CIS 1100 Summer 2022 exam 1 solutions

2.1

While you can conceptually imagine arbitrarily large numbers, in reality you need space to represent them. Since computers have finite space, they can only represent finitely sized numbers.

2.2

`left` still represents the first index of the portion of the array we are currently searching. `right` represents the index 1 past the end of the portion of the array we are currently searching.

2.3

A for loop makes the initialization step, the loop condition, and the update step explicit. Using a while loop, you can just do the initialization before the loop, use the same loop condition in the while loop, and perform the update step inside the loop body. This results in the same loop as the original for loop.

3

A few things can vary, such as where you declare mx and my and the order of statements in the inner for loop.

Version 1:

```
	public static void main(String[] args) {
		PennDraw.enableAnimation(30);
		PennDraw.setPenColor(PennDraw.BLUE);
		double radius = 0.5;
		double px = Math.random();
		double py = Math.random();
		while (true) {
			PennDraw.clear();
			PennDraw.filledCircle(px, py, radius);
```

```
                if (PennDraw.mousePressed()) {
                      double mx = PennDraw.mouseX();
                      double my = PennDraw.mouseY();
                            if (mx > px - radius && mx < px + radius &&
                                  my > py - radius && my < py + radius) {
                            radius *= 0.75;
                            px = Math.random();
                            py = Math.random();
                      }
                }
                PennDraw.advance();
          }
    }
```

Version 2:
Since nothing changes at each iteration unless you click the circle, it works fine to clear only when that occurs.

```
    public static void main(String[] args) {
          PennDraw.enableAnimation(30);
          PennDraw.setPenColor(PennDraw.BLUE);
          double radius = 0.5;
          double px = Math.random();
          double py = Math.random();
          while (true) {
                PennDraw.filledCircle(px, py, radius);
                if (PennDraw.mousePressed()) {
                      double mx = PennDraw.mouseX();
                      double my = PennDraw.mouseY();
                            if (mx > px - radius && mx < px + radius &&
                                  my > py - radius && my < py + radius) {
                            PennDraw.clear();
                            radius *= 0.75;
                            px = Math.random();
                            py = Math.random();
                      }
                }
                PennDraw.advance();
          }
    }
```

4

The corrected version of the code is below:

```
    public static void fixPhoneNumbers(char[] nums) {
          fixPhoneNumbersHelper(nums, 0);
```

```
3:     }
4:     public static void fixPhoneNumbersHelper(char[] nums, int index) {
5:            if (index % 10 == 0) {
6:                   fixPhoneNumbersHelper(nums, index + 1);
7:            } else if (index >= nums.length) {
8:                   return;
9:            } else {
10:                  nums[index] = (char)(nums[index] - ('a' - '0'));
11:                  fixPhoneNumbersHelper(nums, index + 1);
12:           }
13:    }
```

The bugs were:
Line 2: no function of that name (this was a typo on our part)
Line 2: this should call the helper function with index 0. However, it doesn't matter if it's called with index 0 or 1, since the helper function skips index 0 anyways. This was intended to be one of the bugs but ended up not causing any issues.
Line 5: nums can include many phone numbers, and each first digit is already fixed, so we should skip every 10 elements.
Line 6: no function of that name (again, a typo on our part)
Line 7: should be >= instead of >
Line 11: this recursive call was missing. Without this, the helper function only fixes one digit of the phone number.

5

```
String total = "";
String piece = "";
for (int i = 0; i < input.length(); i++) {
       if (input.charAt(i) == ' ') {
              total += seqToChar(piece);
              piece = "";
       } else {
              piece += input.charAt(i);
       }
}
return total;
```

Many solutions are possible. Another intuitive one is to store an index of where the beginning of the current "letter" starts, and to use that to process the current "letter" once a space is seen.

```
String total = "";
int begin = 0;
for (int i = 0; i < input.length(); i++) {
       if (input.charAt(i) == ' ') {
              total += seqToChar(input.substring(begin, i));
```

```
            begin = i + 1;
        }
}
return total;
```

6.1

For all of the problems in question 6, there are many possible solutions.

```
public static boolean isValidArray(int[] input) {
        if (input.length % 2 != 0) {
                return false;
        }

        for (int i = 0; i < input.length; i++) {
                if (input[i] < 1 || input[i] > 13) {
                        return false;
                }
        }
        return true;
}
```

6.2

```
@Test
public void resultOfWarTest() {
        int[] twoRoundGame = {10, 4, 9, 12};
        String expected = "Paul";
        String actual = resultOfWar(twoRoundGame);
        assertEquals(expected, actual);
}
```

6.3

```
public static String resultOfWar(int[] input) {
        if (!isValidArray(input)) {
                return "Error: invalid input";
        }
        int paulScore = 0;
        int eliScore = 0;

        for (int i = 0; i < input.length; i += 2) {
                if (input[i] > input[i+1]) {
                        eliScore++;
                } else if (input[i] < input[i+1]) {
                    paulScore++;
            }
      }

      if (paulScore >= eliScore) {
            return "Paul";
      } else {
            return "Eli";
      }
}
```

6.4

```
public static int[] addNewRounds(int[] input) {
      int[] arr = new int[input.length * 2];

      for (int i = 0; i < input.length; i++) {
            arr[i] = input[i];
      }

      for (int i = input.length; i < arr.length; i++) {
            arr[i] = (int)(Math.random() * 13) + 1;
      }
      return arr;
}
```