Lab 2: Designing Methods

Software provided

- class `Soda` with two methods: `isBig` and `sameBrand`
- class `Letter` and class `Ad` with examples, but no methods

Goals

- Develop methods for a simple class — learn how to organize work.
- Develop methods for a class that contains member data that is an instance of another class.
- Develop methods for a class hierarchy with abstract class and derived classes.
- QUIZ: Demonstrate understanding of class definitions

Details

- Methods for a single class `Soda`
  - Open up the file **soda.java**. Read the existing code and observe its organization
    - class definition first
    - fields (member data) definitions
    - the constructor
    - template as a *Java Comment Box* - initially with all fields, later we add each method as it is developed
    - method definitions
    - examples of class instances (after the class closing brace) in *Java Interactions Boxes*
    - *Java Interactions Box* that indicates the start of method tests
    - *Java Interactions Boxes* with method tests: if possible written in such way that they only produce `true`
  - Add to the class Soda a method `unitPrice` that determines the price in cents per one ounce of soda. Follow the design recipe, and keep the structure of the code.

  - Add to the class Soda a method `betterPrice` that determines whether the unit price of one soda is better than the unit price of some other soda. Follow the design recipe, and keep the structure of the code. Remember to use the template as needed.

- Methods for a class with containment.

  - Save the work on class `Soda`, then start again.
  - Modify class `Soda`, so that it refers to `Brand` class, where the brand besides the name includes the information whether this soda contains caffeine and whether this is diet soda. Make new examples of class instances for both classes.
  - Modify the method `sameBrand` in the class `Soda` to reflect the new class design. Make sure the code for the class `Soda` does not have to change if we decide to change the way we record the information about the brand.
    Hint: Follow the principle of single point of control.

- QUIZ (10 minutes, closed notes, closed monitor)

- Methods for unions of classes

  - Download the code for the classes that represent mail items — mail.java.
    Currently the two classes represent a letter and a piece of advertisement.
  - Define a new class for items that can be mailed: class Package. In addition to the weight and postage rate, these items can be insured for some dollar value. The insurance fees are given as cost of insurance for each $100 of value.
  - The postage for all items is determined as follows:
    * The base price is computed by multiplying the `weight` by `rate`.
    * For letters, we add the fees as given.
    * For packages, we add the cost of insuring the package.
  - Design abstract class `AMail` to represent there three kinds of mail items and modify the three class definition to extend this class
  - Design the method `basePrice` — a concrete method in the abstract class — that computes the base postage for all mail items.

– Design the method `actualPrice` for this class hierarchy. It must be abstract in the class `AMail` and have concrete implementation in all derived classes.
– Design the method `isCheaper` that determines whether one piece of mail requires less postage than another piece of mail.