CS 51 Test Program #1

Due: Friday, March 12, 2010, at 6 PM

A test program is a laboratory that you complete on your own, without the help of others. It is a form of take-home exam. You may consult your text, your notes, your lab work, or our on-line examples and web pages, but use of any other source for code is forbidden. You may not discuss these problems with anyone aside from the course instructor(s). You may only ask the TAs for help with hardware problems or difficulties in retrieving your program from a disk or network.

Complete each of the following problems, documenting your code well. In the online version of this document, you will find a link to demonstration programs for all three problems. Starters for all three problems can be found in the TP1 folder in the same spot where you get the starters for your lab assignments.

You are encouraged to reuse the code from your labs or our class examples. Submit your code in the usual way by dragging it into the dropbox folder. Please do *not* submit three separate folders. Instead, place the folders for all three of your complete programs into one folder (which should be named `lastname-TP1`, so if I were submitting this my folder would be named `Kauchak-TP1`), make sure that your name appears not only in the main folder but also in each of the subfolders, and then place the main folder in the CS51 dropoff folder.

Note that if you do everything that's required, the maximum number of points you can get is 96. In order to get the full 100, you must implement some extra features. We've provided some ideas below, but you should feel free to exercise your creativity.

Problem 1: Hot and Cold

When I was a kid, we'd sometimes play "Hot and Cold." You hide something in the house somewhere while the "it" person was out. When they came back in, they would search for the hidden treasure to shouts of "You're getting hotter." when they when they got close, or "You're getting colder." if they wandered in the wrong direction.

For your first test program, we want you to write a simple, computer version of this game. The hidden item will be a point on the program's window picked at random by the computer. The program won't draw anything on the screen at that point. That would make it too easy to find.

The program will draw a small colored circle on the screen to represent the "it" that is searching for the hidden point. The player will be able to drag this circle around the window using the mouse (as in the laundry program, the basketball programs, etc.). The program will vary the color of this circle to give the player clues about where the hidden point is. When the circle is close to the hidden point it should be bright red. When it is far away, it should be blue. In between, the red should fade from red through various shades of purple until it finally becomes blue.

Each time the player releases the mouse, the program should check to see if the distance between the current position of the mouse and the hidden point is less than the radius of the circle. If it is, the program should display a message congratulating the player's success and then immediately pick a new random point so the player can try again. Otherwise, the computer should reassure (or insult) the player and tell the player to try again without selecting a new point.

There are two "hints" you may need to write this program. First, here is a method that we have not yet used in class that you will want to use for this program. It is named `distanceTo` and it is an accessor method associated with `Location`s. If the names `point1` and `point2` are associated with `Location`s, then

```
point1.distanceTo(point2)
```

will compute and return the distance (measured in pixels) between the two points.

You will want to use the distance between your hidden point and the mouse location to determine the color of the ball displayed on the screen. Unfortunately, when you make a new `Color`, the numbers given for the amount of red, green and blue to use must be integers and `distanceTo` returns a `double`. Our second hint is how to turn a `double` into an `int`. You can "cast" a `double` to an `int` by putting "(int)" in front of an expression:

```
(int) point1.distanceTo(point2)
```

will produce an integer approximating the distance between two points.

The online version of this handout has a demo version of HotAndCold.

Extra credit ideas: Update the message as you get hotter and colder with different messages. Time how long it takes to find the location each time.

Problem 2: The Coin Game

For this problem you will design a simple game involving coins. When the program begins, three coins appear with randomly chosen markings of heads or tails. At least one of the three coins *must* be tails. You can click on a coin and it changes from heads to tails and vice versa. In addition to the coins, a message at the bottom of the screen gives you instructions: the goal of the game is to flip all of the

coins to heads. When all of the coins are flipped to heads, the message changes and tells you how long (in seconds) it took you to flip all of the coins to heads. In addition, you should no longer be able to flip the coins once you've won.

You will need to include two classes: the `Coin` class, which draws and handles state changes in the coin and the `CoinGame` class which extends `WindowController`. Think carefully about what functionality a coin has and how the two classes will communicate.

Make sure your Coin class supports the following methods:

```
public void flip()
public boolean isHeads()
public boolean contains(Location point)
```

To time the game, the `System.currentTimeMillis()` method may be useful, which we've used previously for smooth animation.

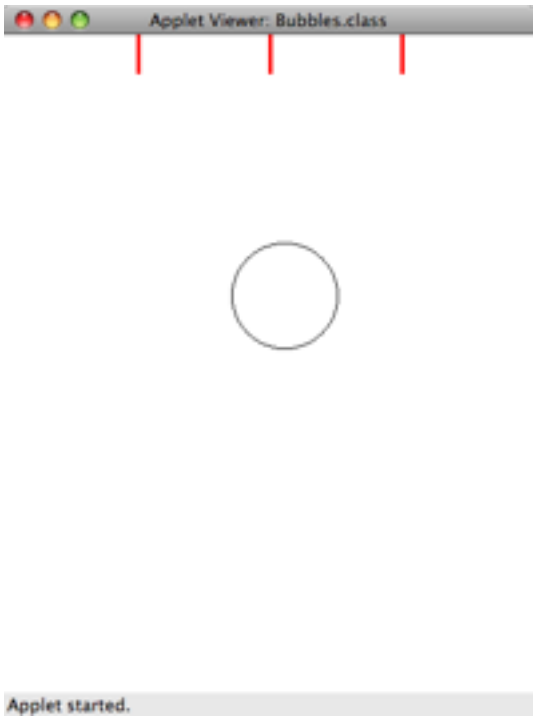
The size of the screen should be 400 by 400 and the start file includes relevant constants.

The online version of this handout has a demo version of CoinGame.

Extra credit ideas: Allow for the ability to reset the game, either with a button or with some other mouse event (like exiting the screen). Improve the graphics of the coins by either using images or more graphics objects.

Problem 3: Dangerous Bubbles

We would like you to implement a program that simulates the blowing of a bubble and its floating off of the screen. Each time the user clicks the mouse, a bubble starts growing. It should remain centered at the same position, increasing in diameter 2 pixels every few milliseconds until it attains a diameter of 80 pixels. At that point it should stop growing and just float directly up the screen. To make this game more interesting, there are three evenly spaced glowing hot needles (thin red rectangles) at the top of the screen. If a bubble hits one of the three needles while growing or floating up then it will pop (disappear). If the bubble misses the needles, then it will just float off the screen



The online version of this handout has a demo version of Bubbles.

To complete the program, you will need to write two classes: a `Bubbles` class that extends `WindowController` and a `Bubble` class that extends `ActiveObject`. The `Bubbles` class is responsible for generating a new bubble each time the mouse is clicked, while the `Bubble` class represents the bubbles that grow and float off the screen (or get popped). The window is 400 pixels wide and 500 pixels high. The needles should be 3 pixels wide and 30 pixels high. The x-coordinates of the left side of the needles should be 100, 200, and 300 pixels, respectively.

Extra credit ideas: bubbles could actually burst by displaying "Pop" momentarily. Keep a running score of how many are made it through vs. how many popped.

Grading Guidelines

Value	Feature
	Style (16 pts for each of 3 programs)
2 pts.	Use of boolean conditions
2 pts.	Ifs/whiles
2 pts.	Appropriate variable (instance/local, public/private)
2 pts.	Descriptive comments
2 pts.	Good names
2 pts.	Good use of constants
2 pts.	Appropriate formatting
2 pts.	Parameters used appropriately
	Correctness (16 pts for each of 3 programs)
Hot and Cold	
2 pts.	Initial screen drawn correctly
3 pts.	Circle is dragged appropriately
4 pts.	Color changes correctly
4 pts.	Display updates correctly when mouse released
3 pts.	Game resets appropriately
Coin Game	
2 pts.	Initial screen drawn correctly
2 pts.	Coins randomly selected appropriately
4 pts.	Coins flip when clicked
3 pts.	Game end detected correctly
3 pts.	Message updated with time
2 pts.	Coins don't flip after gameover
Dangerous Bubbles	
3 pts.	Initial screen drawn correctly
3 pts.	Bubble created at click location
3 pts.	Bubble grows properly
3 pts.	Bubble floats up properly
4 pts.	Bubble pops when it hits a needle (both when growing and floating)
	Miscellaneous (4 pts total)
	Extra Credit (4 pts maximum)