PROGRAMLAMA ÖDEVLERİ İÇİN OTOMATİK PUANLAMA SİSTEMİ

Önder DEMİR

Yüksek Lisans Tezi

Bilgisayar Mühendisliği Anabilim Dalı

Ocak 2014

ANADOLU ÜNİVERSİTESİ

JÜRİ VE ENSTİTÜ ONAYI

Önder Demir'in "**PROGRAMLAMA ÖDEVLERİ İÇİN OTOMATİK PUANLAMA SİSTEMİ**" başlıklı Bilgisayar Mühendisliği Anabilim Dalındaki, Yüksek Lisans Tezi 23.12.2013 tarihinde, aşağıdaki jüri tarafından Anadolu Üniversitesi Lisansüstü Eğitim-Öğretim ve Sınav Yönetmeliğinin ilgili maddeleri uyarınca değerlendirilerek kabul edilmiştir.

	Adı-Soyadı	**İmza**
Üye (Tez Danışmanı):	**Yrd. Doç. Dr. Özgür YILMAZEL**	**………………**
Üye:	**Doç. Dr. Hüseyin POLAT**	**………………**
Üye:	**Yrd. Doç. Dr. Gürkan ÖZTÜRK**	**………………**

Anadolu Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu'nun ……………… tarih ve ………… sayılı kararıyla onaylanmıştır.

Enstitü Müdürü

ANADOLU ÜNİVERSİTESİ

ÖZET

Yüksek Lisans Tezi
PROGRAMLAMA ÖDEVLERİ İÇİN
OTOMATİK PUANLAMA SİSTEMİ

Önder DEMİR

Anadolu Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Danışman: Yrd. Doç. Dr. Özgür YILMAZEL
2014, 57 sayfa

Programlama derslerinde öğrenmenin en etkili yollarından biri ödevlerdir. Her ne kadar ödevler etkili olsada ödevleri hazırlamak, dağıtmak ve notlandırmak çok fazla zaman almaktadır. Bir derse katılan öğrenci sayısı arttıkça, o derste verilen ödev sayıları ya da ödevlerin öğreticiliği ve kalitesi düşmektedir. Bu problemi çözmenin yollarından biri otomatik puanlama sistemlerini kullanmaktır. Bu tezde, böyle bir sistemin modellenmesi ve gerçeklenmesi anlatılmıştır. Bu sistem Java programlama dili ve test güdümlü geliştirme metotları temel alınarak gerçeklenmiştir ve sistem dört temel aşamadan oluşmaktadır. Bu aşamalar; ödevi hazırlama, dağıtma, tekrar toplama ve notlandırmadır. Bu sistem 30 öğrencinin katıldığı bir derste, sekiz ödev konusu ile test edilmiş ve sonuçlar bu tez içerisinde sunulmuştur.

Anahtar Kelimeler: Yazılım testi, TDD, Programlama ödevleri, Otomatik notlandırma, Birim test

ABSTRACT

Master of Science Thesis
AUTOMATIC GRADING SYSTEM
FOR
PROGRAMMING HOMEWORK

Önder DEMİR

Anadolu University
Graduate School of Sciences
Computer Engineering Program

Supervisor: Asst. Prof. Dr. Özgür YILMAZEL
2014, 57 pages

One of the best methods of learning in programming courses depends on practical exercises through homework assignments. Preparing, collecting and grading homework manually requires considerable amount of time from instructors. When the number of students increases, the amount of homework given reduces. This reduces effectiveness of the whole course. One way to solve this problem is to give homework via an automatic grading system and get back immediate feedback.

This thesis describes an open source system that grades programming homework automatically. This system uses test driven software development methods and technologies to create homework assignments. The system was tested on engineering students taking computer-programming courses. The results show that quality of the work completed by students increased, and that students performed better in these courses on the overall compared to previous years.

Keywords: Software testing, TDD, programming assignments, automated grading, unit testing

ACKNOWLEDGEMENTS

I would like to thank my advisor Asst. Prof. Dr. Özgür Yılmazel for his guidance and support during my study. It was a great pleasure to work with him during this study.

Önder DEMİR
Ocak 2014

TABLE of CONTENTS

PAGE

ÖZET ..... **i**

ABSTRACT ..... **ii**

ACKNOWLEDGEMENTS ..... **iii**

TABLE of CONTENTS ..... **iv**

TABLE of FIGURES ..... **vi**

TABLE of LISTS ..... **vii**

TABLE of ABBREVIATIONS ..... **viii**

1. INTRODUCTION **1**

1.1. Challenges 2

1.2. Education and Automatic Grading Systems 3

1.3. Test Driven Development 5

1.4. Methods of Evaluating the Study 8

2. CODOMETER: A LANGUAGE-INDEPENDENT AUTOMATIC GRADING SYSTEM USED FOR PROGRAMMING HOMEWORK **10**

2.1. Requirements of Codometer 11

2.2. Codometer: Design Structure 13

2.2.1. Codepoint 13

2.3. Codometer: How To? 15

2.3.1. Step 1: preparing test-based homework 15

2.3.2. Step 2: distributing homework 17

2.3.3. Step 3: instant results of homework 19

2.3.4. Step 4: recollecting homework 20

2.3.5. Step 5: final grading 21

2.4. Codometer: Given Assignments 22

3. METHODOLOGY **23**

4. RESULTS **26**

5. CONCLUSION AND FUTURE WORK **30**

REFERENCES **31**

APPENDIX -1 2nd Week Homework 36

APPENDIX -2 2nd Week Homework Sources 39

TABLE of FIGURES

1.1 Life Cycle of Test Driven Development Process 6
1.2 A Simple Test Case 6
1.3 A Simple Class to Remove Compilation Errors 7
1.4 A Simple Calculator Class to Pass Test Case 7
2.1 Homework Lifecycle of Codometer 11
2.2 Basic Structure of Codepoint 13
2.3 A Sample Codometer Unit Test Case for Java 15
2.4 A Sample Codometer Unit Test Case for C# 15
2.5 A Sample POM File 16
2.6 An Extended Version of Test Cases 17
2.7 Distribution of Homework 18
2.8 A Sample POM File for Students 19
2.9 A Sample Test Case 20
2.10 Codometer Final Grading Lifecycle 21
2.11 A Homework Result Shown to Instructor 22

TABLE of LISTS

2.1 Requirements of Codometer 11
2.2 Basic Methods of CodepointEvaluator 14
2.3 List of Given Homework Using Codometer 22
3.1 Assignment Groups 23
3.2 The Poll 24
4.1 Average Score of Student Groups 27
4.2 Average Score of Students 28

TABLE of ABBREVIATIONS

IDE	:	Integrated Development Enviroment
POM	:	Project Object Model
SVN	:	Subversion
TDD	:	Test Driven Development

1. INTRODUCTION

Today not only institutions that give computer engineering or computer science education but also many other disciplines need to teach programming skills to their students. For this reason, every successful programming course should incorporate programming exercises where students try out their theoretical knowledge [1, 2]. One of the best ways of learning programming is to try to write programs that work [1, 3]. Homework is a proper medium for students to write correct programs. In previous years, homework was prepared and distributed via Internet, collected back, and then manually graded. This manual process included both program output and source code analysis, and thus took time. This process works with small number of students. However, when the number of students increases, productivity decreases and human-related errors increase for the instructor [3, 4]. These problems make instructors reduce the amount of homework given to students. Another problem with conventional homework is that students are unable to see whether they are on the right track or not when finding a solution to the given homework [2, 3].

The motivation behind designing a system is to find a way both to ease the way students learn programming, and to provide a solution to prepare, distribute and collect homework quickly and efficiently [5]. Instructors are using such automated grading systems [2-4, 6-12]. A few of these systems are using learning management systems [12], there is a system that includes GUI testing called JEWL [13], but majority of grading systems are based on the idea of test-driven development (TDD) [5]. TDD is a software development method, which is introduced by extreme programming (XP) [6, 14-17]. Using TDD in the classroom is not a new method [4, 10, 16]. The most important problem with the systems that are based on TDD is that tests to grade homework need to be written in a way that must guide and motivate students. Writing test can be cumbersome [10] and in order to reduce this cumbersomeness, most of these systems are using TDD tools [5]. The main drawbacks of these systems are they do not give the chance of changing the programming language of courses independently and do not provide a proper and well-known medium to distribute homework.

Thinking on these problems, we decided to implement a language-independent system, whose infrastructure is based on well-known TDD frameworks, for instructors of computer-programming courses. We develop a core grading framework called Codepoint that uses jUnit [18] and nUnit [19] test frameworks. To distribute homework, we use Maven [20], project management and build automation tool; and for collecting homework, we use SVN [21] revision control software.

While introducing such a system in a programming course, changes of the environment are inevitable. A typical course system is composed of components such as instructor, students, curriculum and the methods of exercises. If a change occurs in a component, this change might have effects on one or more other components of the system. In our case, the changes will effect dialog time between instructor and student [22]. The course will become to look like a distance education course in some ways.

While emphasizing the problems and trying to trace the change, a concept of distance education called Systems Approach comes up. Systems Approach considers all parts of a course interrelated between each other. This concept considers each course as a system. That means if a component of course changes the other components will absolutely change in a good or bad way. Usually system approaches shows them in designing new courses. But in our case changing only one component of system and experiment the results will be the best practice. Despite this difference there is still a great parallelism with our study and System Approach concept. The traces of System Approach concept can be found in 1960s and early of 1970s. The best well-known experiments of System Approach are Articulated Instructional Media Project (AIM) at 1964 and Open University (OU) at 1969 [23].

1.1. Challenges

While stating the problem, we realized that there are some major problems:

- Students, especially freshmen, have difficulties to find the preliminaries of an assignment, and struggle to create a workable solution of assignment [4].
- While solving the assignment students are not sure if they are on right track or not.
- For instructors, human-related errors are directly proportional with the number of given assignments and number of students [4].
- Both students and instructor have difficulties in distributing and collecting the given assignments [5].
- Despite there are many project management and build automation tools for Java e.g. Maven, there are not many options for .Net environment [24].
- Previous works do not provide a flexible framework to generate various types of assignments.
- Previous works do not support generating assignments in different programming languages.
- Previous works do not give data about their studies if it was a success or not.

The problems mentioned are not directly involved in the learning process. They only reduce the productivity of a course and learning process. As a solution we present Codometer to overcome these problems.

1.2. Education and Automatic Grading Systems

The idea of evaluating homework automatically is not a new approach in this research area [4, 10]. There have been lots of semi or full automatic grading systems and frameworks used by educators [4]. Most of these systems are based on compiling the program and asserting the results of executed program with certain predefined parameters or thresholds. Ala-Mutka [4] categorized this type of assessment as Dynamic Assessments. Most of these systems are checking the assignment from the perspective of functional requirements. There are some systems that check the output of execution using regular expression like coursemarker [25] and pattern recognition like ASSYST [26]. Another approach to check the functional requirements of an assignment is analysing the subparts of the code such as classes and methods. There are some systems that use reflection classes [2, 27] and injection of pre-implemented codes of the instructor. Most of these systems are written for Java platform. So far, there has been only one platform for .NET [2]. This approach has a lot of similarities with a developing methodology called TDD, about which we give more details in Section 2.3. Programming course educators started to give lessons about TDD in their classes [9, 10].

All previous studies in this field have some good features inspiring this field. But they have some drawbacks too. The most significant drawbacks can be stated as:

- They only check the functional requirements by checking the results with threshold values or using injections of predefined codes. There was no study that merges these two methods.
- They support only one programming language.
- They do not present an efficient way to distribute homework.

All these previous works are trying to reduce the cost of evaluating an assignment in a conventional course. Their focus was only to grade an assignment by using one approach. But as time passes the characteristics of courses changed. Now students and instructors want more freedom over the courses. They want to be free of time and space.

Due to improvements in technology of connectivity, distance-learning courses evolve into their sixth generation named distance-learning 2.0 [23]. In this era of distance learning the main tools are computers, world wide web and social networking tools [28].

After overcoming bandwidth problems of Internet, a new distance-learning course type called “Massive Open Online Course (MOOC)”[29] has been developed in late 2000s. MOOC is an online course aimed for unlimited participants. The first MOOC was opened at September 2008 in University of Manitoba[30]. There were only 2200 students enrolled in the first MOOC[30].

But overall participant number of MOOCs all-over the world is increasing exponentially after each year.

Unlike the early stages of distance learning, not only social science but also formal science courses like programming courses started to be a part of distance learning. So a lot of programming course MOOCs starts to be presented.

Assessing techniques of a MOOC varies from simple Multiple Choice responses to peer review feedbacks[31]. Because of unlimited participation and lack of crowed control over participants, assessments is a very important weakness on current MOOCs that is under research[32]. While peer reviewing consumes lots of human resources, multiple-choice questions are not feasible for all types of courses. Since assessment is important in the success of a course, researchers on this field are trying to find a feasible solution to deliver, recollect and evaluate the assignments given in this type of courses. Also participants of MOOCs want to get the instant results of assignments.

Another important characteristic of MOOCs is participants select their collaboration in course. For programming courses this means participants may want to choose the type of assignments depending on their knowledge or interests.

So by the information we gathered, again this can be said that the previous works in automated grading systems are not sufficient for neither MOOCs nor another distance learning course. Although Codometer is designed for a conventional course, it may also be used for distance learning programming courses as a side tool.

1.3. Test Driven Development

TDD is a methodology that is based on writing test before writing the actual code [15, 33, 34]. In TDD strategy, only code that passes from tests is developed [34, 35]. In his book, Kent [15] implies two main rules of TDD:

- Do not write any code without a failing automated tests [15, 35].
- Refactor your code [15, 35].

First rule implies that coders have to write automated test code before implementing their actual code. This causes failing of all tests. By this method, coder makes sure that all of his test codes are running [14, 15, 33-35]. Then coder implements enough codes to pass tests [34]. After all the tests pass successfully, the second main rule comes in hand. The second rule implies that after all tests passed, coder has to refactor his code to reduce the complexity and remove duplications [5]. Then new tests are added to the system, and new code should be written to pass from those tests [5, 33, 35]. In his book, Wake [36] proposes a sample walkthrough list to coder (Figure 1.1):

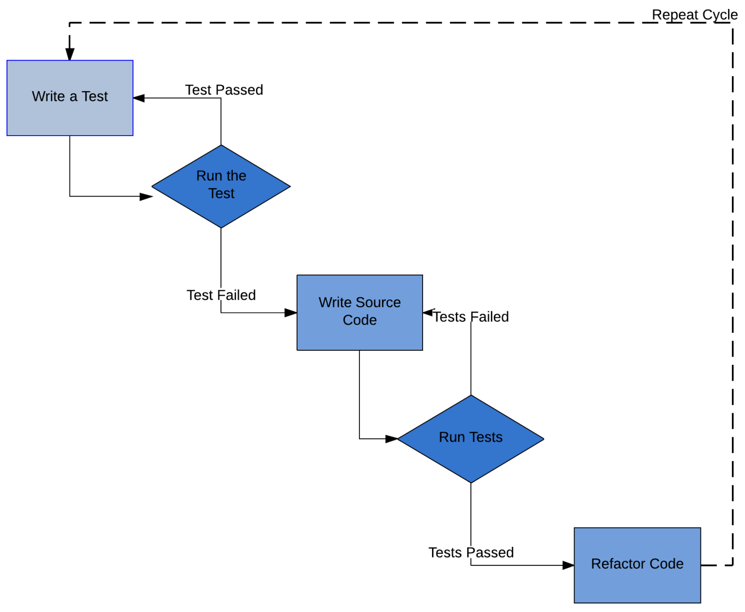


Figure 1.1 Life Cycle of Test Driven Development Process

- Write one test (Figure 1.2).

```
import junit.framework.TestCase;

public class CalculatorTest extends TestCase {

   Calculator cal=new Calculator();

    public CalculatorTest(String name) {
       super(name);
    }

    public void testSum(){
       assertEquals(2,cal.sum(1,1));
    }
}
```

Figure 1.2 A Simple Test Case

- Compile the test. It should fail to compile, because there is not any code that works.

- Implement enough code, which makes errors to disappear and makes the test compliable (Figure 1.3).

```
public class Calculator{

    int sum(int num1,int num2){

        return 0;
    }
}
```

Figure 1.3 A Simple Class to Remove Compilation Errors

- Run the test. You should see it fail because the code, which is tested, has not been implemented yet (You are now sure that your test is working).
- Implement only enough code, which is passed from your tests (Figure 1.4).

```
public class Calculator{

    int sum(int num1,int num2){

        return num1+num2;
    }
}
```

Figure 1.4 A Simple Calculator Class to Pass Test Case

- Run the test, and see it pass.
- Refactor your code for redundancies and duplications.
- Repeat the cycle.

There are some major benefits of repeating that cycle in a development of software. Since this cycle is repeated continuously, the software is developed evolutionarily [5]. While new tests are being developed, old tests are running to see that the code is in a stable state [5].

Writing tests before implementation enables the developer to focus on the interface of the code without drowning in the complexity of implementation details [5, 14, 15, 33, 36]. Refactoring code after a cycle gives a chance to the coder to look for duplication in small amount of code rather than handling mass amount of code [35, 37].

1.4. Methods of Evaluating the Study

In this thesis, we want to express the impact of Codometer to a conventional programming class. There are three main evaluation methods to study the results: quantitative, qualitative and mixed-method evaluation methods.

Qualitative research methods rely on natural settings of a topic mainly using verbal descriptions, case studies and stories [38]. The goal of qualitative research is describe situations by investigating the actors' behaviors and habits by using qualitative ways such as observations and interviews [39].

On the other hand, quantitative research proposes the topic is independent from the beliefs of individuals [40]. Quantitative methods use statistics to summarize and describe a topic [41]. Generally in quantitative methods, researchers are using polls and surveys.

The mixed methods are a combination of qualitative and quantitative methods [38, 42]. In mixed methods approach, researcher choses his methods due to needs of situation which means the researcher may use the benefits of both methods to collect data to understand problems [43].

Although up to today almost all researches on education were quantitative [38] and there are some researches reported that using mixed-methods and qualitative methods.

In this thesis we will use both quantitative and qualitative research methods.

To determine the progress of students, we used quantitative methods because there is no obvious relation between student beliefs and actual improvement of students. Choosing quantitative methods approach has another advantage; numerical results are better in emphasizing the success of study. In order to gather results from students, we administrated a poll. We used Likert [44, 45] which is one of the scaling methods, to create the poll.

On the other hand, the problems and needs of instructors are relative to their tuition style. To understand and experience the problems and needs of instructors we used qualitative methods such as interviewing and observing their problems during the assignment process.

2. CODOMETER: A LANGUAGE-INDEPENDENT AUTOMATIC GRADING SYSTEM USED FOR PROGRAMMING HOMEWORK

Codometer [46] is an open source tool developed by the authors to distribute homework assignments and automate grading of students' submissions.

The main advantage of Codometer is the support of multiple languages. The instructors can give the same homework in any programming language according to the students' level and knowledge.

Codometer is using the hybrid methods of dynamic assessments [4] such as compiling the program and asserting the results of executed programs with predefined parameters and also is using reflection classes and injection of pre-coded classes and methods. This hybrid method helps the instructor assess the assignments in more than one way, depending on the characteristic of the assignment.

Codometer also introduces a distribution method different from its successors. Codometer uses Maven to distribute homework.

Codometer has three main steps seen in Figure 2.1 Homework Lifecycle of Codometer: distribution of the homework with test packages, recollecting the student solutions using source control repository and automatically grading of the committed codes. In conventional announcement process of assignments, there are always problems that are not part of the learning process. For instance, not all students get or find the preliminaries of homework, and students struggle to create working solution of homework. One of the most important advantages of using Codometer is the simplicity of distributing the homework. Before we start the Codometer project, in our requirement analysis phase, we found out that 37% of students are not sure if they understood the requirements of homework well. In another words, they want to know if they prepared the solution of homework correctly. In order to solve this problem, we package homework and use Maven's dependency management infrastructure. Students can setup their working environment by adding the prepared homework package to their Maven project as a dependency. All homework packages include a command, so that students can execute the grading process and see their progress on the given homework. Another important feature of Codometer is that it allows instructors to change the programming language of course independently according to preliminaries and student profile. Currently, with Codometer, instructors can give homework written in Java and C# languages.

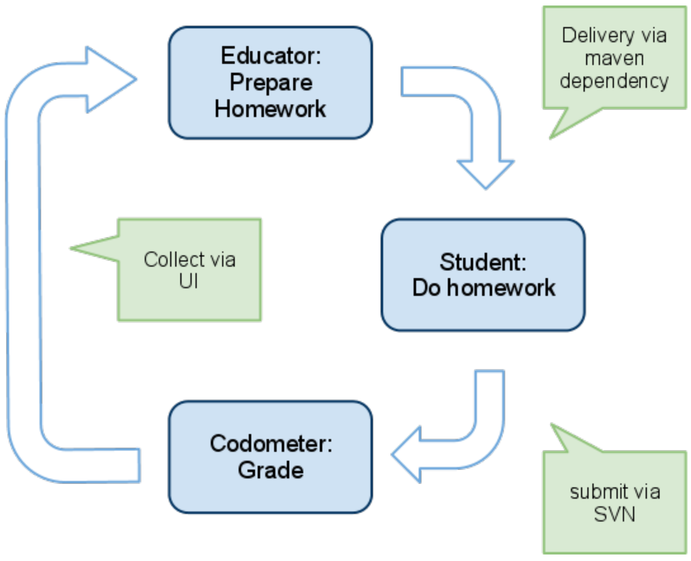


Figure 2.1 Homework Lifecycle of Codometer

2.1. Requirements of Codometer

It is widely recognized that determining the requirements of an information system is essential for success in design[47]. There are some requirements of Codometer given at Table 2.1 Requirements of Codometer. These requirements are gathered in the analyzing phase and approved by the mentor of this thesis.

Table 2.1 Requirements of Codometer

Requirement Code		Requirement
CREQ-	1.	Instructor should write the testing code in unit testing structure.
CREQ-	2.	Instructor should be able to package the tests.
CREQ-	3.	Instructor should be able to give points to every test.
CREQ-	4.	Instructor should be able to start writing test without any installation rather than Codometer.
CREQ-	5.	Instructor should be able to deliver test packages.

Requirement Code		Requirement
CREQ-	6.	Instructor should be able to change the test packages easily by using Codometer.
CREQ-	7.	Instructor should be able to set a deadline to homework.
CREQ-	8.	Instructor should be able to change the deadline of homework.
CREQ-	9.	Instructor should be able to start grading process manually.
CREQ-	10.	Instructor should be able to write their tests in multiple programming languages.
CREQ	11.	Instructors should be able to trace how many students deliver their homework.
CREQ-	12.	Students should be able to get their homework using Codometer.
CREQ-	13.	Students should be able to deliver their solutions using Codometer.
CREQ-	14.	Students should be able to see their instant grades.
CREQ-	15.	Students should be able to write their solution in multiple programming languages.
CREQ	16.	Students should have unique user names and passwords.
CREQ	17.	Students should be able to send their solutions with their user credentials.
CREQ-	18.	Codometer should provide methods to instructors which helps instructors to write their tests.
CREQ-	19.	Codometer should grade all students’ homework automatically after the deadline.
CREQ-	20.	Codometer should provide a user interface for instructor to set the preliminaries of homework.
CREQ-	21.	Codometer should be extendible for other testing frameworks.
CREQ-	22.	Codometer should be able to integrate with various dependency management tools.
CREQ-	23.	Codometer should be able to integrate with source control tools.
CREQ-	24.	Codometer should be able to export results of each assignments into XML files.

2.2. Codometer: Design Structure

Codometer uses Maven for dependency management and distribution and SVN for recollecting homework from students. To make grading, Codometer uses a tool called CodePoint, which is a tool, based on Junit and Nunit testing frameworks.

2.2.1. Codepoint

Codepoint is the basis of Codometer system. Codepoint is used for both testing and grading. A system component view of CodePoint can be seen in Figure 2.2 Basic Structure of Codepoint.

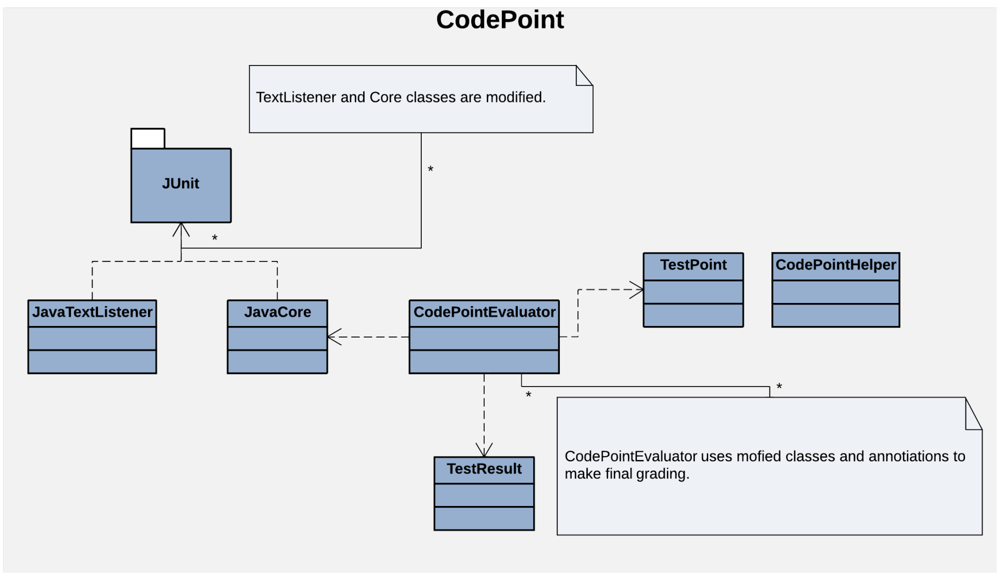

CodePoint
TextListener and Core classes are modified.
JUnit
JavaTextListener
JavaCore
CodePointEvaluator
TestPoint
CodePointHelper
TestResult
CodePointEvaluator uses mofied classes and annotiations to make final grading.

Figure 2.2 Basic Structure of Codepoint

To gain the ability to write test cases, CodePoint uses xUnit frameworks. These frameworks are JUnit for Java and NUnit for C# programming languages. These frameworks are well-documented and trustworthy frameworks in their field.

Despite the mobility and the power of these unit test frameworks, they are not enough to grade homework. To gain grading ability Codepoint modifies these frameworks and injects some powerful extensions such as Codometer assertion methods seen in Table 2.2 Basic Methods of CodepointEvaluator and TestPoint annotiations.

Table 2.2 Basic Methods of CodepointEvaluator

Method Name	Method Description
evaluateMethod()	Checks the method if it is an Codometer annotated method or not
getInstance()	Gets a CodepointEvaluator class.
StartTesting(String)	Starts Codometer cycle
assertArrayEquals(Object[], Object[])	Checks the equality of two arrays' values
assertArrayEquals(String, Object[], Object[])	Checks the equality of two arrays' values. If not returns the first parameter as an error message
assertBetween(int, int, int)	Checks if an integer value is between the given ranges.
assertEquals(double, double)	Checks if two double value are equal or not.
assertEquals(double, double, double)	Checks if two double value's equality with a threshold value.
assertEquals(Object[], Object[])	Checks the equality of two arrays.
assertEquals(Object, Object)	Checks the equality of two objects.
assertFalse(boolean)	Checks the value of a boolean, if it is false or not.
assertNotNull(Object)	Checks the value if it is NULL(1) or not.

To be marked as an Codometer test method, that method must be annotated with TestPoint annotation. Then if that method uses a Codometer assertion method that method is marked as a Codometer test method and given grade point is added to global point.

Another important class is CodePointHelper class, which has injection and reflection methods. These methods are used for getting methods and classes, which are used by students in their solutions by using reflection classes.

(1) NULL, in programming languages, express that any value is not assigned to a variable.

2.3. Codometer: How To?

There are five basic steps of Codometer. In this chapter, these five basic steps will be expressed.

2.3.1. Step 1: preparing test-based homework

Codometer uses the dynamic approach of testing and mainly uses dependency injection techniques and reflection methods.

To prepare a test-based homework, instructor has to create a Maven project. In POM file, she has to declare dependency to Codepoint, the grading tool of Codometer. Codepoint uses a free testing tool called jUnit for Java based homework and nUnit for C# based homework as core. With Codepoint, educator can create countless test cases and assign points for assignments. Before every test method, educator has to put two annotations, first @TestPoint(<point for Test>) and second @Test for Java-based homework. For C# based homework educator has to add "TestPoint(<point for Test>)" attribute after [Test] attribute. An example of a java test case is shown in Figure 2.3:

```java
@TestPoint(25)
@Test

public void testSum()
{
    int expected= 2;
    int result= Math.add(1,1);
    CodePointEvaluator.getInstance().assertEquals(expected,result);
}
```

Figure 2.3 A Sample Codometer Unit Test Case for Java

An example of a C# test case is shown in Figure 2.4:

```csharp
[Test,TestPoint(10)]


public void testSum()

{
     int a = 20, int b=22;
     int expected= 42;
     int result= Math.add(a,b);
     CodePointEvaluator.getInstance().assertEquals(expected,result);
}
```

Figure 2.4 A Sample Codometer Unit Test Case for C#

After preparing all test cases instructor has to package his test classes into jar files or dll files depending on the programming language. To pacakage homework, instructor uses maven. To use maven instructor creates a POM (Figure 2.5) file.

```xml
<project>
	<modelVersion>4.0.0</modelVersion>
	<groupId>com.codometer.homework.bim460</groupId>
	<artifactId>homework8</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<repositories>
		<repository>
			<id>internal</id>
			<name>Archiva Managed Internal Repository</name>
			<url>http://cengsvn.anadolu.edu.tr:8080/archiva/repository/internal</url>
		</repository>
	</repositories>
	<dependencies>
		<dependency>
			<groupId>com.codometer</groupId>
			<artifactId>codepoint</artifactId>
			<version>0.0.1-SNAPSHOT</version>
		</dependency>
	</dependencies>
</project>
```

Figure 2.5 A Sample POM File

Running “maven install” command makes maven to create test packages. After successful compiling, instructor has a test package, which can be used by students in their homework solutions. In this phase, we suggest two steps of test cases. First step of cases will be distributed to the students only if they are on the right way to solve the homework. In the second step of cases, educator will use Codometer to do final grading. This second step can be a new set of test cases or also an extended version of the first step cases (Figure 2.6) or the same test cases of the first step.

```
@TestPoint(25)
@Test

public void testSum()
{
    int expected= 2;
    int result= Math.add(1,1);
    CodePointEvaluator.getInstance().assertEquals
                        ("1+1=2 olmalıdır",expected,result);
}
```

Figure 2.6 An Extended Version of Test Cases

After the deadline of homework, instructor can easily change his first step with second step test cases or leave it in the same way and see the final grading of homework. By using first step test cases, students will only have a brief idea if they are on the right way to the solution of the problem. Using two sets of test cases to grade encourages students to do more tests on their own.

2.3.2. Step 2: distributing homework

To distribute the homework, we use MAVEN. MAVEN is an open source tool to manage distribution and dependency management. A sample illustration of Codeometer's dependency management structure is shown in Figure 2.7 Distribution of Homework:

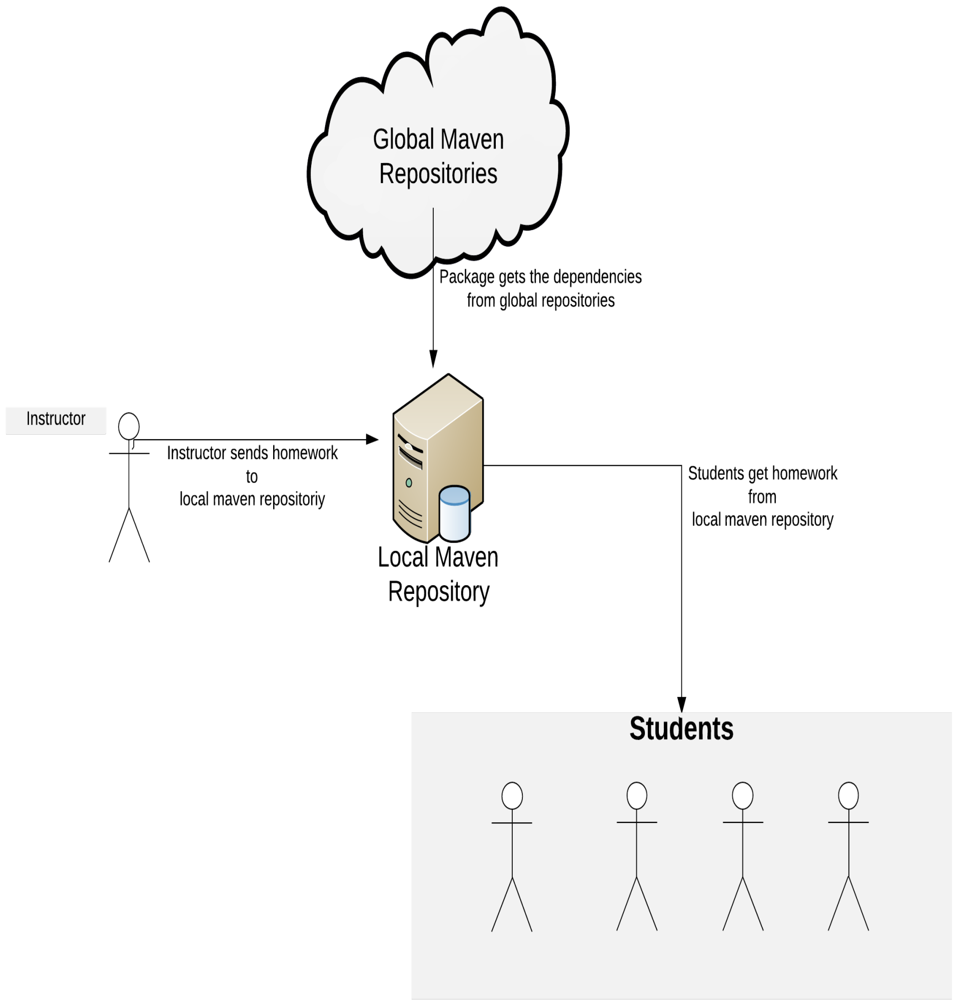

Global Maven Repositories
Package gets the dependencies from global repositories
Instructor
Instructor sends homework to local maven repositoriy
Local Maven Repository
Students get homework from local maven repository
Students

Figure 2.7 Distribution of Homework

Instructor has to put her test packages in a maven repository, which is accessible by students. In the announcement of homework, instructor must announce the settings of the test package, the name and URL of repository, group id, artifact id and version of test package, which are essential parts of a regular Maven project. An example of part of POM file is shown Figure 2.8 A Sample POM File for Students:

```
<project>
          <modelVersion>4.0.0</modelVersion>
          <groupId>com.codometer.homeworks</groupId>
          <artifactId>homework8Impl</artifactId>
           <version>0.0.1-SNAPSHOT</version>
          <repositories>
                    <repository>
                              <id>internal</id>
                              <name>Archiva Managed Internal Repository</name>
                              <url>http://cengsvn.anadolu.edu.tr:8080/archiva/repository/internal/</url>
                    </repository>
          </repositories>
          <dependencies>
                    <dependency>
                              <groupId>com.codometer.homework.bim460</groupId>
                              <artifactId>homework8</artifactId>
                              <version>0.0.1-SNAPSHOT</version>
                    </dependency>
          </dependencies>
</project>
```

Figure 2.8 A Sample POM File for Students

2.3.3. Step 3: instant results of homework

After student creates a successful error-free solution, she must use the method *startTests* with the full-qualified path of her package as a parameter. While the test cases in package run, the student can instantaneously see the output of tests and the scores she gets. If she made a mistake during the implementation, she will simultaneously find the line number, the expected value, the error messages that Codepoint creates or the helpful hints that instructor puts in the test code to guide student in completing homework.

After student calls method startTests, each test case runs one by one. For example if a test case is like at Figure 2.9 A Sample Test Case, after running the test case, student will see messages on the screen.

```
        @TestPoint(25)
        @Test
        public void CheckAveaSend() {
                init(packageName);
                System.out.println("Trying to send a message from Avea using
IConnectorAdapter.send()...");
                try {
                        IConnectorAdapter connector = (IConnectorAdapter) AveaClass.newInstance();
                        String message="Hello World!";
                        connector.send("905555555555", message);
                        eval.assertEquals("Checking Avea Phone Number Format... Passed",
                                "Checking Avea Phone Number Format... FAILED",
                                "MSISDN:905555555555", AveaConnector.recipient);
                        eval.assertEquals("Checking Avea Message... Passed",
                                "Checking Avea Message... FAILED",
                                message, AveaConnector.message);
                        System.out.println("Trying to send a message from Avea using
IConnectorAdapter.send()...Done");
                } catch (Exception e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
        }
```

Figure 2.9 A Sample Test Case

At first student will see the message: "*Trying to send a message from Avea using IConnectorAdapter.send()...*". After this message student will see a message: "*Checking Avea Phone Number Format... Passed*" if his code passes the formatting test. After the test completed, student will see a message: "*Trying to send a message from Avea using IConnectorAdapter.send()...Done*".

2.3.4. Step 4: recollecting homework

In conventional method, each student has to deliver her homework solution to instructor via e-mail, FTP or by physical media (such as CD, USB drive, etc.). This process works well with small number of students. Nevertheless, when the number of students increases, productivity decreases and human-related errors such as losing solution increase. This causes another burden on instructor in archiving the solutions. To overcome these challenges, we are using SVN, a free source control tool. Every student has to have a repository, which can be accessed by himself and Codometer. We suggest instructors to give only read authentication to Codometer. Each student commits her solution to SVN. SVN provides a safe place for solutions and keeps track dates of commits. So even if a student commits homework after the deadline, with SVN instructor can checkout the last solution committed before the deadline. After the deadline, Codometer collects all homework from SVN automatically. After gathering all homework, the final grading step starts.

2.3.5. Step 5: final grading

Last part of Codometer process is grading. After collecting all homework, Codometer starts grading process at a time which educator may set or by manual grade command by educator anytime. The flow of final grading step can be seen in Figure 2.10 Codometer Final Grading Lifecycle.

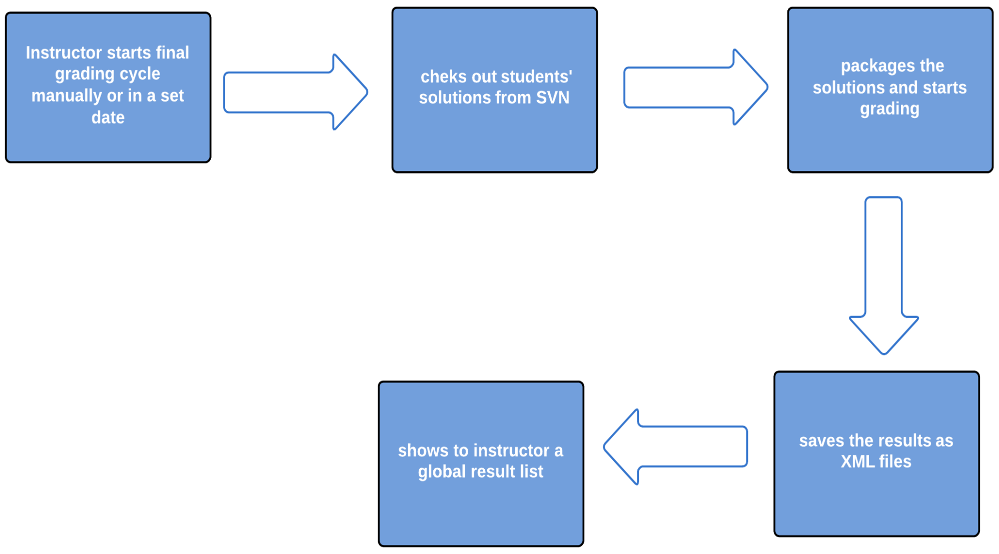
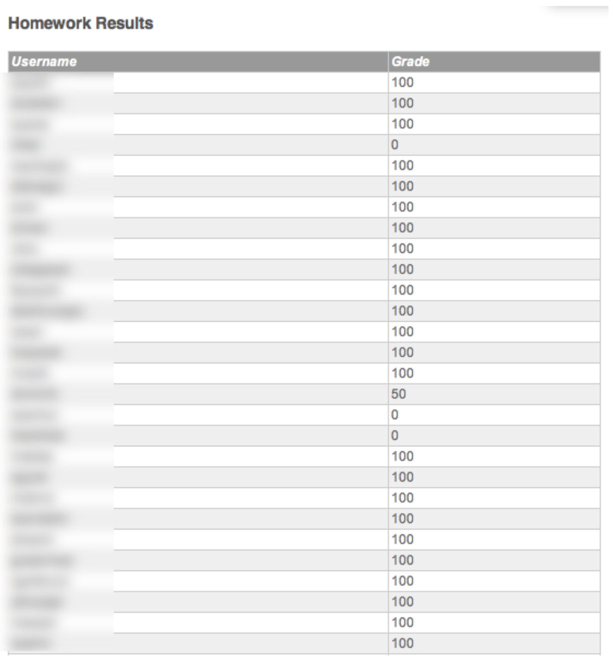

Instructor starts final grading cycle manually or in a set date
cheks out students' solutions from SVN
packages the solutions and starts grading
saves the results as XML files
shows to instructor a global result list

Figure 2.10 Codometer Final Grading Lifecycle

After grading finishes, the results could be seen in Codometer user interface. The result interface can be seen at Figure 2.11 A Homework Result Shown to Instructor. The usernames of students blurred due to privacy concerns. The results are also exported to XML files so that they can be used in further analysis.



Homework Results

Username	Grade
[redacted]	100
[redacted]	100
[redacted]	100
[redacted]	0
[redacted]	100
[redacted]	100
[redacted]	100
[redacted]	100
[redacted]	100
[redacted]	100
[redacted]	100
[redacted]	100
[redacted]	100
[redacted]	100
[redacted]	100
[redacted]	50
[redacted]	0
[redacted]	0
[redacted]	100
[redacted]	100
[redacted]	100
[redacted]	100
[redacted]	100
[redacted]	100
[redacted]	100
[redacted]	100
[redacted]	100
[redacted]	100

Figure 2.11 A Homework Result Shown to Instructor

2.4. Codometer: Given Assignments

By using Codometer, there are eight assignments given. In Table 2.3, a list of given homework can be seen. Sample POM file and source codes of Week 2 homework, Weather-Broadcasting, can be seen in Appendix 1. These all assignments are about the main topic of course, design patterns.

Table 2.3 List of Given Homework Using Codometer

Week #	Name of Homework
Week 1	Reach First
Week 2	Weather-Broadcasting
Week 3	SMS sender
Week 4	Pseudo Osi Layer 3-4
Week 5	Village
Week 6	Composite Components
Week 7	Connector Adapters
Week 8	Cruise Control –State Machine

3. METHODOLOGY

Codometer is used in Advanced Programming Course in the Computer Engineering Department at Anadolu University. There were thirty students enrolled in this course. These students had taken three semesters of programming courses in Java. We divided these students into three groups (SG1, SG2 and SG3) and divided assignments into three groups as seen in Table 3.1 Assignment Groups. “Assignment Group 1” (AG1) consisted four assignments generated by using Codometer. “Assignment Group 2” (AG2) consisted four assignments generated in conventional way. “Assignment Group 3” (AG3) consisted four assignments generated by using Codometer. In first four weeks period SG1 took AG1, SG2 took AG2. In the same period SG3 took assignments from either AG1 or AG2. And all of three-student groups took the third assignment group for the rest of their semester. We administrated same poll three times of the semester. First at the start of semester, the second time at the end of Week four and the last time is at the end of semester. The poll is shown Table 3.2 The Poll.

Table 3.1 Assignment Groups

Week #	Name of Homework	Assignment Group
Week 1	Reach First	Group 1&2
Week 2	WeatherBroadcasting	Group 1&2
Week 3	SMS sender	Group 1&2
Week 4	Pseudo Osi Layer 3-4	Group 1&2
Week 5	Village	Group 3
Week 6	Composite Components	Group 3
Week 7	Connector Adapters	Group 3
Week 8	Crusise Control –State Machine	Group 3

Table 3.2 The Poll

Number	Question
1	I have been in a professional project before. ☐ Yes ☐ No
2	Test Driven Development is a good way to programming. ☐ Strongly Disagree ☐ Disagree ☐ No Idea ☐ Agree ☐ Strongly Agree
3	Using a version control tool is a must in programming. ☐ Strongly Disagree ☐ Disagree ☐ No Idea ☐ Agree ☐ Strongly Agree
4	I find bugs while I am coding. ☐ Strongly Disagree ☐ Disagree ☐ No Idea ☐ Agree ☐ Strongly Agree
5	Writing tests before coding is not possible. ☐ Strongly Disagree ☐ Disagree ☐ No Idea ☐ Agree ☐ Strongly Agree
6	Using a dependency management tool simplifies coding. ☐ Strongly Disagree ☐ Disagree ☐ No Idea ☐ Agree ☐ Strongly Agree
7	Every week there must be homework. ☐ Strongly Disagree ☐ Disagree ☐ No Idea ☐ Agree ☐ Strongly Agree
8	I will use test driven development. ☐ Strongly Disagree ☐ Disagree ☐ No Idea ☐ Agree ☐ Strongly Agree

9	I will use a source control tool in my professional life. ☐ Strongly Disagree ☐ Disagree ☐ No Idea ☐ Agree ☐ Strongly Agree
10	Using a automated grader is a good way to give assignments. ☐ Strongly Disagree ☐ Disagree ☐ No Idea ☐ Agree ☐ Strongly Agree
11	Using TDD helps me find my bugs. ☐ Strongly Disagree ☐ Disagree ☐ No Idea ☐ Agree ☐ Strongly Agree
12	Homework is a good method in programming courses. ☐ Strongly Disagree ☐ Disagree ☐ No Idea ☐ Agree ☐ Strongly Agree
13	I can participate in a professional project as developer. ☐ Strongly Disagree ☐ Disagree ☐ No Idea ☐ Agree ☐ Strongly Agree

4. RESULTS

We conducted this study by using Codometer in Advanced Programming Course in the Computer Engineering Department at Anadolu University. There were 30 students enrolled in this course and eight homework assignments were given using Codometer. To evaluate the effectiveness of Codometer use in the course, we administered same poll at the beginning of semester, at the middle of semester and at the end of the semester.

At the beginning of semester before administering the poll we explained Codometer and the mechanics of Codometer to students. The results of the poll are:

55% of students collaborated in a professional project before. While 90% of students never heard before about TDD, only 5% of students have used this strategy and 5% of students do not know if they used or not TDD. While 98% of students had no information about if TDD is useful or not, only 2% of students thought TDD is a good way to write programs. Only 2% of students thought that they will use TDD in their future life, the rest of the students indicate that writing test code before the actual code is not possible or not a good way to code. 94.4% of students had no idea about if automatic grader will be a good method in the class or not, despite the majority 4% of students strongly disagrees that automatic graders would be good at class.

Only 3% of students strongly agree, 2% agree that using a version control tool is important while writing programs, on the other hand 86% of students disagrees that using a version control tool is necessary. 91% of students disagree that using dependency management tools are necessary, on the other hand only 4% of students thought that dependency management tools are beneficial while programming.

The majority of students who collaborate in a professional project, approximately 90%, indicate that the clients found majority of their bugs.

95% of students indicated that every week there must be at least one assignment. 55% of students agree that they can participate a developer role in a project, on the other hand 33% of students disagree that they are ready to professional life.

Average score of student groups for first four weeks period can be seen in Table 4.1.

Table 4.1 Average Score of Student Groups

	Week 1	Week 2	Week 3	Week 4
Student Group 1	62.1	63.3	80.2	87.9
Student Group 2	75	73	65.9	66.0
Student Group 3	75	65	45.3	80.4

As can be seen in Table 4.1, at first and second weeks the “Student Group 1” had some problems with Codometer but after getting familiar with Codometer, an improvement of scores are observed each week. Despite the success of “Student Group 1”, since scores are inversely proportional with the complexity of the assignment, scores of “Student Group 2” decreases each week as the complexity increases for each assignment week by week. “Student Group 3” suffers the same causes of “Student Group 2” while they got the same assignments at weeks 1, 3. But again a great increase of scores can be observed at week 4 because of getting familiar with the Codometer.

After week four, we administrated the same poll again. The responses from the “Student Group 2” were not different from first poll responses. The responses of Group 1&3 were different from the beginning of semester. 90% of students strongly agree, 7% of students agree that TDD is a good method while programming. 85% of students strongly agree, 5% of students agree that if project is using TDD they have the confidence to take a developer role in the project.

75% of students strongly agree, 22% agree that using version control system and dependency management system while coding is necessary. 97% of students agree that they will use version control systems and version control systems in their professional life.

The average scores of students from week 4 to the end of the semester can be seen Table 4.2.

Table 4.2 Average Score of Students

	Week 5	Week 6	Week 7	Week 8
Student Group 1&3	95.3	93.3	98.7	97.9
Student Group 2	65	80.4	89.5	93.3

Due to lack of experience of "Student Group 2" at week 5 an overall decrease of average score is observed. But after getting familiar with Codometer mechanics there were a great increase at average scores week by week.

We administrated the last poll at the end of semester. The responses from students were very positive according to the first poll we administrated at the beginning of semester. Students expressed the beneficiate of the course to their future life. 94% of students strongly agree, 5% of students agree that TDD is a good method while programming. Approximately 95% of students agree or strongly agree that they will use TDD in their professional future life. 95% of students agree or strongly agree that after using TDD methodology they have released more bug free software. 80% of students strongly agree, 17% agree that using version control system and dependency management system while coding is necessary. 97% of students agree that they will use version control systems and version control systems in their professional life. 85% of students strongly agree, 10% of students agree that if project is using TDD they can take a developer role in the project.

75% of students also indicate that Codometer do not have support of writing testing codes. So they need more assistance on writing tests.

Overall at the end of the semester, we got very positive responses from both student and instructor groups. Students expressed a strong preference for Codometer compared to the classical homework assignments. From the instructor's view, we see a 40% increase in time while preparing the homework, but the grading procedure has shortened by approximately 90%. Instructors only have to answer the questions of students about the homework. Thus, in overall, the time spent on an assignment decreases by 82%. Instructors stated that they are willing to use Codometer in their future courses. But instructors also indicated that Codometer is more suitable for algorithms or data structure courses and is vulnerable against plagiarism in code.

Also, instructors requested some important new features:

- They need a framework to enable students to do more tests by themselves.
- They need a framework to test user interfaces in homework assignments.
- They need a module for detect plagiarism in written codes.

After this study, we gathered critical information about Codometer system and its benefits to students. After applying Codometer, students learnt the concepts about TDD, source controlling and dependency management. Students started to write tests before coding not only in class but also in their professional projects. Students prefer Codometer more than conventional way and prefer the use of Codometer in other classes too. Students feel more confident about writing codes than before.

5. CONCLUSION AND FUTURE WORK

In this thesis, we presented an open source tool created at Anadolu University Computer Engineering department to distribute and grade programming assignments automatically. We found that using Codometer, an automated grading system, in the classroom, is a very positive experience and increases the productivity of the course. A significant increase in the quality of student codes and decrease in the human-related errors is observed.

One of the most important features of Codometer is the support of multiple programming languages. Currently Codometer supports Java and C#. We plan to continue to use Codometer in programming courses at Computer Engineering and in Industrial Engineering classes that use C# programming languages. For the future, we plan to add support for other programming languages like C/C++ to Codometer. Also because of the flexibility to create various types of assignments, Codometer is a good candidate for distance learning courses.

We further plan to add new frameworks and new features to Codometer according to the feedback from student and instructors groups. We intend to add a new framework to Codometer named Testometer, which will be used for improving students' ability to write test cases. In this scenario, instructor writes a set of buggy code and expects students to write codes that reveal and catch those bugs. Another feature that is worth adding is the identification of plagiarism in code, which can be achieved by integrating Moss [48] from Stanford University.

As a result of using Codometer to create assignments, there will become a library of assignments. On instructor view, to find a usable assignment will become troublesome. To overcome this, we plan to add labelling feature to Codometer-assignments in our future releases.

REFERENCES

1. Huet, I., et al., *New Challenges in Teaching Introductory Programming Courses: a Case Study.* 34th ASEE/IEEE Frontiers in Education Conference, 2004.
2. Cerioli, M. and P. Cinelli, *GRASP: Grading and Rating ASsistant Professor.* Proceedings of the ACM-IFIP IEEIII 2008 Informatics Education Europe III Conference Venice, Italy, December 4-5, 2008, 2008: p. 37-51.
3. Cheang, B., et al., *On automated grading of programming assignments in an academic institution.* Comput. Educ., 2003. **41**(2): p. 121-131.
4. Ala-Mutka, K., *A Survey of Automated Assessment Approaches for Programming Assignments.* Computer Science Education, 2005. **15**(2): p. 83-102.
5. Önder Demir, A.S., Ahmet Arslan, Özgür Yılmazel, *Automatic Grading System for Programming Homework*, in *Annual International Conference on Computer Science Education*2010.
6. EDWARDS, S.H., *Using Test-Driven Development in the Classroom: Providing Students with Automatic, Concrete Feedback on Performance.*
7. Edwards, S.H., *Rethinking computer science education from a test-first perspective*, in *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*2003, ACM: Anaheim, CA, USA. p. 148-155.
8. Morris, D.S. *Automatic grading of student's programming assignments: an interactive process and suite of programs*. in *Frontiers in Education, 2003. FIE 2003 33rd Annual*. 2003.
9. Jones, C.G., *Test-driven development goes to school.* J. Comput. Small Coll., 2004. **20**(1): p. 220-231.

10. Li-Ren, C., et al. *An evaluation of TDD training methods in a programming curriculum*. in *IT in Medicine and Education, 2008. ITME 2008. IEEE International Symposium on*. 2008.

11. Matt, u.v., *Kassandra: the automatic grading system.* SIGCUE Outlook, 1994. **22**(1): p. 26-40.

12. Suleman, H., *Automatic marking with Sakai*, in *Proceedings of the 2008 annual research conference of the South African Institute of Computer Scientists and Information Technologists on IT research in developing countries: riding the wave of technology*2008, ACM: Wilderness, South Africa. p. 229-236.

13. English, J., *Automated assessment of GUI programs using JEWL*, in *Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education*2004, ACM: Leeds, United Kingdom. p. 137-141.

14. Beck, K., *Aim, fire [test-first coding]*. Software, IEEE, 2001. **18**(5): p. 87-89.

15. Beck, K., *Test-Driven Development: By Example*. Addison-Wesley, Boston, MA.

16. Keefe, K., J. Sheard, and M. Dick, *Adopting XP practices for teaching object oriented programming*, in *Proceedings of the 8th Australasian Conference on Computing Education - Volume 52*2006, Australian Computer Society, Inc.: Hobart, Australia. p. 91-100.

17. Beck, K. and C. Andres, *Extreme Programming Explained: Embrace Change (2nd Edition)*. 2004: Addison-Wesley Professional.

18. Mentor, J.-O. *JUnit.org Resources for Test Driven Development*. 2011; Available from: http://www.junit.org/.

19. nUnit. *NUnit- a unit-testing framework for all .Net languages*. 2011; Available from: http://www.nunit.org.

20. maven. http://maven.apache.org/. 2011; Available from: http://maven.apache.org/.

21. SVN. *Open Source Software Engineering Tool*. 2011; Available from: http://subversion.tigris.org/.

22. Moore, M.G., *Handbook of Distance Education*. 2007.

23. Michael Moore, G.K., *Distance Education- A Systems View*. Second Edition ed. 2005.

24. NPANDAY. *NPanday - a project to integrate Apache Maven into .NET development environments*. 2012; Available from: http://incubator.apache.org/npanday/.

25. Higgins, C., et al., *The CourseMarker CBA System: Improvements over Ceilidh*. Education and Information Technologies, 2003. **8**(3): p. 287-304.

26. Jackson, D. and M. Usher, *Grading student programs using ASSYST*, in *Proceedings of the twenty-eighth SIGCSE technical symposium on Computer science education*1997, ACM: San Jose, California, United States. p. 335-339.

27. Bettini, L., et al. *An environment for self-assessing Java programming skills in first programming courses*. in *Advanced Learning Technologies, 2004. Proceedings. IEEE International Conference on*. 2004.

28. Yüzer, T.V., *Uzaktan Öğrenmede Etkileşimlilik*. 2013.

29. McAuley, A., et al., *The MOOC model for digital practice*, 2010.

30. Mackness, J., S. Mak, and R. Williams. *The ideals and reality of participating in a MOOC*. in *Networked Learing Conference*. 2010. University of Lancaster.

31. Conole, G., *MOOCs as disruptive technologies: strategies for enhancing the learner experience and quality of MOOCs*. Preprint]. Recuperado de: http://eprints. rclis. org/19388/4/Pegagogies% 20for% 20enhanced% 20the% 20learner% 20experience% 20and% 20quality% 20of% 20MOOCs. pdf, 2013.

32. Alario-Hoyos, C., et al., *Analysing the impact of built-in and external social tools in a MOOC on educational technologies*, in *Scaling up learning for sustained impact*. 2013, Springer. p. 5-18.

33. Astels, D., *Test Driven development: A Practical Guide*. 2003: Prentice Hall Professional Technical Reference. 592.

34. Janzen, D.S. and H. Saiedian. *On the Influence of Test-Driven Development on Software Design*. in *Software Engineering Education and Training, 2006. Proceedings. 19th Conference on*. 2006.

35. Newkirk, J.W. and A.A. Vorontsov, *Test-Driven Development in Microsoft .Net*. 2004: Microsoft Press.

36. Wake, W.C., *Extreme programming explored*. 2002: Addison-Wesley Longman Publishing Co., Inc. 192.

37. Wells, D. *Extreme Programming: A gentle introduction*. 2012; Available from: http://www.extremeprogramming.org/rules/simple.html.

38. MCMILLAN, J. and J. WERGIN, *Understanding and evaluating educational research*. 2006.

39. Kaplan, B. and J.A. Maxwell, *Qualitative research methods for evaluating computer information systems*. Evaluating the Organizational Impact of Healthcare Information Systems, 2005: p. 30-55.

40. Firestone, W.A., *Meaning in method: The rhetoric of quantitative and qualitative research*. Educational researcher, 1987. **16**(7): p. 16-21.

41. Cook, T.D. and C.S. Reichardt, *Qualitative and quantitative methods in evaluation research*. Vol. 1. 1979: Sage publications Beverly Hills^ eCA CA.

42. Johnson, R.B. and A.J. Onwuegbuzie, *Mixed methods research: A research paradigm whose time has come*. Educational researcher, 2004. **33**(7): p. 14-26.

43. Creswell, J.W., *Research design: Qualitative, quantitative, and mixed methods approaches*. 2013: Sage Publications, Incorporated.

44. *Likert scale*. 2013; Available from: http://en.wikipedia.org/wiki/Likert_scale.

45. Likert, R., *A technique for the measurement of attitudes.* Archives of psychology, 1932.

46. Codometer. *Codometer- an open source tool to distribute homework assignments and automate grading of the students' submissions*. 2012; Available from: http://blog.anadolu.edu.tr/onderdemir/OpenSourceProjects/Codometer.

47. Yadav, S., et al., *Comparison of analysis techniques for information requirement determination.* Communications of the ACM, 1988. **31**(9): p. 1090-1097.

48. MOSS. *A System for Detecting Software Plagiarism*. 2012; Available from: http://theory.stanford.edu/~aiken/moss/.

APPENDIX -1 2nd Week Homework

BIM460 - Homework 2

WeatherBroadcasting

Published on 02.03.2010

Due Date 09.03.2010 - Extended to 12.03.2010

The Preliminaries

- Knowledge about Observer pattern
- Knowledge about loose coupling

The Objective

- Write three classes one is WeatherBroadCaster, second one is WeatherListenerCelcius and third one is WeatherListenerFahrenheit
- In WeatherBroadCaster there must be two methods:
    - AddWeatherListener can take observer
    - BroadCastWeather takes nothing

- • In WeatherListenerCelcius there must be three methods:
    - RegisterToWeatherBroadCaster can take observe
    - DisplayTemperature returns temperature in celcius
    - Update takes float temperature value in celcius
    - Default temperature value must be 120 (float)

- In WeatherListenerFahrenheit there must be two methods:
    - RegisterToWeatherBroadCaster can take observe
    - DisplayTemperature returns temperature in Fahrenheit
    - Update takes float temperature value in celcius
    - Default temperature value must be -10 (float)
- In every BroadCast temperature must raise by 10 (float) in celcius. Starting from 0(float).

•

How to get

add mvn dependency:

```
<dependencies>
        <dependency>
                <groupId>com.codometer.homework.bim460</groupId>
                <artifactId>homework2</artifactId><version>0.0.2-SNAPSHOT</version>
        </dependency>
</dependencies>
```

Interface to Implement

- *com.codometer.bim460.homework2.interfaces.IObserver*
- *com.codometer.bim460.homework2.interfaces.IObservee*

See your grade:

java com.codometer.homework.bim460.HomeworkTester [YourWBC full-qualified class name] [YourWLC full-qualified class name] [YourWLF full-qualified class name]

Where to put?

/bim460/[YOUR USERNAME]/hw2

example: /bim460/odemir/hw2

so your pom.xml file should be in /bim460/odemir/hw2/pom.xml and your src folder should be in /bim460/odemir/hw2/src

sample pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>com.codometer.homework.bim460</groupId>
<artifactId>homework2Impl</artifactId>
<name>homework2Impl</name>
<version>0.0.2-SNAPSHOT</version>
<build>
<pluginManagement>
<plugins>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-compiler-plugin</artifactId>
<version>2.0.2</version>
<configuration>
<source>1.6</source>
<target>1.6</target></configuration>
</plugin>
</plugins>
</pluginManagement>
</build>
<repositories>
<repository>
<id>internal</id>
<name>Archiva Managed Internal Repository</name>
<url>http://cengsvn.anadolu.edu.tr:8080/archiva/repository/internal/
</url>
</repository>
</repositories>
<dependencies>
<dependency>
<groupId>com.codometer.homework.bim460</groupId>
<artifactId>homework2</artifactId>
<version>0.0.2-SNAPSHOT</version>
</dependency>
</dependencies>
</project>
```

APPENDIX -2 2nd Week Homework Sources

IObservee.java

```
package com.codometer.homework.bim460.homework2.interfaces;

public interface IObservee {
      void AddWeatherListener(IObserver newObservee);
      void BroadCastWeather();
}
```

IObserver.java

```
package com.codometer.homework.bim460.homework2.interfaces;

public interface IObserver {
      void RegisterToWeatherBroadCaster(IObservee newObservee);
      float DisplayTemperature();
      void Update(float temp);
}
```

CodometerWeatherBroadCaster.java

```
package com.codometer.homework.bim460.homework2;


import java.util.ArrayList;

import
com.codometer.homework.bim460.homework2.interfaces.IObservee;
import
com.codometer.homework.bim460.homework2.interfaces.IObserver;

public final class CodometerWeatherBroadCaster implements
IObservee{
      ArrayList<IObserver> listeners=new ArrayList<IObserver>();
      public void AddWeatherListener(IObserver newObserver)
      {
            listeners.add(newObserver);

      }
      public void BroadCastWeather()
      {
            for(int i=0;i<listeners.size();i++)
            {

      ((IObserver)listeners.get(i)).Update(10.0f);//hep ayni
            }
      }
}
```

CodometerWeatherListenerC.java

```
package com.codometer.homework.bim460.homework2;

import
com.codometer.homework.bim460.homework2.interfaces.IObservee;
import
com.codometer.homework.bim460.homework2.interfaces.IObserver;

public final class CodometerWeatherListenerC implements IObserver
{
      float temperature=120;
      public void RegisterToWeatherBroadCaster(IObservee
newObservee)
      {
            newObservee.AddWeatherListener(this);


      }
      public float DisplayTemperature()
      {
            return temperature;
      }

      public void Update(float temp)
      {
            this.temperature=temp;
      }
}
```

CodometerWeatherListenerF.java

```
package com.codometer.homework.bim460.homework2;

import
com.codometer.homework.bim460.homework2.interfaces.IObservee;
import
com.codometer.homework.bim460.homework2.interfaces.IObserver;

public final class CodometerWeatherListenerF implements IObserver
{
      float temperature=-10;
      public void RegisterToWeatherBroadCaster(IObservee
newObservee)
      {
            newObservee.AddWeatherListener(this);
      }
      public float DisplayTemperature()
      {

            return temperature;
      }

      public void Update(float temp)
      {
            this.temperature= (9.0f/5.0f)*temp+32.0f;
      }
}
```

HomeworkTester.java

```
package com.codometer.homework.bim460.homework2;

import java.lang.reflect.InvocationTargetException;
import java.util.ArrayList;

import org.junit.Test;

import com.codometer.codepoint.CodePointEvualator;
import com.codometer.codepoint.CodePointHelper;
import com.codometer.codepoint.TestPoint;
import
com.codometer.homework.bim460.homework2.interfaces.IObservee;
import
com.codometer.homework.bim460.homework2.interfaces.IObserver;


public class HomeworkTester {


	private static String packageName;
	private static CodePointEvualator eval;
	static Class observer1Class;
	static Class observer2Class;
	static Class observeClass;
	static ArrayList<IObserver>observers;
	private IObservee CreateObservee(Class observe)
	{
		IObservee result = null;
		java.lang.reflect.Constructor co = null;
		try {
			co = observe.getConstructor(null);
		} catch (SecurityException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (NoSuchMethodException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
		 try {
			 result= (IObservee) co.newInstance(null);
		} catch (IllegalArgumentException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (InstantiationException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (IllegalAccessException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (InvocationTargetException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
		return result;
	}

	private IObserver CreateObserver(Class observer)
	{
		IObserver result = null;
		java.lang.reflect.Constructor co = null;
		try {
			co = observer.getConstructor(null);
		} catch (SecurityException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (NoSuchMethodException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
		 try {
			 result= (IObserver) co.newInstance(null);
		} catch (IllegalArgumentException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (InstantiationException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (IllegalAccessException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (InvocationTargetException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
		return result;
	}

	@TestPoint(20)
	@Test
	public void TestAddObserverCF()
	{
		IObservee observee=new CodometerWeatherBroadCaster();
		HomeworkTester.observers=new ArrayList<IObserver>();
		for(int i=0;i<10;i++)
		{
			IObserver tmp=null;
			if(i%2==0)
			{
				tmp=CreateObserver(this.observer1Class);
			}
			else
			{
				tmp=CreateObserver(this.observer2Class);

			}
			HomeworkTester.observers.add(tmp);
			tmp.RegisterToWeatherBroadCaster(observee);
			if(i==8)//son eleman hava durumunu alamamis
olacak. Digerleri alacak
			{
				observee.BroadCastWeather();
			}

		}

		for(int i=0;i<10;i++)
		{
			float expected;
			float actual;
			if(i%2==0)
			{
				expected=10.0f;

	actual=HomeworkTester.observers.get(i).DisplayTemperature();
			}
			else
			{
				expected=50.0f;

	actual=HomeworkTester.observers.get(i).DisplayTemperature();
			}
			if(i==9)
			{
				expected=-10.0f;

	actual=HomeworkTester.observers.get(i).DisplayTemperature();
			}

	CodePointEvualator.getInstance().assertEquals(expected, 
actual);
		}
	}


	@TestPoint(20)
	@Test
	public void TestAddObserverF()
	{
		IObservee observee=new CodometerWeatherBroadCaster();
		HomeworkTester.observers=new ArrayList<IObserver>();
		for(int i=0;i<10;i++)
		{
			IObserver tmp=null;

			tmp=CreateObserver(this.observer2Class);


			HomeworkTester.observers.add(tmp);
			tmp.RegisterToWeatherBroadCaster(observee);
			if(i==8)//son eleman hava durumunu alamamis 
olacak. Digerleri alacak
			{
				observee.BroadCastWeather();
			}

		}

		for(int i=0;i<10;i++)
		{
			float expected;
			float actual;
			expected=50.0f;

	actual=HomeworkTester.observers.get(i).DisplayTemperature();
			if(i==9)
```

```
                  {
                        expected=-10.0f;

      actual=HomeworkTester.observers.get(i).DisplayTemperature();
                  }

      CodePointEvualator.getInstance().assertEquals(expected,
actual);
            }
      }


      @TestPoint(20)
      @Test
      public void TestAddObserverC()
      {
            IObservee observee=new CodometerWeatherBroadCaster();
            HomeworkTester.observers=new ArrayList<IObserver>();
            for(int i=0;i<10;i++)
            {
                  IObserver tmp=null;

                  tmp=CreateObserver(this.observer1Class);


                  HomeworkTester.observers.add(tmp);
                  tmp.RegisterToWeatherBroadCaster(observee);
                  if(i==8)//son eleman hava durumunu alamamis
olacak. Digerleri alacak
                  {
                        observee.BroadCastWeather();
                  }

            }

            for(int i=0;i<10;i++)
            {
                  float expected;
                  float actual;
                  expected=10.0f;

      actual=HomeworkTester.observers.get(i).DisplayTemperature();
                  if(i==9)
                  {
                        expected=120.0f;

      actual=HomeworkTester.observers.get(i).DisplayTemperature();
                  }

      CodePointEvualator.getInstance().assertEquals(expected,
actual);
            }
      }

      @TestPoint(40)
      @Test
      public void TestObservee()
      {
            IObservee observee=CreateObservee(this.observeClass);
            IObserver observer1=new CodometerWeatherListenerC();
            IObserver observer2=new CodometerWeatherListenerC();
            IObserver observer3=new CodometerWeatherListenerC();
            IObserver observer4=new CodometerWeatherListenerF();
            observee.BroadCastWeather();

            observer1.RegisterToWeatherBroadCaster(observee);
            observer2.RegisterToWeatherBroadCaster(observee);
            CodePointEvualator.getInstance().assertEquals(10.0f,
observer1.DisplayTemperature());
            observee.BroadCastWeather();
            CodePointEvualator.getInstance().assertEquals(20.0f,
observer2.DisplayTemperature());
            observer3.RegisterToWeatherBroadCaster(observee);
            observee.BroadCastWeather();

            IObservee
anotherObservee=CreateObservee(this.observeClass);

      observer4.RegisterToWeatherBroadCaster(anotherObservee);

      observer1.RegisterToWeatherBroadCaster(anotherObservee);
            anotherObservee.BroadCastWeather();

            CodePointEvualator.getInstance().assertEquals(10.0f,
observer1.DisplayTemperature());
            CodePointEvualator.getInstance().assertEquals(30.0f,
observer2.DisplayTemperature());
            CodePointEvualator.getInstance().assertEquals(30.0f,
observer3.DisplayTemperature());
            CodePointEvualator.getInstance().assertEquals(50.0f,
observer4.DisplayTemperature());


      }

      public static void RunGrader() {


CodePointEvualator.getInstance().PuanlamayaBasla(HomeworkTester.cl
ass.getName());
            System.out.println();
            System.out.println();
            System.out.println();
            System.out.println("Toplam: "+
CodePointEvualator.getInstance().getOverall());
            System.out.println("Alinan: "+
CodePointEvualator.getInstance().getPointsTaken());
            System.out.println("Basari: %"+
CodePointEvualator.getInstance().normalizeTo100());
      }
//
      //First is observer class name second and third is observees


      public static boolean init(String packageName) {
      Class []classes=null;
            try{
                  classes=CodePointHelper.getClasses(packageName);
                  for (Class c : classes) {
                        //System.out.println(c.getName());
                        if (c.newInstance() instanceof IObservee)
                        {
                              observeClass = c;
                        }
                        else if (c.newInstance() instanceof
IObserver)
                        {
                              if(((IObserver)
c.newInstance()).DisplayTemperature()==120.0f)
                              {
                                    HomeworkTester.observer1Class
= c;
                              }
                              else if(((IObserver)
c.newInstance()).DisplayTemperature()==-10.0f)
                              {
                                    HomeworkTester.observer2Class
= c;
                              }

                        }

                  }

                  if (observeClass==null) {
                        System.err.println("I can't find your
Wheather Broadcaster");
                        return false;
                  }
                  if (observer1Class==null) {
                        System.err.println("I can't find your
Wheather Broadcaster Listener For Celcius");
                        return false;
                  }
                  if (observer2Class==null) {
                        System.err.println("I can't find your
Wheather Broadcaster Listener For Fahrenheit");
                        return false;
                  }
                  else
                  {

                  }
            }
            catch(Exception ex)
            {
                  System.err.println(ex.getLocalizedMessage());

            }
            return true;
    }
      public static void main(String[] args)
      {



            try {
                  packageName=args[0];
                  System.out.println(packageName);
                  eval=CodePointEvualator.getInstance();
                  if(init(packageName))
                  {
                        HomeworkTester.RunGrader();
                  }


//                HomeworkTester.observeClass =
Class.forName(args[0]);
//
      HomeworkTester.observer1Class=Class.forName(args[1]);
//
      HomeworkTester.observer2Class=Class.forName(args[2]);

            } catch (Exception e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            }



      }
}
```