CS 51 Test Program #1

Due: Friday, October 15, 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.

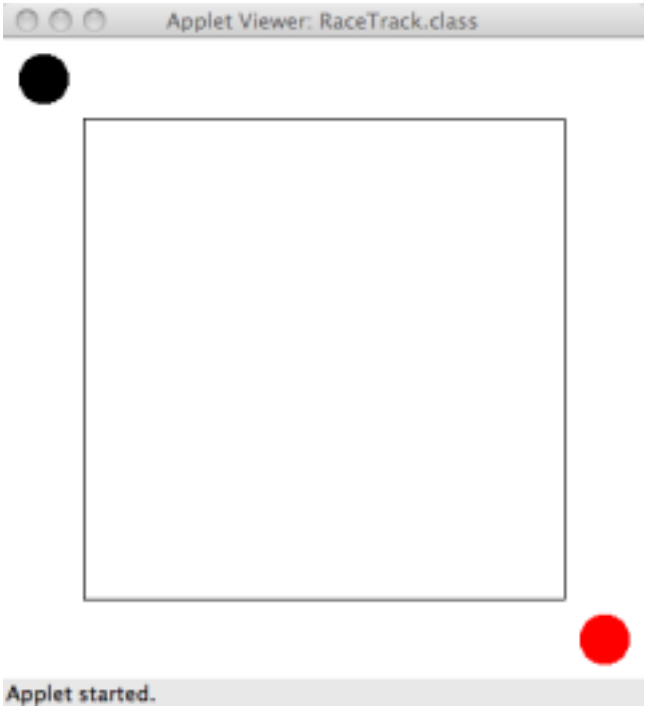
Complete each of the following problems, documenting your code well. In the online version of this document, you will find a link to demonstration programs for all three problems. Starters for all three problems can be found in the TP1 folder in the same spot where you get the starters for your lab assignments.

You are encouraged to reuse the code from your labs or our class examples. Submit your code in the usual way by exporting and 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: Race Track

For this first problem you will draw a square racetrack around which the user can drag a black circle with the goal of getting it to overlap a red target. However, if the black circle ever touches a wall (either the outer or inner) of the racetrack, it will be reset to its original starting location. If the user drags the black circle and releases it when the red and black circles overlap, a message saying "Got it!" should be displayed in the middle of the window, and the black circle should be reset to the upper left hand corner. The "Got it!" message should disappear when the user next presses the mouse on the black circle.



The online version of this handout has a demo version of Race Track.

The window should be 400 pixels wide and 400 pixels tall. The inside of the track should be a rectangle that is 300 by 300, centered in the screen. The filled black circle and the target should have diameter 30.

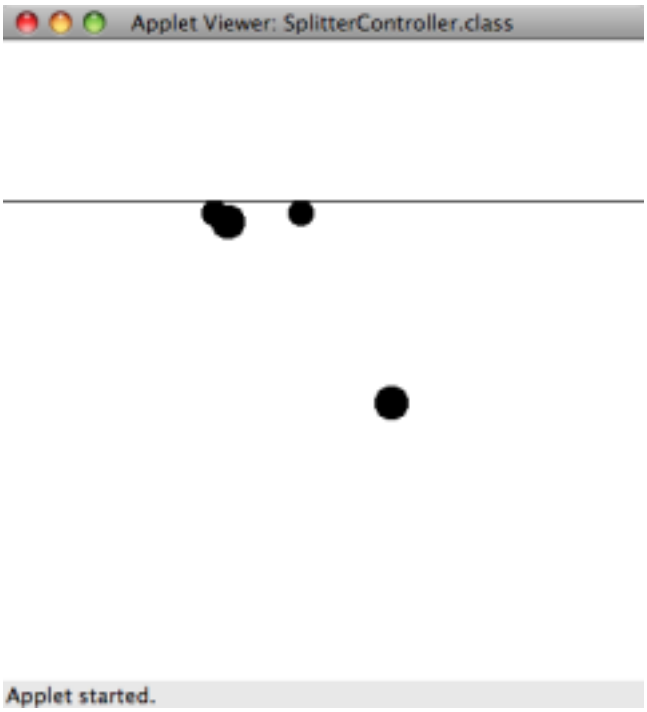
Extra credit ideas: time how long the user takes to make it through the track, setup a more complicated maze for the user to navigate, use better graphics for the track, circle and target.

Problem 2: Splitting Hairs

For this problem you will be implementing a rising, splitting ball. When the program begins, the screen is empty except for a horizontal line. If the user clicks above this line, nothing happens. If the user clicks below this line, a new ball with diameter 20 is created that then proceeds to rise with a random speed towards the vertical line. When it hits the line, it splits into two smaller balls, each of which have 3/4s the diameter of the original ball and those two parts then proceed horizontally in opposite directions at the same speed the original ball had along below the horizontal line. Recall that inside a `run` method you can have more than one while loop and that they can be sequential (i.e. not nested).

You will need to include two classes: the `SplitterController` class, which extends `WindowController` and handles the mouse events and a `SplittingBall` class that extends `ActiveObject` which handles the animation of a splitting ball.

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

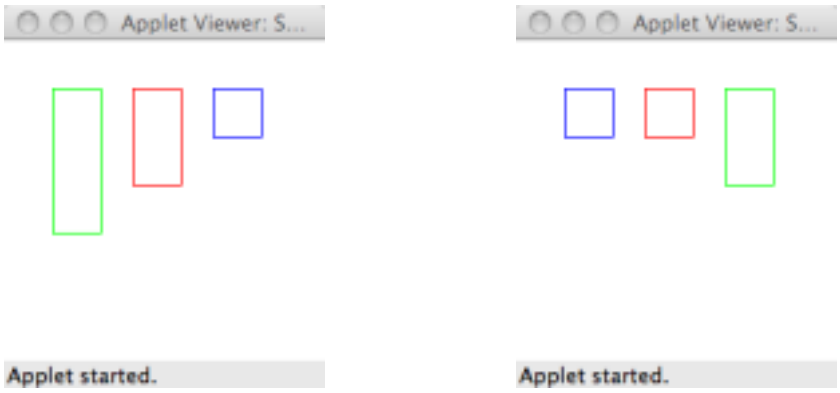


The online version of this handout has a demo version of SplittingBalls.
Extra credit ideas: make the speed depend on how far down the screen the user clicks, have the speed increase as the ball moves up (like reverse gravity), add other balls that split off when it impacts that come off in the diagonal directions, make it an underwater scene with the balls being bubbles.

Problem 3: Set

Set is a simple card game designed in the 1970s that tests players' ability to identify sets of 3 cards that are either the same or all-different along four dimensions: shape, number, color and shading (see *http://www.setgame.com/*). We're going to implement a simplified version of this game.

Our set cards will consist of rectangles with only two features. The rectangles can have differing heights (1, 2 and 3 times the base height) or differing colors (red, green or blue). The program will start with three `SetCard`s being displayed on the screen next to each other in a 200 by 200 window. The user clicks if he/she thinks it is a set. In our case, we'll use a very simplified version of a set where the features need to be differing, that is, all the sizes are different and all of the colors are different. For example, the figure on the left below *is* a set while the figure on the right *is not* since the first two have the same size.



When the user thinks the cards show a set, they can press the mouse anywhere in the window and if the user is correct "Correct" will be displayed at the bottom of the window or "Incorrect" if the cards do not make a set. The user obtains a new set of cards by moving the mouse out of the window (exiting). If/when the user gets a new set of cards, the message is removed.

You should implement two classes, a `SetGame` class that extends `WindowController` and a `SetCard` class representing the cards. Your `SetCard` class should contain a method called `reset` that chooses another random configuration for that card. Notice that this means when the user requests a "new" set of cards, you shouldn't actually create a new set of cards, instead you should pick a new random configuration for each of the cards.

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

Extra credit ideas: allow a set to also include the case when a feature is all equal (for example, all the same color, but all different shapes), add an additional feature such as shape, keep track of how many the user has gotten right and wrong as well as the number of card sets viewed, include buttons (either GUI or non) for "set" or "not set" rather than just clicking to indicate a set, use improved graphics.

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)
Race Track	
2 pts.	Initial screen drawn correctly
3 pts.	Circle is dragged appropriately
4 pts.	Circle correctly detects and resets to wall touches
4 pts.	Winning is detected and message displayed
3 pts.	Game and text resets appropriately
Splitting Hairs	
1 pts	Initial screen drawn correctly
4 pts	balls create where clicked and move up
2 pts	balls can't be created above line
5 pts	balls split appropriately
2 pts	random ball speeds
2 pts	balls are removed appropriately when finished
Set	
5 pts	Initial cards/screen drawn correctly
4 pts	game resets appropriately
4 pts	correct/incorrect are detected appropriately
3 pts	not recreating cards
	Miscellaneous (4 pts total)
	Extra Credit (4 pts maximum)