CSE 142 - Computer Programming 1 - Su 02
Homework 2

Assigned: Wednesday, July 3
Due: Wednesday, July 10, BEFORE MIDNIGHT

** General Comments about the Homework **

All homework is turned in electronically. Go the the class web site and use the link on the homework page to do the turnin. Don't be late! Late homeworks will not be accepted.

The homework assignments will all have three parts: Practice Problems, Graded Problems and a Programming Project.

** Homework 2 Practice Problems ** Do not turn in the Practice Problems.

1. Many programs that we write involve sorting lists. We keep lists of objects, and number them starting from 0. For example, a list of 6 elements would be numbered 0, 1, 2, 3, 4, 5. Sometimes we need to divide the list in half and find the "center element".

```
    /** index of the first list element */
    int first=0;

    /** index of the last list element */
    int last=5;

    /** index of the center list element */
    int center = (first+last)/2;
```

What is the value of "center" after the above code is executed?

2. Suppose we wanted to modify the Dog class so that it kept track of how healthy our dog is. To do that, we could say that when a dog is created, it starts with 100 "health points." Every time it goes to sleep without eating something, it loses a number of health points equal to 5% of its weight. If its health points ever go to 0 or less, it has to go to the vet for a checkup.

In order to implement this, you would need to add some variables. For each of the following variables, decide whether you'd make them instance variables or local variables, and explain why.

```
    /** The number of health points the dog currently has */
    double healthPoints;

    /** Indicates whether the dog has eaten since he last slept */
    boolean eatenToday;

    /** Health points the dog loses if he doesn't eat before sleeping */
    double damage;
```

3. Give the values resulting from evaluating the following expressions.

```
4 + 5
4 % 5
(4 + 5) / 2
4 + 5 / 2
(4 + 5) / 2.0
1 / 17
```

4. When you have finished the requirements for the programming problem below, feel free to extend it. One can easily imagine using sounds for the animal voices (see example from Monday July 1 lecture), or drawing an image that shows each pet (see HW1), including weight gain, perhaps. If you implement an "extended pet", use a new class for it so that the basic pet class doesn't get messed up if something goes wrong with your implementation. If you do an extension, you can turn it in. We won't grade it or give extra credit, but we'd like to see (and hear?) what you've done.

** Homework 1 Graded Problems ** Turn in your answers to the graded problems in a text file.

1. Consider the following sequence of code.

```
    /** growth rate */
    double rate = 0.1;

    /** base amount */
    int base = 10;

    /** cumulative amount */
    double toDate;

    /** interval since last update */
    double interval=5;

    toDate = base;
    ...
    toDate = toDate+(interval*rate);    // (a)
    ...
    interval = 25;
    toDate = toDate+(interval*rate);    // (b)
```

What is the value of toDate after statement (a)?

What is the value of toDate after statement (b)?

2. Assume that you have executed these two statements:

```
    int x = 1;
    int y = 2;
```

Say what each of the following expressions evaluates to:

(a) x/y
(b) (double) x / (double) y
(c) 2*x/y
(d) (2*x)/y
(e) 2*(x/y)
(f) 2*(x / (double) y)

3. The remainder operator is often useful when you are looking at the elements in a list.

a. Consider the following few statements.

```
    /** index value */
    int idx = 0;
    ...
    idx = (idx+1) % 3;
    ...
    idx = (idx+1) % 3;
```

What is the value of "idx" after this code executes?

b. Consider the following few statements.

```
/** the index number of the next object to process in a list*/
int next;

/** number of objects in the list */
int bufferSize = 100;
...
next = 98;
...
next = (next+1)/bufferSize;
...
next = (next+1)/bufferSize;
```

What is the value of "next" after this code executes?

4. This question requires you to use your web browser and the documentation for the Java class libraries. I hope you already have a shortcut in your browser that takes you directly to the JavaAPI documentation. If you don't, you should get it set up. The documentation is available online and for download. The first section on the "other links" page describes where to get the documentation. There is a paragraph on the "software" page that describes how to install it. And the tutorial slides (on the "lectures" page) that I gave on Monday July 1, show what the browser looks like when you have set up the shortcuts.

During the lecture on Monday July 1, I showed how to get several different MAX_VALUEs for the various numeric types that we can define. Java also has definitions for a couple of common numeric constants.

Go to the JavaAPI documentation web page. Slide 5 of the tutorial shows what this screen looks like. In the upper left corner of the screen is a set of "packages". Scroll down until you find "java.lang" and click on it. Now scroll the lower left frame down until you see the class name Math. Click on that. You will have a description of the Math class in the main part of the window. Refer to the Field Summary.

What are the two fields defined there and what are their descriptions?

(a)

(b)

5. Consider the following rudimentary Car class.

```
public class Car {
    /**
     * Create a car with a full tank of gas of size capacity
     * with gas mileage mpg.
     * @param mpg Miles per gallon (mileage).
     * @param capacity The capacity of the gas tank.
     */
    public Car(double mpg, double capacity) {
        speed = 0.0;     // begin at zero MPH
        gas = capacity;  // start with full tank
        mileage = mpg;
    }
    /**
     * Calculate how many miles 'til next refueling.
     * @return number of miles to go
    */
    public double getMilesToGo() {
        double mtg = mileage * gas;
        return mtg;
    }
    //---------
    /** speed of car, in miles per hour */
    double speed;
    /** gas in tank, in gallons */
    double gas;
    /** gas mileage, in miles per gallon */
    double mileage;
}
```

Do the following identifiers refer to parameter variables, local variables, or instance variables?

(a) speed

(b) capacity

(c) mtg

(d) mpg

** Homework 2 Programming Project **

Turn in PetSet.java, Whatever.java (or whatever you named your new class), and any required supporting files when you have completed this task.

This project has you develop one new class (named Whatever) to describe a pet other than a cat or a dog, and another new class (named PetSet) to manage a gaggle of pets including a Cat, a Dog, and your new Whatever pet. There is an outline of the PetSet class provided with javadoc comments in place. There are completely implemented Cat and Dog classes.

You need to finish the implementation of PetSet, and define and build a new Whatever.

1. If you haven't done so already, download the cse142-hw2.zip file and unzip it. The project skeleton is in the directory hw142\hw2. Your task is to open the project using BlueJ and implement the two new classes.

2. Start BlueJ and open the hw2 project. Open the PetSet source file and read it. You can see that it will compile, but it doesn't do anything yet. Every method definition is empty.

3. The function of a PetSet is to allow easy control of a bunch of pet objects. Your job is to implement the PetSet constructor and methods described below.

3a. The constructor PetSet(). This is the only required constructor. It should use the "new" operator and the appropriate constructors to create at least one new Cat and at least one new Dog. References to the new objects should be stored in properly defined instance variables that you add at the bottom of the PetSet class definition.

3b. The method snack(). This method calls the eat(food) method of each of the animals that were defined in the constructor. You need to specify the number of pounds of food when you make this call to eat(food). The amount of food provided to each animal is the same as the defined meal size for that animal, whatever that might be. You can get the correct value by using the method getMealSize(), which is defined for each animal.

3c. The method dine(). This method also calls the eat(food) method of each of the animals defined in the constructor. However, the amount of food provided to each animal is two times the defined meal size for that animal. Again, use getMealSize() to get the value, then calculate as needed.

3d. The method sleep(). Calls each of the sleep methods for each of the defined animals.

3e. The method speak(). Calls the appropriate noise-making method for each of the defined animals.

3f. The method main(String[] args). This method is already completely defined. It can be called from BlueJ to run through a complete day's scenario. You should do individual testing of each of the PetSet methods, and then call main() to run through a complete sequence.

4. You should add instance variables to store references to each animal object you create. The creation of the new objects is done in the constructor.

5. After you get the PetSet running okay with a Dog and a Cat, you should create another animal of your own design and write a class for it. The new class can be based closely on Cat.java and Dog.java, and should provide all the same functions, implemented as appropriate for your animal. This can be a Bird, a Snake, an Aibo, or whatever you want. This class does not need to be any fancier than Cat and Dog, just different. For example, it should at least have a different noise-making method, as appropriate to the animal.

6. Once you have your new animal working, go back and modify PetSet to include the new beastie in all the activities defined in part 3 above.