Department of Electrical and Computer Engineering and Computer Science
Milwaukee School of Engineering
Spring Quarter 2016-2017
SE-2832: Introduction to Software Verification

Lab 3: Code Coverage Analysis using EMMA

1. Objectives

- Construct test requirements from testing documentation.
- Understand the importance of Code coverage in assessing testing of software deliverables
- Use a code coverage tool to determine untested segments of source code.
- Measure code coverage in Java using the ECL-Emma tool in Eclipse.
- Construct new TestNG test cases to increase code coverage.
- Use models to generate test cases for a project.

2. Introduction

One of the hallmarks used for software testing is the concept of code coverage. While the exact relationship is often elusive, research has generally shown that the greater the code coverage during testing, the greater the reliability of the deployed software will be once the system is completed. There has been significant study of the relationship between code coverage and the resulting reliability of the source code. Garg and Del Frate indicate that there is a strong correlation between code coverage obtained during testing and software reliability, especially in larger programs. The exact extent of this relationship, however, is unknown, but the general trend is consistent, though highly non-linear, as is shown in Figure 1.

Marick cites some of the misuses for code coverage metrics. A certain level of code coverage is often mandated by the software development process when evaluating the effectiveness of the testing phase. This level is often varied.

Piwowarski, Ohba, and Caruso indicate that 70% statement coverage is necessary to ensure sufficient test case coverage, 50% statement coverage is insufficient to exercise the module, and beyond 70%-80% is not cost effective. Hutchins indicates that even 100% coverage is not necessarily a good indication of testing adequacy, for though more faults are discovered at 100% coverage than 90% or 95% coverage, faults can still be uncovered even if testing has reached 100% coverage. These recommendations, however, are quite old and reflect trying to do code coverage from a acceptance testing approach, not a unit testing approach.

Extreme Programming advocates and agile programming methods tend to endorse 100% method coverage in order to ensure that all methods are invoked at least once, though there are also exceptions given for small functions that are smaller than the test cases would be. Method coverage, however, is a very weak coverage measure. Stronger coverage methods include statement coverage and branch coverage.

There are two approaches to code coverage analysis. In the first case, the actual binary code is instrumented with calls to a routine that captures the method invocations. In the second case, a virtual machine executes the source code, and as part of the machine, execution logs are captured. In either case, there is a performance penalty associated with capturing the execution profile.

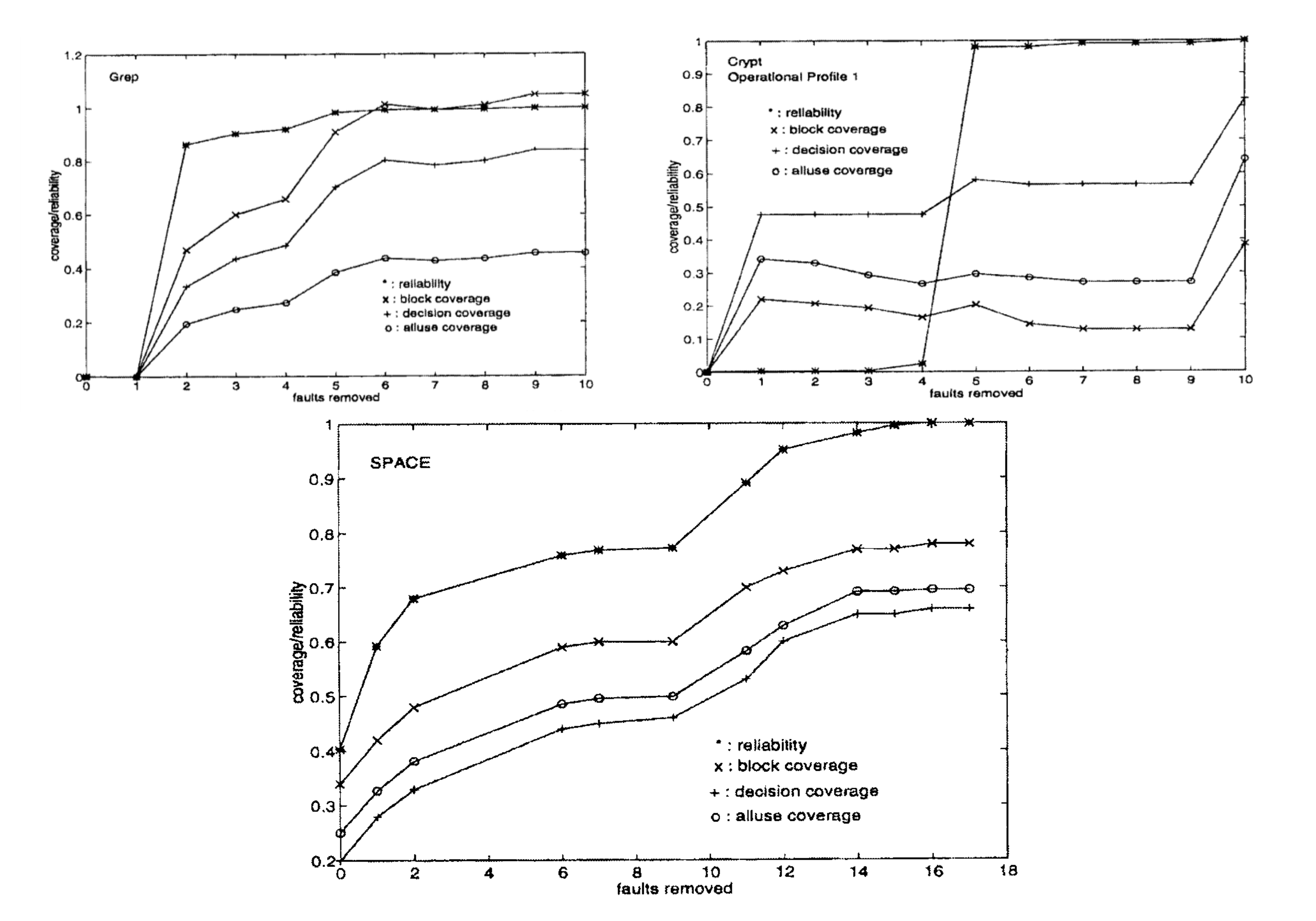
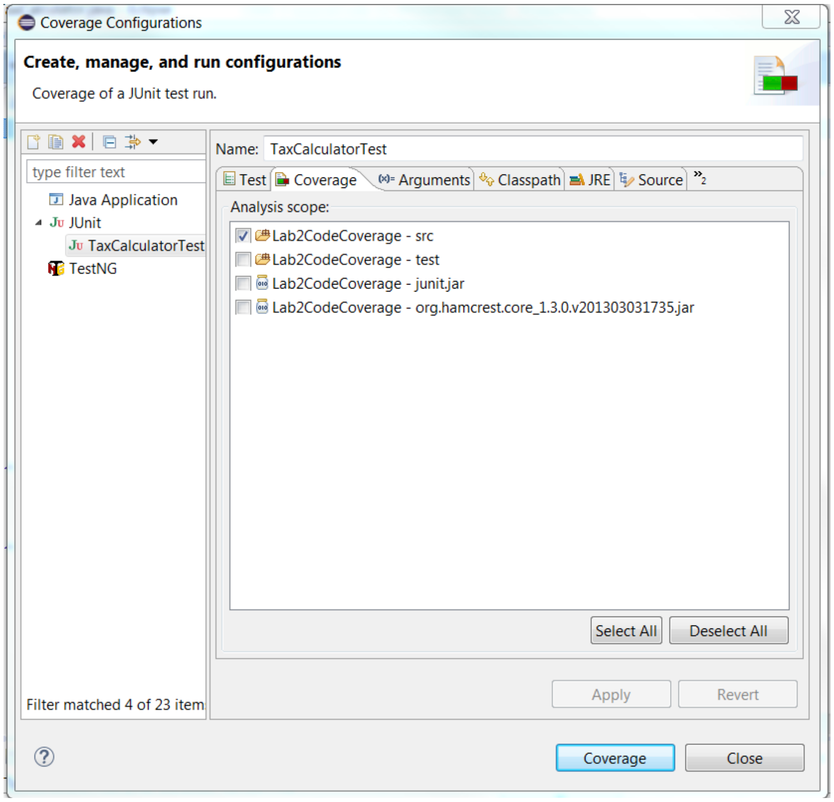


Figure 1 The relationship between software reliability and code coverage.

We will be using the EclEmma plug-in for Eclipse which uses the JaCoCo code coverage tool to collect metrics about the code coverage obtained when you execute your program. The JaCoCo tool is an open-source code coverage analysis tool for Java. It does not require instrumentation of the source code, and is highly effective at performing a code coverage analysis.

3. Procedure

1. Start by installing TestNG into your Eclipse environment. Instructions for doing this are provided on the following website: http://testng.org/doc/download.html Instructions are about halfway down the page.
2. Once you have installed TestNG, import the project which is provided to you as a zip file. This project contains test cases for the Tax Calculator. Run them and make sure they execute properly. No tests should fail based on the tests executing. Only when this is true should you move onto the next step.
3. Install the EclEmma tool in Eclipse using the following instructions:[1]
   a. From your Eclipse menu select Help → Install New Software
   b. Click add for the site.
   c. Enter EclEmma as the name and enter http://update.eclemma.org/ for the site URL:
   d. Press Finish.
   e. Follow the steps in the installation wizard.
4. Select the button and open up the "coverage configurations" menu. Modify the analysis scope entry so that coverage metrics are only collected for the src code, not the test harness.



Coverage Configurations

Create, manage, and run configurations

Coverage of a JUnit test run.

type filter text

- Java Application
- JUnit
  - TaxCalculatorTest
- TestNG

Filter matched 4 of 23 item

Name: TaxCalculatorTest

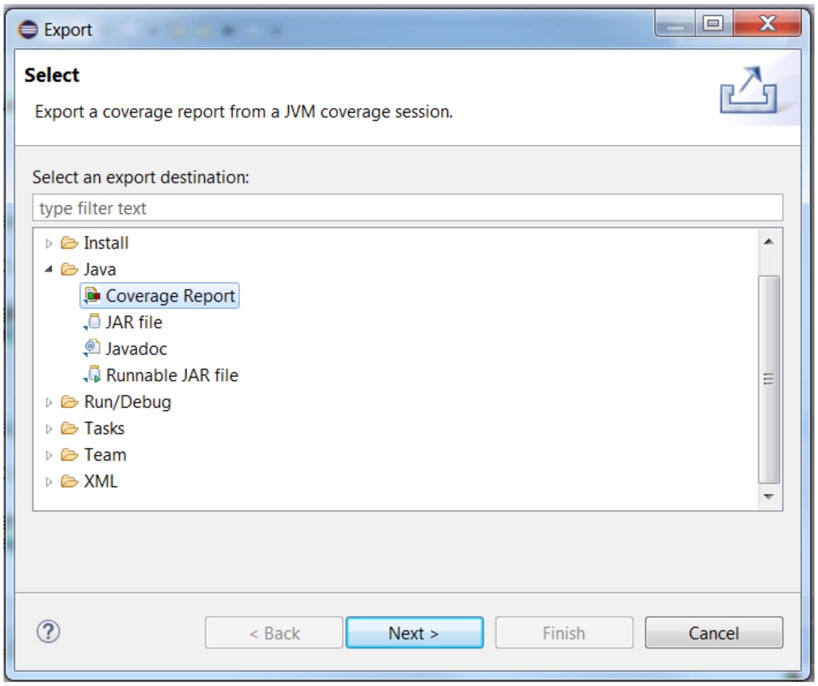
Test | Coverage | Arguments | Classpath | JRE | Source

Analysis scope:

☑ Lab2CodeCoverage - src
☐ Lab2CodeCoverage - test
☐ Lab2CodeCoverage - junit.jar
☐ Lab2CodeCoverage - org.hamcrest.core_1.3.0.v201303031735.jar

5. Using the Coverage menu, run your test cases and take a look at the source code. Is everything green, or are there lines colored "yellow" or "red". Yellow indicates that the line is "partially covered" while red indicates it has "no coverage".
6. Switch into the "Coverage" view in eclipse. Right click on the "element" portion of the code coverage view and select export. Generate a "code coverage report" in html format.

[1] Or, if these instructions will not work, follow the instructions at http://www.eclemma.org/.



Export

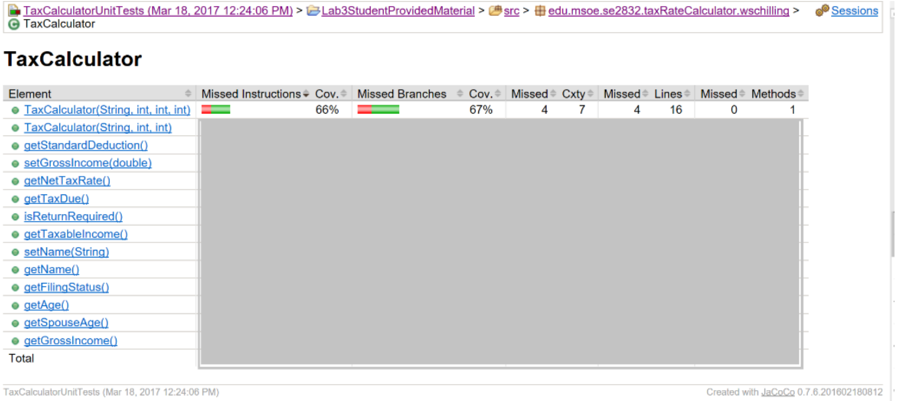
Select

Export a coverage report from a JVM coverage session.

Select an export destination:

type filter text

7. Eventually you should be able to brows through the HTML to find a base coverage report in the exported directory. You should be able to browse and find an image similar to that provided below. (Note this image has been slightly obscured.) Grab a copy of this image for your lab report. Are there methods that have not been tested at all, and are there methods with only partial coverage? Based on this, was the initial testing any good?



TaxCalculatorUnitTests (Mar 18, 2017 12:24:06 PM) > Lab3StudentProvidedMaterial > src > edu.msoe.se2832.taxRateCalculator.wschilling > TaxCalculator

TaxCalculator

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
TaxCalculator(String, int, int, int)	🟥🟩	66%	🟥🟩	67%	4	7	4	16	0	1
TaxCalculator(String, int, int)	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
getStandardDeduction()	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
setGrossIncome(double)	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
getNetTaxRate()	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
getTaxDue()	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
isReturnRequired()	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
getTaxableIncome()	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
setName(String)	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
getName()	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
getFilingStatus()	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
getAge()	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
getSpouseAge()	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
getGrossIncome()	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
Total	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]

TaxCalculatorUnitTests (Mar 18, 2017 12:24:06 PM)

Created with JaCoCo 0.7.6.201602180812

8. Now that you have this as a base analysis, take a look at the Java Doc for the assignment and the explanation of tax calculation given in Appendix A, B, and C. From this, can you develop a set of test requirements in terms of coverage that you would need to fully cover each method?
9. Now that you have setup a set of test requirements for each method, and using the test model which has been provided to you, can you add additional tests to increase the test coverage of your system? To do this, you can proceed in two manners
    a. For simple tests that are not very complex, simple add new test cases to the test file
    b. For more complex test cases, or cases that are related, can you add them to the test model and then export them into the data providers.

Your goal is to try and achieve 100% coverage, but this may or may not be achievable.

4. Lab Deliverables

Each team will be responsible for submitting one report with the following contents:

1. Introduction
   a. What are you trying to accomplish with this lab? This section shall be written IN YOUR OWN WORDS. DO NOT copy directly from the assignment. This should be written in multiple, grammatically correct sentences.
2. Initial Coverage Analysis
   a. Include a graph showing your initial code coverage before any tests have been added.
3. Test requirements
   a. Based on the JavaDoc, document any test requirements for testing each method. This will probably be a table with a column for method names and several columns following for different parameters that impact the method. Within each of the columns, you will then want to list what is needed to test that method.
4. Added tests.
   a. Explain, based on your analysis, the tests that you added to the system in order to meet the requisite coverage criteria. Identify which additional tests required additional test methods and which required changes to existing data providers.
5. Final Coverage Analysis
   a. Include a graph showing your final code coverage after any tests have been added.
6. Discussion of Final Code Coverage
   a. If there are any segments which have not achieved 100% code coverage, identify them (i.e. include a screen shot of the segment of code) and then provide a brief one or two sentence explanation as to why this segment can not be tested to 100% coverage.
7. Things gone right / Things gone wrong
   a. This section shall discuss the things which went correctly with this lab.
   b. This section shall discuss the problems you had in completing this lab.
8. Conclusions
   a. What have you learned with this experience?
   b. What improvements can be made in this experience in the future?

This material should be submitted as a single pdf file named SE2832_Lab3Report.pdf.

Additionally, submit a zip file of your development environment, including the new tests cases and your Java code, as well as the tax spreadsheet model.

If you have any questions, consult your instructor.

Appendix A: The Tax Law

Taxes are complex, but for this assignment, they have been simplified.

The first thing of importance is the person's filing status. A return may indicate that a person is single, head of the household, married filing jointly, married filing separately, or a qualifying widow. While you will not need to check the conditions that would quality a person to file using one of these categories, you will need to use the categories to test your program.

First off you will need to test whether a person needs to file. Depending on the filing status, a person is required to file an income tax return with the IRS if the criterions of table 1 are met.

Table 1. **2008 Filing Requirements Chart for Most Taxpayers (From IRS Publication 501)**

IF your filing status is...	**AND at the end of 2008 you were…**	**THEN file a return if your gross income was at least…**
Single	Under 65	$8950
	65 or older	$10300
Head of household	Under 65	$11500
	65 or older	$12850
Married, filing jointly	Under 65 (both spouses)	$17900
	65 or older (one spouse)	$18950
	65 or older (both spouses)	$20000
Married, filing separately	Any age	$3500
Qualifying widow(er)	Under 65	$14400
	65 or older	$15450

The constructors for the Tax Calculator are responsible for accepting the name of the filer, the type of filer, and the filer's age, along with the spouse's age if the filing type is any form of married.

Income tax is based upon the taxable income of the filer. The taxable income is determined by taking the gross income and subtracting the standard deduction from this amount. The standard deduction amount depends on the filing status and the age of the filer. This information is provided in Table 2.

Table II: Standard Deduction

Filing Status	**Base Standard Deduction**
Single	$5450
Married, filing separately	$5450
Head of household	$8000
Married, filing jointly	$10900
Qualifying widow(er)	$10900

In addition to the base standard deductions, the standard deductions are increased by $1050 for each person on the return who is age 65 or older. For example, a couple who is filing jointly and has the ages of 72 and 71 would have a standard deduction of $13000.

The amount of tax is calculated based upon a sliding scale. For example, a single person making less than 8025 would simply pay 10% on their income. However, if they made more than $32550, they would pay 10% on the first $8025, 15% on the amount between $8025 and $32550, and 25% on the remaining income. B This is detailed in Appendix B.

Appendix B: Tax Law Examples

Jill, age 19, is in college and works part time as a waitress. Each year, she makes a total of $7500.00. Because her income is so low, she is not required by law to file a return. Her taxable income, if she did file, would be $2050.00, and she would be responsible for a total of $205.00 in taxes. While she is not required to file a return, it may be advantageous, as if more than $205.00 has been withheld from her paychecks, she may be able to receive a refund.

Jill's boyfriend Mark, also 19, works as a pizza delivery man. Each year, he makes a total of $11,000.00. He must file a return, as his income is greater than $8950.00. His standard deduction is $5450.00, making his taxable income $11,000.00-$5450.00=$5550.00. His tax due is 10% of his taxable income, yielding a net tax rate of 5.045%.

Oil tycoon, age 95, John D. Rockerfeller earns 5,000,000.00 a year and is single. He must file a return. Because of his age, his standard deduction is $6500.00, yielding taxable income of $4,993,500.00. On his, he pays a total tax of $1,726,321.75, yielding a net tax rate of 34.526%.

Appendix C: Tax Rate Schedules for 2008

From About.com

4.1.1. Single Filing Status

(Tax Rate Schedule X)

- **10%** on taxable income between $0 and $8,025
- **15%** on the taxable income between $8,025 and $32,550; *plus* $802.50
- **25%** on the taxable income between $32,550 and $78,850; *plus* $4,481.25
- **28%** on the taxable income between $78,850 and $164,550; *plus* $16,056.25
- **33%** on the taxable income between $164,550 and $357,700; *plus* $40,052.25
- **35%** on the taxable income over $357,700; *plus* $103,791.75

4.1.2. Married Filing Jointly or Qualifying Widow(er) Filing Status

(Tax Rate Schedule Y-1)

- **10%** on the taxable income between $0 and $16,050
- **15%** on the taxable income between $16,050 and $65,100; *plus* $1,605.00
- **25%** on the taxable income between $65,100 and $131,450; *plus* $8,962.50
- **28%** on the taxable income between $131,450 and $200,300; *plus* $25,550.00
- **33%** on the taxable income between $200,300 and $357,700; *plus* $44,828.00
- **35%** on the taxable income over $357,700; *plus* $96,770.00

4.1.3. Married Filing Separately Filing Status

(Tax Rate Schedule Y-2)

- **10%** on the taxable income between $0 and $8,025
- **15%** on the taxable income between $8,025 and $32,550; *plus* $802.50
- **25%** on the taxable income between $32,550 and $65,725; *plus* $4,481.25
- **28%** on the taxable income between $65,725 and $100,150; *plus* $12,775.00
- **33%** on the taxable income between $100,150 and $178,850; *plus* $22,414.00
- **35%** on the taxable income over $178,850; *plus* $48,385.00

4.1.4. Head of Household Filing Status

(Tax Rate Schedule Z)

- **10%** on the taxable income between $0 and $11,450
- **15%** on the taxable income between $11,450 and $43,650; *plus* $1,145.00
- **25%** on the taxable income between $43,650 and $112,650; *plus* $5,975.00
- **28%** on the taxable income between $112,650 and $182,400; *plus* $23,225.00
- **33%** on the taxable income between $182,400 and $357,700; *plus* $42,755.00
- **35%** on the taxable income over $357,700; *plus* $100,604.00

Appendix D: JavaDoc

edu.msoe.se2832.taxRateCalculator.wschilling

Interface TaxCalculatorInterface

- **All Known Implementing Classes:**

  TaxCalculator

`public interface` **`TaxCalculatorInterface`**

- ***Field Summary***

Fields

Modifier and Type	Field and Description
`static int`	**`HEAD_OF_HOUSEHOLD`** Indicates a person filing as a head of the household.
`static int`	**`MARRIED_FILING_JOINTLY`** Indicates a married couple filing jointly.
`static int`	**`MARRIED_FILING_SEPARATELY`** Indicates a married couple which has chosen to file separately.
`static int`	**`QUALIFYING_WIDOWER`** Indicates a qualifying widower filing.
`static int`	**`SINGLE`** Indicates a single, unmarried filer.

- ***Method Summary***

All Methods**Instance Methods****Abstract Methods**

Modifier and Type	Method and Description
`int`	**`getAge`**`()` Obtain the age of the filer.
`int`	**`getFilingStatus`**`()` Obtain the filing status for the tax return.
`double`	**`getGrossIncome`**`()` Obtain the gross income.
`java.lang.String`	**`getName`**`()` This method will obtain the name of the tax payer.
`double`	**`getNetTaxRate`**`()`

	Calculate the net tax rate.
`int`	**`getSpouseAge`**`()` Obtain the age of the spouse.
`double`	**`getStandardDeduction`**`()` Calculate the standard deduction.
`double`	**`getTaxableIncome`**`()` Obtain the taxable income.
`double`	**`getTaxDue`**`()` Calculate the tax due.
`boolean`	**`isReturnRequired`**`()` This routine will determine if a tax return is required.
`void`	**`setGrossIncome`**`(double grossIncome)` This method will set the gross income.
`void`	**`setName`**`(java.lang.String name)` This method will set the name of the tax payer to the appropriate value.

- ***Field Detail***

- **SINGLE**

```
static final int SINGLE
```

Indicates a single, unmarried filer.

See Also:

`Constant Field Values`

- **HEAD_OF_HOUSEHOLD**

```
static final int HEAD_OF_HOUSEHOLD
```

Indicates a person filing as a head of the household.

See Also:

`Constant Field Values`

- **MARRIED_FILING_JOINTLY**

```
static final int MARRIED_FILING_JOINTLY
```

Indicates a married couple filing jointly.

See Also:

`Constant Field Values`

- **QUALIFYING_WIDOWER**

```
static final int QUALIFYING_WIDOWER
```

Indicates a qualifying widower filing.

See Also:

Constant Field Values

- ### MARRIED_FILING_SEPARATELY

```
static final int MARRIED_FILING_SEPARATELY
```

Indicates a married couple which has chosen to file separately.

See Also:

Constant Field Values

- ## *Method Detail*

- ### getFilingStatus

```
int getFilingStatus()
```

Obtain the filing status for the tax return.

Returns:

The filing status value will be returned.

- ### getAge

```
int getAge()
```

Obtain the age of the filer. Age must be greater than 0.

Returns:

The age of the filer.

- ### getSpouseAge

```
int getSpouseAge()
```

Obtain the age of the spouse. If there is no spouse based on the filing type, a value of 0 will be returned.

Returns:

The age of the spouse.

- ### setGrossIncome

```
void setGrossIncome(double grossIncome)
             throws java.lang.Exception
```

This method will set the gross income. The gross income must be greater than or equal to 0. Otherwise, an exception will be thrown.

Parameters:

grossIncome - This is the gross income for the taxpayer.

Throws:

java.lang.Exception - An exception will be thrown if a negative income is provided.

- ### getGrossIncome

```
double getGrossIncome()
```

Obtain the gross income.

Returns:

The gross income will be returned.

- ### isReturnRequired

```
boolean isReturnRequired()
```

This routine will determine if a tax return is required. A tax return is required based on income and filing status.

Returns:

true will be returned if a tax return is required. False will be returned otherwise.

- ### getStandardDeduction

```
double getStandardDeduction()
```

Calculate the standard deduction.

Returns:

This method will return the standard deduction.

- ### getTaxableIncome

```
double getTaxableIncome()
```

Obtain the taxable income. Taxable income is the gross income minus the standard deduction. Taxable income must never be less than 0.

Returns:

The taxable income will be returned.

- ### getTaxDue

```
double getTaxDue()
```

Calculate the tax due. Calculation will be based upon IRS tax tables.

Returns:

The tax due will be returned.

- ### getNetTaxRate

```
double getNetTaxRate()
```

Calculate the net tax rate. The net tax rate is defined as the tax due divided by the gross income.

Returns:

The net tax rate will be returned. The value is returned as a percent.

- ### getName

```
java.lang.String getName()
```

This method will obtain the name of the tax payer.

Returns:

The name of the taxpayer is returned.

- ### setName

```
void setName(java.lang.String name)
```

This method will set the name of the tax payer to the appropriate value.

Parameters:

name - This is the name of the taxpayer.