CS 51 Test Program #1

Due: Friday, October 16, at 4 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. You may only ask the TA's for help with hardware problems or difficulties in retrieving your program from a disk or network.

If you get an error message that mentions in internal compiler error, or another error message that makes no sense (e.g., refers to a line of your program that does not exist), you may send a copy of the program (along with a copy of the error statement) to the instructor at kim@cs.pomona.edu. If I find that the error statement is not as good as would normally be expected, I will provide you with a better error statement.

Complete each of the following three problems, documenting your code well. Start-up files can be found in the cs51G labs folder as usual. 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 Dropoff folder. Please do *not* submit three separate folders. Instead, place all three of your complete programs into one folder, make sure that your name appears in the title of the folder, and then drag the folder into our dropoff folder.

To receive full credit, your programs must compile correctly using the `rtobjectdraw` dialect.

The test programs consist of three programs.

- `StickyCarpet`, which will be completed during this week's lab session.
- `CardGame` and `ThrownBall`, which you will work on outside of lab and are due on Friday, 10/16, at 4 p.m.

The latter two are described below, while `StickyCarpet` will be distributed during your lab session.

Problem 1: CardGame

For this problem you will implement a very simple card game.

In this game, a hand of cards consists of three cards. When a player clicks a card, the color and the face value of the card is reset. The color is randomly selected among three colors: yellow, cyan, and red. The face value is randomly selected among 5 integers between 1 and 5.

The value of the hand is calculated as follows:

- If the three cards all have the same color, then the value of the hand is the sum of the face values of the three cards;
- If all three cards are of the same value, then, the value of the hand is the sum of the face values of the three cards;
- Otherwise, the value of the hand is the largest among the face values of the three cards. (Note that Grace has a built-in method `max` that can be applied to as many numeric arguments as you like.)

During the game, the player can choose to click any card to reset it. The player wins the game when the value of the hand is within a value range between `minValue` and `maxValue`. Here, we set `minValue` to 12 and `maxValue` to 14.

After each click, the value of the hand is displayed at the bottom of the screen, as is a count of the number of clicks the player has tried. When the player wins, "YOU WIN!" is displayed on screen and the game ends. When the game ends, the player can no longer reset any card.

You will need to implement the main program and one class:

- The `cardGame` object, which `inherits graphicApplication`, sets up the game and responds to the player's clicks.
- The `card` class, which constructs and resets cards.

The size of the screen should be 400 by 300, and the starter includes relevant constants. Note that you may want and/or need to add additional methods.

If your browser is set up to display Java applets, you will be able to see a working version of a Java version of this program at http://www.cs.pomona.edu/classes/cs051/labs/tp1/cardgame/CardGame.html.

Extra credit ideas: allow the game to restart, set a cap on how many times a player can try and have the player lose the game if they have not won by the time the cap is reached, setup a GUI interface so the player can set the value range of the cards and the min and max values.

Table 1: Grading Guidelines for Card Game

Value	Feature
	Code Quality & Readability (16 pts)
2 pts.	use of boolean conditions
2 pts.	ifs/whiles
2 pts.	appropriate vble (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)
4 pts.	correctly display the cards and the text when game starts
4 pts.	card is reset to random color & value on click
4 pts.	correctly compute the value of hand
2 pts.	correctly track number of clicks
2 pts.	end the game when the player wins

Problem 2: ThrownBall

For this problem you will implement a thrown ball that appears to fall under the influence of gravity and bounces.

- The program begins with an empty window.
- Clicking the mouse within that window causes a ball to be thrown, from the left edge of the window, at the same height as the mouse click.

- The ball moves from left to right with a constant speed.
- But the ball begins to fall, ever faster, under the influence of gravity. (*See how to do this below.*)
- When the ball touches the bottom of the window, it bounces, and gravity slows its climb until it begins falling again. The ball should not travel past the bottom of the window.
- When the ball reaches the right edge of the window, it should be removed from the canvas.

You will need to implement the main program and a class:

- The object `thrownBall` will inherit from graphicApplication and create a new ball whenever the mouse is pressed.
- Objects created from class `ball` will put a ball on the screen and move it along its falling/bouncing path. To implement this, the ball must know (at least) its initial starting position, and the locations of the bottom and left edges of the window.

The supplied starter includes skeletons for each of these, as well as suggested constants and parameters for the ball class. Choose a horizontal velocity so that the ball can cross the entire window in a few seconds.

Getting it to bounce Make sure you get uniform horizontal movement (and disappearance) working before you try to implement gravitational acceleration and bouncing.

Start with a vertical change (`dy`) of 0 the first time you move the ball. However, after each move add a constant to `dy`. I found it looked good with adding 0.2 to `dy` each time through the loop.

When the ball reaches the bottom of the window, you can make it bounce by changing its vertical speed. Multiplying the vertical speed by -1 would simply turn its downwards motion into upwards motion. We want the ball to lose a little energy on each bounce, so we should multiply its vertical speed by -0.9.

Important: If any part of the ball is off the bottom of the screen after a move, you should instead move it up so that it is just touching the bottom of the screen (but with the same x coordinate). If you do not do that then the ball will occasionally get stuck vibrating at the bottom of the screen, changing direction after each move.

Extra credit:

- As with Frogger, you can use `sys.elapsed` to find out how long the pause took, and compute the actual distance moved in each iteration based on this measured time.
- Use the X position of the mouse click to determine the ball's horizontal velocity. If the click is at the left edge of the window, the ball should have no horizontal velocity. If the click is at the right edge of the window, the ball should be moving fast enough to cross the entire window in one second.
- Each time you move the ball, place a small dot with diameter 2, where the center of the ball was, to leave a trace of the balls bouncing path.
- Choose a random color for each new ball, and make its trace dots the same color.

A running version of the applet should be visible below (if your browser handles applets correctly). If your browser is set up to display applets, you will be able to see a working version of this applet at http://www.cs.pomona.edu/classes/cs051/labs/tp1/thrownball/thrownball.html.

Table 2: Grading Guidelines for Thrown Ball

Value	Feature
	Code Quality & Readability (16 pts)
2 pts.	use of boolean conditions
2 pts.	ifs/whiles
2 pts.	appropriate vble (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)
2 pts.	Ball launches horizontally from the place of the click
2 pts.	Ball moves with constant horizontal velocity
2 pts.	Ball falls at an accelerating rate
2 pts.	Ball bounces back up at bottom
2 pts.	Ball loses energy on the bounce
2 pts.	Ball rises at a decelerating rate
1 pts.	Ball disappears off right side of screen
1 pts.	Ball does not pass the bottom of the screen
2 pts.	Ball moves smoothly with no strange behavior