Nick Troccoli
CS 106A

Solutions to Final Exam

Problem 1: Designing a Spaceship

Part 1

```
public class Spaceship {

	// The number of pounds of food on board
	private int foodOnBoard;

	// A map from crew member names to the pounds of food they eat per day
	private HashMap<String, Integer> crewMemberMap;

	// A list of visited planet names, in order of visit
	private ArrayList<String> planetsVisited;

	// Create a spaceship with an initial amount of food
	public Spaceship(int initialFood) {
		foodOnBoard = initialFood;
		crewMemberMap = new HashMap<>();
		planetsVisited = new ArrayList<>();
	}

	/* Boards a crew member with the given food intake.  This
	 * crew member will now consume food during trips.
	 */
	public void board(String crewMemberName, int foodPerDay) {
		crewMemberMap.put(crewMemberName, foodPerDay);
	}

	/* Unboards a crew member with the given name from the ship.  This
	 * crew member is no longer on the ship and no longer consumes food.
	 */
	public void unboard(String crewMemberName) {
		crewMemberMap.remove(crewMemberName);
	}

	/* Returns a String of visited planets, in order of visit.  The string
	 * should be formatted like "[Earth, Mars, Venus]"
	 */
	public String getPlanetsVisited() {
		return planetsVisited.toString();
	}

	/* Attempts to fly to a planet, which takes the given number of days.
	 * Returns true if we had enough food, and false otherwise.
	 */
	public boolean flyTo(String planetName, int daysRequired) {
		// See if we have enough food for this trip
		int foodRemaining = foodOnBoard;
		for(String person : crewMemberMap.keySet()){
			foodRemaining -= crewMemberMap.get(person)*daysRequired;
		}

		if(foodRemaining < 0) {
			return false;
```

```
            }
            planetsVisited.add(planetName);
            foodOnBoard = foodRemaining;
            return true;
      }
}
```

Part 2

```
private String visitablePlanets(HashMap<String, Integer> crewMemberMap,
            HashMap<String, Integer> planetsToVisit, int startingFood) {

      Spaceship myShip = new Spaceship(startingFood);

      // Board all crew members and specify their daily food consumption
      for(String person: crewMemberMap.keySet()){
            myShip.board(person, crewMemberMap.get(person));
      }

      // Visit each planet until we run out of food
      for(String planet : planetsToVisit.keySet()){
            boolean success = myShip.flyTo(planet, planetsToVisit.get(planet));
            if (!success) {
                  break;
            }
      }

      // Return the list of planets we had enough food to visit
      return myShip.getPlanetsVisited();
}
```

Problem 2: Alien Communications

Part 1

a) `{boop=alien, moo=cow}`
b) `{robot=space, daisy=woof}`
c) `{meow=kitten}`

Part 2

1) **Give an example of where inheritance is useful in programming, and explain why it is useful in this example.**

   Inheritance is useful in cases where you have multiple variables that relate to each other in some way, and therefore share code. For instance, you could have an Employee class that has some defined behavior, and also want to write a Programmer class that does everything an Employee does, plus some additional things. Instead of rewriting all of the code from the Employee class in our new Programmer class, we can instead use inheritance to have Programmer *extend* Employee, which means it automatically inherits all of the Employee's behavior without duplicating any code. Then, we can add additional behavior in the Programmer class that is unique to Programmer.

   Key points:
   - providing an example of where inheritance is useful
   - explaining why inheritance is useful in that example (allows you to extend behavior without duplicating code)

2) **Explain why, when we pass primitives as parameters to a method and change them in that method, the changes do not persist outside of that method, but when we pass objects as parameters and change their properties, the changes <u>do</u> persist outside of that method.**

   This behavior occurs because primitives are passed *by value*, while objects are passed *by reference*. Whenever a variable is passed as a parameter, Java makes a copy of whatever is inside that variable's box, and passes that copy to the method being called. For primitives, their *actual value* is stored inside their variable box (such as 2 for an int or 'a' for a char). This means their actual value gets copied, so if you change the copy the original value is still unchanged. However, for objects, their *address* (location where their info is stored) is stored inside their variable box. This means their *location* gets copied, so when a method goes to change something about it, it goes to the *same* location as the original object. Therefore, its changes will change the original version.

   Key points:
   - Primitive variables store their actual value
   - Object variables store the location where their information lives

Problem 3: Decoding Messages

Part 1 *– there are many possible solutions. Here are 2:*

```
private void correctMessage(int[] message) {
	if (message.length == 0) {
		return;
	}

	// The current repetition length, and its repeated bit
	int currentRepetitionLength = 1;
	int repeatedBit = message[0];

	// Iterate over the rest of the bits and flip as needed
	for (int i = 1; i < message.length; i++) {
		if (message[i] == repeatedBit) {
			currentRepetitionLength++;

			// If we reach 4 in a row, flip the bit and start our counting over.
			if (currentRepetitionLength == 4) {
				message[i] = 1 - message[i];
				currentRepetitionLength = 1;
				repeatedBit = message[i];
			}
		} else {
			currentRepetitionLength = 1;
			repeatedBit = message[i];
		}
	}
}
```

Possible Solution 2

```
private void correctMessage(int[] message) {
	// Check every combination of sequential 4 bits
	for (int i = 0; i <= message.length - 4; i++) {

		// Check if i, i+1, i+2, i+3 are the same
		int count = 0;
		for (int checkIndex=i+1; checkIndex < i+4; checkIndex++) {
			if (message[checkIndex - 1] == message[checkIndex]) {
				count++;
			}
		}

		// If all bits are the same, flip the last one
		if (count == 3) {
```

```
                  message[i+3] = 1 - message[i+3];
            }
      }
}
```

Part 2

```
private String decodeMessage(int[] message, int decodeLength,
                             HashMap<String, Character> decodeMap) {
      // Build up our decoded message by decoding each chunk of bits
      String decodedMessage = "";
      for (int i = 0; i < message.length; i+= decodeLength) {
            // Build up a string containing our current chunk of bits
            String currMessage = "";
            for (int j = i; j < i+decodeLength; j++) {
                  currMessage += message[j];
            }

            // Add the decoded letter to our messagge
            decodedMessage += decodeMap.get(currMessage);
      }
      return decodedMessage;
}
```

Problem 4: Analyzing Alien DNA

```
private String mostFrequentKmer(String dnaString, int k) {
	// Build a map of kmers to frequencies
	HashMap<String, Integer> kmerCounts = new HashMap<>();

	for (int i = 0; i < dnaString.length() - k + 1; i++) {
		String currentKmer = dnaString.substring(i, i + k);

		// Update the count for this in our map
		int currentVal = 0;
		if (kmerCounts.containsKey(currentKmer)) {
			currentVal = kmerCounts.get(currentKmer);
		}
		kmerCounts.put(currentKmer, currentVal + 1);
	}

	// Find the most frequent kmer
	int maxCount = 0;
	String mostFrequentKmer = "";
	for (String kmer : kmerCounts.keySet()) {
		if (kmerCounts.get(kmer) > maxCount) {
			maxCount = kmerCounts.get(kmer);
			mostFrequentKmer = kmer;
		}
	}

	return mostFrequentKmer;
}
```

Problem 5: Space Rocks!

```
public class SpaceRocks extends GraphicsProgram {
	// START PROVIDED CODE **********************
	private static final int NUM_ROCKS = 7;
	private static final int ROCK_DIAMETER = 80;
	private static final int ROCK_SPACING = 20;
	// END PROVIDED CODE ************************

	private GOval lastSelection;
	private ArrayList<GOval> rocks;

	public void run() {
		addRocks();
		repositionRocks();
	}

	// This method adds NUM_ROCKS rocks to our ArrayList, and the screen at (0, 0)
	private void addRocks() {
		rocks = new ArrayList<>();
		for (int i = 0; i < NUM_ROCKS; i++) {
			GOval rock = new GOval(ROCK_DIAMETER, ROCK_DIAMETER);
			rock.setFilled(true);
			rock.setColor(RandomGenerator.getInstance().nextColor());
			add(rock);
			rocks.add(rock);
		}
	}

	// This method updates the locations of all rocks in our arraylist.
	private void repositionRocks() {
		double x = 0;
		for (GOval rock : rocks) {
			rock.setLocation(x, 0);
			x += ROCK_DIAMETER + ROCK_SPACING;
		}
	}

	public void mouseClicked(MouseEvent e) {
		GOval currentSelection = getElementAt(e.getX(), e.getY());

		// If this is the first of two clicks, record the selection
		if (lastSelection == null) {
			lastSelection = currentSelection;
		} else {
			// If this click is the same as the last, remove the rock
			if (lastSelection == currentSelection) {
				rocks.remove(currentSelection);
				remove(currentSelection);
				lastSelection = null;
			} else {
				// This click is different from the last, so swap rocks
```

```
                        int lastSelectionIndex = rocks.indexOf(lastSelection);
                        int currentSelectionIndex = rocks.indexOf(currentSelection);
                        rocks.set(lastSelectionIndex, currentSelection);
                        rocks.set(currentSelectionIndex, lastSelection);
                        lastSelection = null;
                    }

                    // Since we have changed rock positions, recalculate positions
                    repositionRocks();
                }
            }
}
```

Problem 6: Images of Outer Space

Part 1

```
private GImage detectEdges(GImage image){
  int[][] pixels = image.getPixelArray();
  int[][] newPixels = new int[pixels.length][pixels[0].length];

  for (int r = 0; r < newPixels.length; r++) {
    for (int c = 0; c < newPixels[0].length; c++) {
      int ownLuminosity = computeLuminosity(GImage.getRed(pixels[r][c]),
          GImage.getGreen(pixels[r][c]),
          GImage.getBlue(pixels[r][c]));

      // Compute the average luminosity of our neighbors
      int totalLuminosity = 0;
      int numNeighbors = 0;
      for (int neighborRow = r-1; neighborRow <= r+1; neighborRow++) {
        for (int neighborCol = c-1; neighborCol <= c+1; neighborCol++) {

          // If this neighbor index is in bounds...
          if (neighborRow >= 0 && neighborRow < pixels.length
              && neighborCol >= 0 && neighborCol < pixels[0].length) {

            // If this neighbor index is not the original pixel...
            if(neighborRow != r || neighborCol != c) {
              // Add its luminosity to our calculations
              totalLuminosity += computeLuminosity(
                  GImage.getRed(pixels[neighborRow][neighborCol]),
                  GImage.getGreen(pixels[neighborRow][neighborCol]),
                  GImage.getBlue(pixels[neighborRow][neighborCol])
                  );
              numNeighbors++;
            }
          }
        }
      }

      int neighborAvgLuminosity = totalLuminosity/numNeighbors;

      // If our difference is > THRESHOLD, (r,c) is white.  Else, it is black.
      int luminosityDifference = Math.abs(ownLuminosity - neighborAvgLuminosity);
      if (luminosityDifference > THRESHOLD){
        newPixels[r][c] = GImage.createRGBPixel(255, 255, 255);
      } else {
        newPixels[r][c] = GImage.createRGBPixel(0, 0, 0);
      }
    }
  }

  return new GImage(newPixels);
}
```

Part 2

```
public class EdgeDetector extends GraphicsProgram {
	// START PROVIDED CODE ********************
	private int computeLuminosity(int r, int g, int b) {
		return GMath.round((0.299 * r) + (0.587 * g) + (0.114 * b));
	}

	private static final int THRESHOLD = 6;
	// END PROVIDED CODE **********************

	JTextField textField;

	// The loaded-in image
	GImage image;

	// Add interactors
	public void init() {
		add(new JLabel("Filename: "), SOUTH);
		textField = new JTextField(16);
		textField.addActionListener(this);
		textField.setActionCommand("Load Image");
		add(textField, SOUTH);
		add(new JButton("Load Image"), SOUTH);
		add(new JButton("Detect Edges"), SOUTH);
		addActionListeners();
	}

	public void actionPerformed(ActionEvent event) {
		// Load a new image and put it at (0, 0)
		if (event.getActionCommand().equals("Load Image")) {
			image = new GImage(textField.getText());
			add(image, 0, 0);
		} else if (event.getActionCommand().equals("Detect Edges")) {
			// Run edge detection on the already-loaded-in image.
			GImage newImage = detectEdges(image);
			add(newImage, image.getWidth(), 0);
		}
	}
}
```