BSM 261: Object Oriented Programming
Programming Project 1

Due date: Friday, November 11 at 11:59PM

1 Overview

In this assignment, you will be implementing code for investment account for a customer. This class will keep track of owned cash, savings, and loan. To keep things somewhat simple, you will implement the processing for a savings account and a loan. The real purpose of the assignment is to give you practice writing classes, fields, methods, conditional statements, and constructors. You will also be introduced to polymorphism.

Please submit your assignment to the ubs as a single zipfile containing: Date.java, Cash.java, Savings.java, Loan.java, Customer.java, and your testing report.

NOTE: There are technical terms. You do not have to understand the meaning of the terms. The project instructions will tell you where to set a value and where to use a value so that you can code this assignment without knowing anything about these terms. This is a fairly common situation in the real world. A software engineer will work to understand the customer's needs, and design the instructions (or project specifications) so the software developers can write the program without necessarily understanding everything about the business.

2 Code Readability (20% of your project grade)

Back in time, the only goal of the programming was to get a program working. There was also less number of programmers and the complexity of the projects was not that complicated. Now, with highly complex software running much of our lives, the industry has learned that computer code is a written document that must be able to communicate to other humans what the code is doing. If the program is too hard for a human to quickly understand, the industry does not want it.

The companies in the market enforces strict rules about how the program should look. This rules are not fixed for all companies. In our class will do the same, but will not be quite as strict so you can have some freedom for developing your own style.

To receive the full readability scores, your code must follow the following guideline:
• All variables (fields, parameters, local variables) must be given appropriate and descriptive names.
• All variable and method names must start with a lowercase letter. All class names must start with an uppercase letter.
• The class body should be organized so that all the fields are at the top of the file, the constructors are next, and then the rest of the methods.
• Every statement of the program should be on it's own line and not sharing a line with another statement.
• All code must be properly indented (see Appendix F of the Lewis book for an example of good style). The amount of indentation is up to you, but it should be at least 2 spaces, and it must be used consistently throughout the code.

• You must be consistent in your use of {, }. The closing } must be on its own line and indented the same amount as the line containing the opening {.
• There must be an empty line between each method.
• There must be a space separating each operator from its operands as well as a space after each comma.
• There must be a comment at the top of the file that includes both your name and a description of what the class represents.
• There must be a comment directly above each method that, in one or two lines, states what task the method is doing, not how it is doing it. Do not directly copy the homework instructions.
• There must be a comment directly above each field that, in one line, states what the field is storing.
• There must be a comment either above or to the right of each non-field variable indicating what the variable is storing. Any comments placed to the right should be aligned so they start on the same column.

Here is the readability rubric:

Points	Metric
20	Excellent, readable code. Correct comments at top of class file, above each method and above each field. Each comment at most one or two sentences and is a good description. Class has fields first, then constructors, then the remaining methods. All code follows the proper indentation and naming style.
18	Very readable code. There are only a few places (at most 5) where there is inconsistent or missing indentation or poorly named fields/methods, or missing comments, or comments that are not good descriptions, or fields or constructors buried inside the code or missing whitespaces.
15	Reasonably readable code. Overall good commenting, naming, and indentation. However, there are more than 5 places with missing or inconsistent indentation, poor variable names, or missing, unhelpful, or very long comments OR many comments that are a direct copy of the homework description.
12	Poorly readable code. A significant number of missing comments or inconsistent and/or missing indentation or a large number of poorly named variables. However, at least half of the places that need comments have them.
10	Some commenting done (close to half), but missing majority of the comments.
5	There is at least five useful comment in the code OR there is some reasonable indentation of the code.
0	Code very unreadable. No useful comments in the code AND/OR no indentation in the code.

3 Program Testing Document (20% of your project grade)

The current softwares are very complex and one minor error might even cause a death. The software glitches starting to become a major issue and destroying companies. Now, standard practice is that all code must be thoroughly verified before a company is willing to release it. Even some companies are requiring the programmers to write the test cases before even start programming. In this class, we will not be that strict, but you will need to test your code.

To receive full testing marks, you must write a testing report that shows that you thoroughly tested every method of the program. The report should be a short English or Turkish description for each test (what you are testing and what the expected result of the test is) followed by the actual result of the test. **If you are using DrJava, you can enter the test into the interactions pane and then copy and paste the test code plus the result to your report.** If you fail to complete the program (this will not result in point deduction on your testing or readability), your report should indicate how you would go about testing the incomplete methods.

Your grade on the testing report is how thoroughly you test your code, not how correctly your

code runs. If your code is not 100% correct then your report should show an incorrect result to some test. Testing methods that do not have conditional statements should be pretty straightforward, but you need to put thought into testing methods with conditional statements so that each branch of the if-statement is tested.

Hint 1: You can test multiple methods with one test. For example, you can test each setter/getter method pair together or you can test constructors and getter methods together.
Hint 2: Do not put off testing to the end! est each method after you complete it. Many methods depend on other methods. Delaying testing could mean cascading errors that cause your whole project to collapse. Since you need to test anyway, copy the tests you do into a document, and you are most of the way to completing your report.

If you are not using DrJava, you are allowed (but not required) create a separate class that tests your program. You must still write a testing report that documents the tests you do in this class. Do not place testing code into a main method of the classes below. That is not the purpose of a main method.
Here is the testing rubric:

Points	Metric
20	Excellent demonstration of how to test the program. Completed tests for each working method of the student's submission, either on its own or as part of a related "unit" of the program. Good descriptions of how to test the parts of the submission that are not working. Tests of conditional statements cover the different possible execution paths. Each test gives the true output of the student's submission. A note on any test that gives an incorrect result.
18	Very good: Good descriptions, tests covers most of the program but a few tests were missed.
15	Reasonable demonstration of how to test the program. The tests performed and the testing description covers a majority of the program; however, multiple obvious cases were missed.
12	Poor demonstration of how to test the program. The testing report shows how to test some of the methods and conditional statements, but at least half of the needed tests are missing.
10	Some testing done (at least half), but nothing that demonstrates the proper way to test a conditional statement.
5	There is at least five proper testing of the methods/constructors in the code.
0	The testing report does not match the behavior of the student's code.

4 Programming (60% of your grade)

Guidelines for the programming part:

• Unless specifically indicated, the listed methods must be public instance methods.
• You will need to create several instance fields to store data, and every field must be private.
• All fields must be initialized to an appropriate value. They can be initialized either as part if the field declaration or in the constructor. Even if you feel that the default value provided by Java is appropriate, you still must give an explicit initialization.
• Any method whose name begins with set should only assign a value to an appropriately named field. The method should do no other processing. Any processing described in a set method description below is for information only. That actual processing will be done by other methods.
• Any method whose name begins with get should only return the appropriate value. No other processing should occur in these methods.
• Your class must include only the methods listed. You may not write any other methods.
• The behavior of your methods must match the descriptions below.
• You should not write any loops in your program (though loops are allowed in the testing code).

For the programming part, create the following classes that will keep track of a customer's cash, savings, and the loan :

4.1 Date class:

The Date class will represent a date. (Java has both Date and Calendar in the API, but both are more complex than needed for this homework.) A Date contains a day, month, and a year.

Constructor: The Date class should have one constructor. It has input **int** day, **int** month, **int** year and initializes the Date object with the given inputs. You may assume that the inputs are all valid values (checking if the given input values form a proper Date is optional).

Methods: The Date class should have the following methods:
• **getDay:** takes no input and returns an int that is the day of the date. The day should be a value between 1 and 31.
• **getMonth:** takes no input and returns an int that is the month of the date. The month should be between 1 and 12.
• **getYear:** takes no input and returns an int that is the year of the date.
• **incrementDay:** takes no input and returns nothing. The method adds 1 to the day of the date. If the day exceeds the number of days for the month (assume no leap years), the day is set to 1 and the month is incremented. If the month exceeds 12, the month is set to 1 and the year is incremented.
• **toString:** overrides the toString method inherited from Object so that the string returned is the date in "xx/xx/xxxx" format.
• **equals:** overrides the equals method inherited from Object so that the method returns true if the input is a Date object (our Date, not the API Date) with the same month and day (the year may be different). Otherwise the method returns false.

4.2 Cash class:

The Cash class represents an account that deals in money. The Cash class will have a balance, an interest rate, and a record of the amount of interest earned.

Constructor: The Cash class should have a single constructor that takes a **double** (the interest rate) as input.

Methods: The class should have the following methods:
• **getBalance:** takes no input and returns a double that is the current balance in the account.
• **getInterestRate:** takes no input and returns a double that is the interest rate.
• **setInterestRate:** takes a double as input and returns nothing. Changes the interest rate.
• **transfer:** takes a double as input and returns nothing. Reduces the current balance by the input amount.
• **getInterestEarned:** takes no input and returns a double that is the amount of interest accrued this month.
• **processDay:** takes no input and returns nothing. If the account balance is positive, multiplies the balance by the interest rate (divided by 365) and adds the amount to the current monthly interest.
• **processMonth:** takes no input and returns nothing. Adds the current monthly interest to the balance and sets the current monthly interest to zero.

4.3 Savings class:

The Savings class is a **Cash** class that represents money deposited or earned by the customer. The Savings class has the same properties and behaviors of the Cash class plus the following two additional methods.

Constructor: The Savings class should have a single constructor that takes a **double** (the interest rate) as input and sets the interest rate. (**Hint:how can you use super()?**)

Methods: The class should have the following methods:
• **deposit:** takes a double as input and returns nothing. Increases the balance by the input amount.
• **withdraw:** takes a double as input and returns a boolean. If the input amount is less than or equal to the current balance, reduces the balance by the input amount and returns true. Otherwise returns false and makes no change to the balance.

4.4 Loan class:

The Loan class is a **Cash** class the represents money borrowed by the customer. The Loan class has all the same properties and behaviors of the Cash class in addition to the loan limit, the overdraft penalty as well as whether the loan is overdrafted. The class should have all the methods of Cash except the processDay and processMonth will need additional behavior as listed below and there will be four additional methods.

Constructor: The Loan class should have a single constructor that takes three double inputs: the interest rate, the loan limit, and the overdraft penalty.

Methods: The class should have the following methods:
• **getLoanLimit:** takes no input and returns a double that is the loan limit.
• **setLoanLimit:** takes a double as input and returns nothing. Changes the loan limit.
• **getOverdraftPenalty:** takes no input and returns a double that is the overdraft penalty.
• **setOverdraftPenalty:** takes a double as input and returns nothing. Changes the overdraft penalty.
• **processDay:** takes no input and returns nothing. This method should have the same behavior as in the Cash class plus if the current balance exceeds the loan limit, then the method should record that the loan is overdrafted for this month.
• **processMonth:** takes no input and returns nothing. This method should have the same behavior as in the Cash class plus if the loan was ever overdrafted since the last time processMonth was called, then the overdraft penalty should be added to the balance.

4.5 Customer class:

The Customer class represents a customer account. The Customer class will have a customer name (first and last), the customer's savings, the customer's loan, the commission, and the date. The class should have the following methods.

Constructor: The Customer class should have one constructors that sets the first name, the last name, the savings, the loan, and the date of the account.

Methods: The class should have the following methods:
• **getFirstName:** takes no input and returns a String that is the first name associated with the account.
• **setFirstName:** takes a String as input and returns nothing. Changes the first name associated with the account.
• **getLastName:** takes no input and returns String the last name associated with the account.
• **setLastName:** takes a String as input and returns nothing. Changes the last name associated with the account.
• **getSavings:** takes no input and returns a Savings that is the savings account associated with this customer. The savings account will not change.
• **getLoan:** takes no input and returns a Loan that is the loan associated with this customer. The loan will not change.
• **getDate:** takes no input and returns a Date instance.
• **setDate:** takes a Date as input and returns nothing. Changes the Date on the account.
• **currentValue:** takes no input and returns a double that is the Savings balance subtracted by the Loan balance.
• **deposit:** takes a double as input and returns nothing. Adds the input amount to the savings balance.

• **payLoan:** takes a double as input and returns nothing. Reduces the loan balance by the input amount.
• **withdraw:** takes a double as input and returns a boolean. If the input amount is less than or equal to the savings balance, reduces the savings balance by the input amount and returns true. Otherwise returns false and makes no change to the savings balance. (**Hint: how much do you need to code?**)
• **incrementDate:** takes no input and returns nothing. Calls the associated method of the Date class to increment the date. If the savings balance is negative, transfer that balance to the loan (increasing the balance of the loan), and set the savings balance to zero. Then call the processDay method of both the savings and loan instances. Finally, if the month changed as a result of the increment, call the processMonth method of the savings and loan instances.

Programming part of the assignment will have 4 main parts including Fields/Getter/Setter methods, Constructors, Object-Oriented coding, and Non-Getter/Setter methods. Here is the programming rubric:

Fields/Getter/Setter Methods: 20 points

Points	Metric
20	Perfect design and implementation of fields and getter/setter methods. All necessary fields and getter/setter methods provided. All fields are private and the getter/setter methods are public. All fields are given explicit initial values either at the declaration or in the constructor. All getter/setter methods only set or retrieve the field values. There are no extraneous fields. There are no static fields or methods.
18	Very good design and implementation of fields and getter/setter methods. In almost all cases, the above features are present in the fields and getter/setter methods. However, there are a couple places where the above features are missing (for example a couple missing methods, an extraneous field, or a couple fields that do not get initial values) OR there is some single error type that shows up in the code (for example, most fields and getter/setter methods are correct, but a significant number of fields are missing explicit initial values).
15	Reasonable design and implementation of fields and getter/setter methods. A majority of the fields and getter/setter methods meet the above criteria, but a significant number have errors and there are multiple types of errors OR all the fields and getter/setter pairs are correct except that most or all have the same type of error (for example, everything is correct except that most of the fields and methods are static).
10	Poor design and implementation of fields and getter/setter methods. A majority of the fields and getter/setter methods have errors, but there are a significant number that are correctly implemented. Or, most of the fields and getter/setter methods are correctly implemented except for two or three consistent errors (for example, all fields are public, static, and not given initial values).
5	There is at least one case of a private, non-static required field that has an explicit initial value and proper non-static getter/setter methods for the field.
0	Code does not demonstrate proper use of fields and getter/setter methods. There is no case of a private, non-static, required field that has an explicit initial value and proper non-static getter/setter methods for that field.

Constructors: 10 points

Points	Metric
10	Perfect design and implementation of constructors.
9	Very good design and implementation of constructors. The constructors work correctly but have extra unnecessary operations or duplicate computation.
7	Reasonable design and implementation of constructors. At least one class has a correct constructor.
5	Poor design and implementation of constructors. All constructors fail to properly set field values OR the methods are not really constructors because they are incorrectly given return values or incorrect names.
0	No constructors provided.

Object-Oriented Coding: 10 points

Points	Metric
10	Perfect object-oriented design. Classes correctly use inherited methods and correctly override methods (if needed). There are no extraneous methods. There are no unnecessary fields storing duplicate data. Correct use of this and super. The code (other than the constructor) uses getter/setter methods when available and not the fields directly.
9	Very good object-oriented design. Classes correctly use inherited methods and correctly override methods (if needed). There are no extraneous methods. There are no unnecessary fields storing duplicate data. Correct use of this and super. The code may incorrectly use fields directly instead of the available getter/setter methods. The code may incorrectly use public getter/setter methods in the constructors.
7	Reasonable object-oriented design. In multiple places, the code uses inherited methods. There are some places where the inherited and/or overridden methods are not done correctly leading to a few extra fields or unnecessary methods.
5	Poor object-oriented design. There is some use of inherited methods, but the code does not properly use the inherited methods or properly override the methods (if needed) so there are many unnecessary extra fields or methods.
0	No object-oriented design. There is no use of inherited methods or overriding of inherited methods.

Non-getter/setter Methods: 20 points

Points	Metric
20	Perfect logic and coding. All methods present and correct. There is no unnecessary operations or a significant amount of unnecessary duplicated code.
18	Very good logic and coding. All methods that require conditional statements have well-formed statements for the problem. All the methods that do not require conditional statements are correct or have have correct logic but only minor errors.
15	Reasonable logic and coding. At least half of the methods that require conditional statements have well-formed statements, possibly with minor errors. At least half of the methods that do not require conditional statements have the correct mathematical logic, possibly with minor errors.
10	Poor logic and coding. There are at least two cases of methods that require conditional statements that have well-formed statements, possibly with minor errors.
5	At least one non-getter/setter method is either correct or at least has a correctly formed conditional statement for the problem.
0	No non-getter setter method is correct and no method that requires a conditional statement has a correctly formed conditional statement.

Penalties

Deduction	Cause
-10	The submitted code does not compile.
-10	Code that contradicts the rules of the assignment (example: using loops in the non-testing portion of the assignment).

5 Course Honor Policy

See the university policy on academic integrity for general rules that you should follow on tests and quizzes. Rules specific to programming in this class are listed here.

Programming is a collaborative enterprise. However, you cannot be an equal collaborator until you first build up your own programming skill set. Because of that, it is essential that you do programming project coding on your own. That does not mean that you can not seek help or give help to other

students, but the assistance must be at a high level - for example English descriptions of how to solve a problem and not coding descriptions. The most important skill you will develop in this class is to translate a solution description into the Java commands needed to implement the solution. If you copy code from other sources, you are not developing this needed skill for yourself.

Here is a short list of things you may and may not do:
• **DO NOT** send an electronic copy of your code to another student. (Exception: You may share with your lab partner a copy of the code you wrote together during the recitation section.)

• **DO NOT** look at another person's code for the purpose of writing your own.

• **DO NOT** tell another student the Java code needed for an assignment.

• **DO** sit down with a student and go over the student's code together in order to help the student find a bug.

• **DO** describe in English (not code!) the steps needed to solve an assignment problem.

• **DO** show students how to do certain Java coding tasks as long as the task does not directly relate to a homework question.

• **DO** use coding examples from the lecture and textbook as a guide in your programming.

• **DO NOT** search the internet to find the exact solutions to assignment problems.

• **DO NOT** post code you write for this class onto an internet site such as StackExchange.

• **DO NOT** post the homework specific problem descriptions onto internet sites such as StackExchange.