No method was found, should return null instead */
            setUp = null;
        }
        return setUp;

    }

    /**
     * Loads the method called teardown for the given class
     *
     * @param testclass
     * @return A method if the method tearDown() exists, else null.
     */
    public static Method getTearDown(Class<?> testclass) {
        Method tearDown;
        /* Try get the method called tearDown with no parameters */
        try {
            tearDown = testclass.getMethod("tearDown", (Class<?>[]) null);
```



```
        } catch (NoSuchMethodException e) {
            /* No method found, should return null instead */
            tearDown = null;
        }
        return tearDown;

    }

    /**
     * Searches a class for a constructor with no parameters
     *
     * @param testclass
     * @return true if a constructor with no parameters exist, else false.
     */
    public static boolean checkConstructor(Class<?> testclass) {
        /* Iterate over all constructors of the class */
        for (Constructor<?> con : testclass.getDeclaredConstructors()) {
            if (con.getParameterTypes().length == 0) {
                return true; /* Found constructor with no parameters */
            }
        }
        return false; /* No valid constructor found */
    }

    /**
     * Checks if the given class implements the correct test class interface.
     *
     * @param testclass
     * @return true if the class implements the interface, else false.
     */
    public static boolean implementsInterface(Class<?> testclass) {
        /* Iterate over all interfaces implemented by class */
        for (Class<?> c : testclass.getInterfaces()) {
            if (c.getName().equals(TEST_INTERFACE)) {
                return true; /* Correct interface found */
            }
        }
        return false; /* Does not implement correct interface */
    }

    /**
     * Getter for test status
     * @return status
     * @see TestStatus
     */
    public TestStatus getStatus() {
        return status;
    }

    /**
     * Getter for test result summary
     * @return summary
     * @see TestResultSummary
     */
    public TestResultSummary getResult() {
        return result;
    }

}
```

```
package view;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;

import controller.Mediator;

import java.awt.*;
import java.awt.event.*;

public class TesterGUI {
    private JFrame mainFrame;

    private JPanel inputField;
    private JPanel statusField;
    private JPanel resultField;
    private JPanel testResults;

    private JLabel nrOfTests;
    private JLabel classExists;
    private JLabel classHasInterface;
    private JLabel classCorrectConstr;


    private JLabel nrOfCompletedTests;
    private JLabel nrOfFailedTests;
    private JLabel nrOfCaughtException;
    private JLabel passedTest;

    private JButton runButton;
    private JButton resetButton;

    private JTable resultTable;


    private JTextField testClassName;

    private Mediator mediator;

    public TesterGUI(Mediator med){
        mediator = med;
        mainFrame = new JFrame("Unit Tester");
        mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        mainFrame.setLayout(new GridBagLayout());
        mainFrame.setResizable(false);

        GridBagConstraints constraints = new GridBagConstraints();
        constraints.insets = new Insets(2, 2, 2, 2);

        buildInputField();

        constraints.gridy = 1;
        constraints.gridx = 1;
        constraints.anchor = GridBagConstraints.NORTHWEST;
        constraints.weightx = 1;
        constraints.weighty = 1;
        mainFrame.add(inputField, constraints);

        buildStatusField();
        buildResultField();

        JPanel statsPane = new JPanel();
```



```
        statsPane.setLayout(new FlowLayout(FlowLayout.LEFT));

        statsPane.add(statusField);
        statsPane.add(resultField);


        constraints.gridy = 2;
        constraints.anchor = GridBagConstraints.WEST;
        mainFrame.add(statsPane, constraints);


        buildTestResultField();

        constraints.gridy = 3;

        mainFrame.add(testResults, constraints);

        clear();

        mainFrame.pack();
        mainFrame.setMinimumSize(new Dimension(506, 400));


    }

    public void show(){
        mainFrame.setVisible(true);
    }


    private void buildInputField(){
        inputField = new JPanel();
        testClassName = new JTextField(20);
        runButton = new JButton("Run test");
        resetButton = new JButton("Reset Test");

        runButton.addActionListener(new ActionListener() {

            @Override
            public void actionPerformed(ActionEvent arg0) {
                // TODO Auto-generated method stub
                if(!testClassName.getText().isEmpty()){
                    clear();
                    mediator.runTest(testClassName.getText());
                }
            }
        });

        resetButton.addActionListener(new ActionListener() {

            @Override
            public void actionPerformed(ActionEvent e) {
                clear();

            }
        });

        inputField.add(new JLabel("Class: "), BorderLayout.WEST);
        inputField.add(testClassName, BorderLayout.EAST);
        inputField.add(runButton, BorderLayout.LINE_END);
        inputField.add(resetButton, BorderLayout.LINE_END);
```

```
    }

    private void buildStatusField(){
        final JLabel tests = new JLabel("Number of tests");
        final JLabel foundClass = new JLabel("Class found");
        final JLabel followsInterface = new JLabel("Class follows interface");
        final JLabel hasConstr = new JLabel("Class has valid constructor");

        nrOfTests = new JLabel();
        classExists = new JLabel();
        classHasInterface = new JLabel();
        classCorrectConstr = new JLabel();

        statusField = new JPanel();
        statusField.setLayout(new BoxLayout(statusField, BoxLayout.X_AXIS));
        statusField.setBorder(BorderFactory.createTitledBorder("Status"));

        JPanel firstColumn = new JPanel();
        JPanel secondColumn = new JPanel();

        firstColumn.setLayout(new BoxLayout(firstColumn, BoxLayout.Y_AXIS));
        secondColumn.setLayout(new BoxLayout(secondColumn, BoxLayout.Y_AXIS));


        firstColumn.add(foundClass);
        secondColumn.add(classExists);

        firstColumn.add(followsInterface);
        secondColumn.add(classHasInterface);

        firstColumn.add(hasConstr);
        secondColumn.add(classCorrectConstr);

        firstColumn.add(tests);
        secondColumn.add(nrOfTests);

        statusField.add(firstColumn);
        statusField.add(Box.createRigidArea(new Dimension(15, 0)));
        statusField.add(secondColumn);

    }

    private void buildResultField(){
        final JLabel testsCompleted = new JLabel("Tests passed");
        final JLabel testsFailed = new JLabel("Tests Failed");
        final JLabel exceptionFailed = new JLabel("Caught exceptions");
        final JLabel testResult = new JLabel("Passed all tests");

        nrOfCompletedTests = new JLabel();
        nrOfFailedTests = new JLabel();
        nrOfCaughtException = new JLabel();
        passedTest = new JLabel();

        resultField = new JPanel();
        resultField.setLayout(new BoxLayout(resultField, BoxLayout.X_AXIS));
        resultField.setBorder(BorderFactory.createTitledBorder("Result"));

        JPanel firstColumn = new JPanel();
        JPanel secondColumn = new JPanel();

        firstColumn.setLayout(new BoxLayout(firstColumn, BoxLayout.Y_AXIS));
        secondColumn.setLayout(new BoxLayout(secondColumn, BoxLayout.Y_AXIS));
```



```
        firstColumn.add(testResult);
        secondColumn.add(passedTest);

        firstColumn.add(testsFailed);
        secondColumn.add(nrOfFailedTests);

        firstColumn.add(exceptionFailed);
        secondColumn.add(nrOfCaughtException);

        firstColumn.add(testsCompleted);
        secondColumn.add(nrOfCompletedTests);



        resultField.add(firstColumn);
        resultField.add(Box.createRigidArea(new Dimension(15, 0)));
        resultField.add(secondColumn);

    }

    private void buildTestResultField(){
        testResults = new JPanel();

        DefaultTableModel mod = new DefaultTableModel(){

            /* Default */
            private static final long serialVersionUID = 1L;

            @Override
            public boolean isCellEditable(int row, int column) {
                return false;
            }

        };

        mod.addColumn("Nr");
        mod.addColumn("Test");
        mod.addColumn("Result");

        resultTable = new JTable(mod);
        resultTable.setName("Test Results");

        /* A more appropriate size for the test index column */
        resultTable.getColumnModel().getColumn(0).setMaxWidth(30);

        /* No need to have movable columns */
        resultTable.getTableHeader().setReorderingAllowed(false);


        JScrollPane resultPane = new JScrollPane(resultTable);
        resultPane.setPreferredSize(new Dimension(483, 200));

        testResults.add(resultPane, BorderLayout.SOUTH);

    }

    public void addData(String testName, String result){
        DefaultTableModel mod = (DefaultTableModel) resultTable.getModel();
        mod.addRow(new Object[]{resultTable.getRowCount()+1, testName, result});
    }
    public void clear(){
```

```
        ((DefaultTableModel) resultTable.getModel()).setRowCount(0);

        nrOfTests.setText("-");
        classExists.setText("-");
        classHasInterface.setText("-");
        classCorrectConstr.setText("-");

        nrOfCompletedTests.setText("-");
        nrOfFailedTests.setText("-");
        nrOfCaughtException.setText("-");
        passedTest.setText("-");
    }

    private void setNrOfTests(int nrOfTests){
        this.nrOfTests.setText(Integer.toString(nrOfTests));
    }

    private void setClassExists(boolean exists){
        if(exists){
            this.classExists.setText("Yes");
        } else {
            this.classExists.setText("No");
        }
    }

    private void setClassFollowsInterface(boolean follows){
        if(follows){
            this.classHasInterface.setText("Yes");
        } else {
            this.classHasInterface.setText("No");
        }
    }
    private void setClassHasValidConstructor(boolean valid){
        if(valid){
            this.classCorrectConstr.setText("Yes");
        } else {
            this.classCorrectConstr.setText("No");
        }
    }

    public void updateStatusField(int nrOfTests, boolean classExists,
            boolean valid, boolean followsInterface){
        setNrOfTests(nrOfTests);
        setClassExists(classExists);
        setClassFollowsInterface(followsInterface);
        setClassHasValidConstructor(valid);

    }

    private void setNrOfCompletedTests(int nrOfCompleted){
        this.nrOfCompletedTests.setText(Integer.toString(nrOfCompleted));
    }

    private void setNrOfFailed(int nrOfFailed){
        this.nrOfFailedTests.setText(Integer.toString(nrOfFailed));
    }
    private void setNrOfExceptionsCaught(int nrOfExceptions){
        this.nrOfCaughtException.setText(Integer.toString(nrOfExceptions));
    }

    private void setPassedTest(boolean passed){
        if(passed){
```



```
            this.passedTest.setText("Yes");
        } else {
            this.passedTest.setText("No");
        }
    }

    public void updateResultField(int nrOfCompleted, int nrOfFailed,
            int nrOfExceptions, boolean passed){
        setNrOfCompletedTests(nrOfCompleted);
        setNrOfFailed(nrOfFailed);
        setNrOfExceptionsCaught(nrOfExceptions);
        setPassedTest(passed);

    }

}
```

```java
package controller;

import javax.swing.SwingUtilities;

import model.TestResultSummary;
import model.TestStatus;
import model.unitTester.UnitTester;
import view.TesterGUI;

/**
 * A controller class to mediate interaction between
 * the TesterGui and the model.
 * @author c12amn
 *
 */
public class Mediator {
    /* Communicators */
    private TesterGUI gui;
    private UnitTester tester;

    /**
     * Only constructor for mediator. Initiates fields to null.
     * The GUI and the Tester MUST be set via setters.
     */
    public Mediator() {
        /* Must be set with setters */
        gui = null;
        tester = null;
    }

    /**
     * Runs test for given test class.
     * @param testClass
     */
    public void runTest(String testClass) {
        /* Perform all test in test class (if it is valid) */
        if(tester.performUnitTest(testClass)) {
            /* A successful test means new data for the GUI */
            updateGui();
        }
    }

    /**
     * Adds the given test result and test name to the GUI
     * @param testName
     * @param test result
     */
    public void addData(final String testName, final String testResult) {
        /* This must be execute by the EDT */
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                /* Send data */
                gui.addData(testName, testResult);
            }
        });
    }

    /**
     * Updates the GUI with the current test status and result summary.
     */
    private void updateGui() {
```



```java
        /* Get the current test status and result summary */
        final TestStatus s = tester.getStatus();
        final TestResultSummary r = tester.getResult();

        /* Must be executed by the EDT */
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                /* Update the status field of the GUI */
                gui.updateStatusField(s.getNrOfTests(), s.classFound(),
                        s.hasValidConstructor(), s.followsInterface());

                /* Update the result summary field of the GUI */
                gui.updateResultField(r.getNrOfPassed(), r.getNrOfFailed(),
                        r.getNrOfExceptions(), r.getPassedAllTests());
            }
        });

    }

    /**
     * Setter for Unit Tester
     * @param tester
     */
    public void setTester(UnitTester t) {
        tester = t;

    }

    /**
     * Setter for Tester GUI
     * @param gui
     */
    public void setGui(TesterGUI g) {
        gui = g;
    }
}
```