# Midterm 1 Sample (Part 2)[*]

CSS 162—Computer Programming II

## Directions

This part of the exam is closed book and closed notes. No books, notes, electronic devices, or any material or information is allowed. Write neatly and use proper syntax. Partial credit may be awarded. It is highly recommended that you write supplementary explanations if you cannot produce a final answer.

Each question is worth 10 points. Please answer the questions on separate sheets of paper, **one question per sheet of paper.** Please turn in this exam sheet with your answer sheets.

## Questions

1. Consider a `GermanCar` class that stores the make (e.g., Porsche) and model (e.g., 928) of an automobile manufactured by a company based in Germany. Both variables are stored as `String`s. List two class invariants for legitimate instances of `GermanCar` and write two tests for those invariants (i.e., one test for each invariant) and embed those tests into a testing harness (name the class the testing harness is in `TestingHarness`).

   **Do not write the `GermanCar` class!** You **cannot** assume the test methods you write are part of the `GermanCar` class; they are part of the testing harness class. You may assume the methods `getMake` and `getModel` exist in `GermanCar` objects and return the values of the make and model, respectively, and that the `GermanCar` class has a two-argument constructor that accepts initial values for the make and model.

   Your tests should be written/called in such a way that it conforms to the best-practices for testing harnesses that we described. That is to say, you are to write a `main` method in which you execute your tests, and this method should execute your tests in a way that conforms to best-practices. You may put your tests in other methods outside of the `main`, and call them in `main`, but you do not have to.

2. Write a class `GameStore` that will be used to track the number of games sold and the number of customers at a store that sells games. Your class should have the following public methods:

[*]As of April 28, 2017. Questions are written by Johnny Lin.

- A constructor that takes a parameter (of type `int`) indicating how many games are in the initial inventory of the store. (You can assume the input parameter will be valid.)
- A copy constructor.
- A method `sale`: When called, it represents a sale to a single customer. It accepts an `int` parameter for the number of games sold to that single customer. A customer must go away empty-handed if there is not enough inventory to complete the customer's sale in its entirety. No exception or error, however, is returned by the method in this case. Besides that situation, you can assume the input parameter is valid. The method has no return value.
- A method `getNumCustomers` that returns the total number of fulfilled customers that have bought things in the store. Only customers whose purchases are entirely fulfilled count as a "fulfilled customer."
- A method `getAverageGamesPerSale` that returns the average number of games sold per fulfilled customer (as a `float`). Only customers whose purchases are entirely fulfilled count as a "fulfilled customer."

The following should be legal code with your class:

```
GameStore aStore = new GameStore(10);
aStore.sale(2);
aStore.sale(2);
aStore.sale(1);
aStore.sale(3);
aStore.sale(3);
System.out.println("Total fulfilled customers = " + aStore.getNumCustomers());
System.out.println("Average per sale =    " + aStore.getAverageGamesPerSale());
```

The output of this code should be:

```
Total fulfilled customers = 4
Average per sale = 2.0
```

Your class should exhibit good programming practice (e.g., no privacy leaks, follows coding style guidelines, etc.). You do not need to provide comments nor any additional methods/constructors beyond those specified above. (You will, however, have to create whatever instance and/or static variables are needed to store data, and if you decide to use other methods, you have to define those too.)

3. Build a `Queue` class for a queue of `char`s that is compatible with the driver code below. The `Queue` should operate in a FIFO (first in, first out) fashion and implement the variables and methods also listed below:

   - Data Members: Declare and initialize, as needed, the data item(s) you will need to manage a queue of `char`s. You may **only use arrays and primitives** for your instance and/or static variables.

- Method Members:
  - `public void enqueue(char input)`: This should add the `char input` to your queue
  - `public char dequeue()`: This should remove and return the `char` from the front of your queue. It does not have to test if the queue is empty.
  - `public boolean isEmpty()`: Returns `true` if the queue is empty, `false` otherwise.
- Driver code:

```
public static void main(String[] args) {
    Queue a = new Queue();

    a.enqueue('D');
    a.enqueue('o');
    a.enqueue('g');

    while(!a.isEmpty()) {
        System.out.println("\n\nRemoved: " + a.dequeue());
    }
}
```

In addition to building your class, please describe what is the output when running the main driver code above.

Your class should exhibit good programming practice (e.g., no privacy leaks, follows coding style guidelines, etc.). You do not need to provide comments, a copy constructor (or no-argument constructor), nor any additional methods beyond those specified above. You do need to provide the correct class statement. The class does *not* have to support an infinite queue nor related issues.

4. Consider the following main method using `Pet` class and a `Date` class objects:

```
public static void main(String[] args) {
    Date dogBirthDate = new Date("February", 1, 2010);
    Pet rover = new Pet();
    rover.setBirthDate(dogBirthDate);

    System.out.println("Before: " + rover.getBirthDate().toString());
    dogBirthDate.setMonth("March");
    System.out.println("After: " + rover.getBirthDate().toString());
}
```

and the console output from running this method:

```
Before: February 1, 2010
After: March 1, 2010
```

which reveals that something in the `Pet` and/or `Date` classes are written with a privacy leak.

Your task is to write a new version of whatever method(s) you need to to eliminate the above privacy leak. That is to say, you are to make changes to `Pet` and/or `Date` so that the console output from running the above `main` method is:

```
Before: February 1, 2010
After: February 1, 2010
```

In crafting your solution you may assume the following:

- The birth date in `Pet` is stored in the private instance variable `birthDate` and is a `Date` object.
- The month in `Date` is stored in the private instance variable `month` and is a `String` object.
- `setBirthDate`, `getBirthDate`, and `setMonth` are accessor or mutator methods for instance variables `birthDate` and `month`, as the names suggest.
- Both `Pet` and `Date` have deep-copy copy constructors.
- Both `Pet` and `Date` have `toString` methods.

You just need to provide the code for your corrected method(s) (i.e., the header and body for the method). You do not have to provide anything else in the class definition.