CS 116

HW Assignment # 6: Inheritance

- **Points: 9**
- **Submission**
  - **Deadline: Friday 04/20 11:59 PM**
  - **Submit on Blackboard under Submissions->HW Assignments ->HW6. Please make sure that you click the "Submit" button and not just "Save".**
- **Late Submission Policy**
  - **You can do a late submission until Monday 04/23 11:59PM with a 5% penalty on the total points for this assignment.**
  - **After that solutions will be posted and no submission will be accepted**
- **Early Submission**
  - **You can also get 5% extra point on your score on the assignment for early submission if you submit by Thursday 04/19 11:59PM.**
- **Getting help**
  - **From instructor during office hours or by email.**
  - **By seeing one of the TAs during the listed TA office hours.**
  - **By visiting the ARC (Academic Resource Center).**
- **Academic Dishonesty Policy**
  - **Working with a partner: You can work with a partner as assigned by the instructor. Otherwise this should be considered individual work.**
    - **Even if you are working with a partner, you and your partner are required to make individual submissions.**
  - **Please note: In case two submissions are declared identical (and if you are not supposed to work together) the excuse: we worked together, does not hold and both submission will be treated according to ethics rules.**

Note: Some details in this assignment has been left deliberately vague. You have to make decision on how best to implement certain features.

Note: Remember that you should follow good software engineering principles (without being explicitly mentioned in the assignment) such as having appropriate accessor and mutator methods.

PROGRAMMING TASK A: Sanchez Construction Loan Co.[4pts]

Sanchez Construction Loan Co. makes loans of up to $100,000 for construction projects. There are two categories of *Loans* those to businesses and those to individual applicants.

Write an application that tracks all new construction loans. **The application must also calculate the total amount owed at the due date (original loan amount + loan fee (simple interest accrued)).** The application should include the following classes:

- **Loan**: A public abstract class that implements the *LoanConstants* interface. A Loan includes a loan number (unique ID), customer last name, amount of loan, interest rate, and term. The constructor requires data for each of the fields except interest rate. Do not allow loan amounts over $100,000. Force any loan term that is not one of the three defined in the *LoanConstants* class to a short-term, one-year loan. Create a *toString()* method that displays all the loan data.
- **LoanConstants**: A public interface class *LoanConstants* includes constant values for short-term (1 year), medium-term (3 years), and long-term (5 years) loans. It also contains constants for the company name and the maximum loan amount. (Interface can have variables. Check out - what type of variables?)
- **BusinessLoan**: A public class that extends Loan. The *BusinessLoan* constructor sets the interest rate to 1% over the current prime interest rate.
- **PersonalLoan**: A public class that extends Loan. The *PersonalLoan* constructor sets the interest rate to 2% over the current prime interest rate.
- **CreateLoans**: An application that creates an array of five *Loan* objects. Prompt the user for the current prime interest rate. Then, in a loop, prompt the user for a loan type and all relevant information for that loan. Store the created *Loan* objects in the array. When data entry is complete, display all the loans.

Part A Deliverables

- Save the files as Loan.java, LoanConstants.java, BusinessLoan.java, PersonalLoan.java, and CreateLoans.java.
- Create test cases (use the table format given below) to test your classes. Store the test plan in a .doc file called "PartA-test.doc". You should run each test case. Comment out the test cases inputs in your client class if you have to.
-

Test Case	Output	Tested?

- Store all the files for Part A (including the test plan document) in a sub-folder called PartA.

PROGRAMMING TASK B: Card Games[3pts]

Card games have a couple of things in common. For example, cards are shuffled before they are dealt and the players are dealt a specific number of cards. In this exercise, you will encapsulate common things about a card

game in a superclass, which is then inherited by sub-classes that represent the specific card games (along with the game rules such as number of cards dealt).

a. Create an abstract CardGame class. The class contains a "deck" (it is up to you to decide whether you want to implement the deck using array, vector, etc.) of 52 playing cards that uses a Card class that holds a suit and value for each Card object. It also contains an integer field that holds the number of cards dealt to a player in a particular game. The class contains a constructor that initializes the deck of cards with appropriate values (e.g., "King of Hearts"), and a shuffle() method that randomly arranges the positions of the Cards in the array. The class also contains two abstract methods: displayDescription(), which displays a brief description of the game in each of the child classes, and deal(), which deals the appropriate number of Card objects to one player of a game. Save the file as CardGame.java.

b. You will now utilize CardGame class to deal cards for two games: Poker and Bridge. Create two child classes (for Poker and Bridge respectively) that extend CardGame. Create a constructor for each child class that initializes the field that holds the number of cards dealt to the correct value. For example, in standard poker, a player receives five cards, but in bridge, a player receives 13. Create an appropriate displayDescription() and deal() method for each child class. Save each file using an appropriate name—for example, Poker.java or Bridge.java.

c. Create an application that instantiates one object of each game type and demonstrates that the methods work correctly. Save the application as PlayCardGames.java.

Part B Deliverables

- Save the files as Poker.java, Bridge.java, CardGame.java, Card.java and PlayCardGames.java.
- Create test cases (use the table format given in Part A) to test your classes. Store the test plan in a .doc file called "PartB-test.doc". You should run each test case. Comment out the test cases inputs in your client class if you have to.
- Store all the files for Part B (including the test plan document) in a sub-folder called Part B.
- Remember to place Part A and Part B in your assignment folder.

Submission instructions

- In your submission you must include:
    a. The source code files and the compiled files for the program.
- Zip all files and name the zip file using your last name followed by your first name followed by the name of the assignment
    i.e. Doe_Jane_Lab5.zip
- Upload the file on assignment folder: Submissions -> HW assignments -> HW6 on Blackboard.