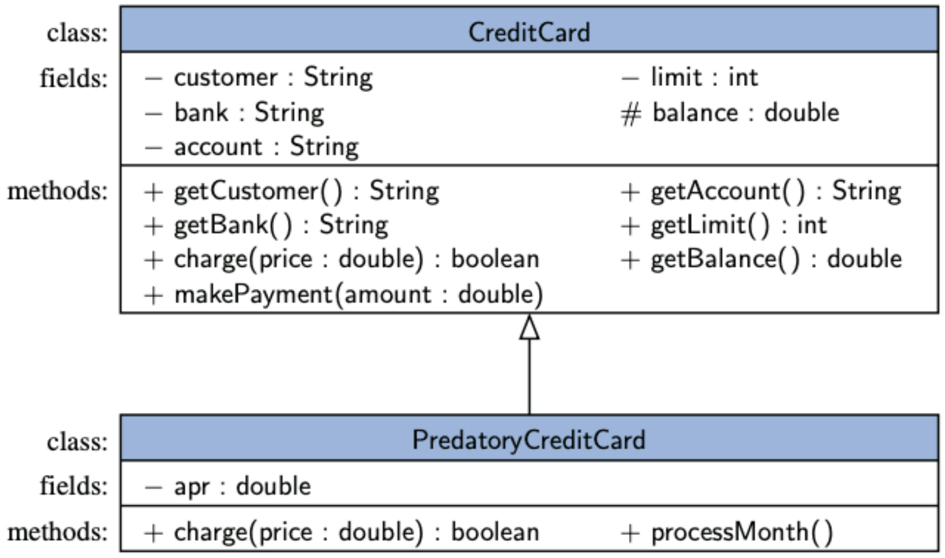
CS171 Lab 1 (Ungraded)
Object-Oriented Programming (OOP) in Java: Inheritance

Background: As we discussed in class, *inheritance* is an important pillar of OOP that allows us to design the different components of a software package using a hierarchical approach. We start by grouping common properties and behaviors in a ***parent*** class (aka ***super*** class). Then, we can create ***subclasses*** that inherit these fields and methods from the superclass (enabling code reuse!), while also being able to add their own custom properties. More specifically, subclasses can ***augment*** the superclass by adding new fields and new methods, and/or can ***override*** existing behaviors by providing a new implementation for inherited methods.

Objective: This lab exercise is designed to give you hands-on experience with implementing and using inheritance in Java. You will be given a hierarchical design that involves multiple classes and a list of required features, and your job is to provide a complete Java implementation of this design.

Lab Description:
Remember the `CreditCard` example that we covered in class when we first introduced OOP? Turns out, the bank is now interested in offering another line of credit cards that supports some additional features, compared to the regular credit card. And, for lack of a better name, they named the new type of cards `PredatoryCreditCard`. Your job is to implement a class that represents this new type.

The figure below provides a UML diagram that serves as an overview of our design for the new `PredatoryCreditCard` class as a subclass of the existing `CreditCard` class. The hollow arrow in that diagram indicates the use of inheritance, with the arrow oriented from the subclass to the superclass:



The `PredatoryCreditCard` class ***augments*** the original `CreditCard` class, adding a new instance variable named apr to store the annual percentage rate, and adding a new method named processMonth that will assess interest charges. The new class also ***specializes*** its superclass by ***overriding*** the original charge method in order to provide a new implementation that adds a $5 fee for an attempted overcharge.

Note: In a UML Class Diagram, the "**+**" symbol indicates that a field or method are public, while "**–**" indicates that they're private, and "**#**" indicates that they're protected.

Discussion: To make sure you understand this design and to help you put together the plan for your code, let's first answer some questions together:

1. What **instance variables** does subclass `PredatoryCreditCard` inherit from superclass `CreditCard`?

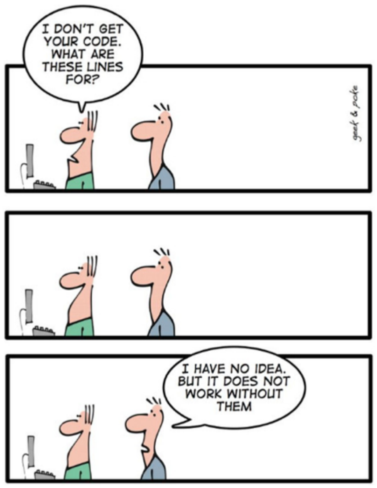
2. What **methods** does subclass `PredatoryCreditCard` inherit from superclass `CreditCard`?

3. How does the subclass `PredatoryCreditCard` **augment** the original `CreditCard` class (i.e., what instance variables and methods does it add)?

4. How does the behavior of the subclass `PredatoryCreditCard` **override** the original `CreditCard` class?

Few additional things to consider:

- How do you indicate that `PredatoryCreditCard` is a subclass of `CreditCard`?
- What parameters should `PredatoryCreditCard's` constructor(s) take?
- Do you need to explicitly invoke any *superclass* methods in your subclass `PredatoryCreditCard`? Why, where, and how?


I DON'T GET YOUR CODE. WHAT ARE THESE LINES FOR?
I HAVE NO IDEA. BUT IT DOES NOT WORK WITHOUT THEM
geek & poke

[8 points] Lab Question 1: After reading (and answering) all the questions in the discussion above, you are now ready to provide a complete implementation of the class `PredatoryCreditCard`. You got this!

If you face any issues or have questions, please feel free to discuss with classmates or ask the instructor/TA.

[1 point] Lab Question 2: Additionally, we want to have an elegant way of printing a string representation of any `PredatoryCreditCard` object, by passing the object to Java's standard println method. Override method toString() in class `PredatoryCreditCard` to provide an elegant string representation of a `PredatoryCreditCard` object. Print customer name, bank name, account identifier, credit limit, current balance, and the annual percentage rate associated with this card.

[1 point] Lab Question 3: Can we easily find out if two different cards of `PredatoryCreditCard` are *equal*? Override method equals(Object o) in class `PredatoryCreditCard` to return true if two cards (i.e., the current object and the passed parameter object) have identical *account* identifiers and *balance* amounts, false otherwise.