CIS 110 — Introduction to Computer Programming

28 June 2012 — Final Exam

Name: ______________________________

Recitation # (e.g. 201): ______________________________

Pennkey (e.g. bjbrown): ______________________________

My signature below certifies that I have complied with the University of Pennsylvania's Code of Academic Integrity in completing this examination.

________________________ ________________

Signature Date

Scores:

1		1
2		5
3		7
4		15
5		20
6		32
Total:		80

CIS 110 Final Instructions

- You have 110 minutes to finish this exam. Time will begin when called by a proctor and end precisely 110 minutes after that time. If you continue writing after the time is called, you will receive a zero for the exam.
- This exam is *closed-book, closed-notes, and closed-computational devices.* Except where noted, you can assume that code included in the question is correct and use it as a reference for Java syntax.
- This exam is long. If you get stuck part way through a problem, it may be to your advantage to go on to another problem and come back later if you have time.
- When writing code, the only abbreviations you may use are for `System.out.println`, `System.out.print`, and `System.out.printf` as follows:

  System.out.println $\longrightarrow$ S.O.PLN
  System.out.print $\longrightarrow$ S.O.P
  System.out.printf $\longrightarrow$ S.O.PF

  Otherwise all code must be written out as normal, including all curly braces and semicolons.
- Please do not separate the pages of the exam. If a page becomes loose, write your name on it and use the provided staplers to reattach the sheet when you turn in your exam so that we don't lose it.
- If you require extra paper, please use the backs of the exam pages or the extra sheet(s) of paper provided at the end of the exam. Clearly indicate on the question page where the graders can find the remainder of your work (e.g. "back of page" or "on extra sheet"). Staple an extra sheets you use to the back of your exam when you turn it in using the provided staplers.
- If you have any questions, please raise your hand and an exam proctor will come to answer them.
- When you turn in your exam, you may be required to show ID. If you forgot to bring your ID, please talk to an exam proctor immediately.

Good luck, have fun!

Miscellaneous

1. (1 points)

   (a) Write your name, recitation number, and PennKey (username) on the front of the exam.

   (b) Sign the certification that you comply with the Penn Academic Integrity Code

Short Answer

2. (5 points) Answer each of the following questions in **at most** two sentences.

   (a) If I pushed 3, then 5, then 7 onto a stack of integers, what would `pop()` remove?

   (b) What are the best and worst case running times for insertion sort, and in which cases do they occur? If you expect the input to be random, what do you expect the running time to be? You may use any notation you like as long as it is clear that you understand the answer.

   (c) What is one reason we would prefer a linked list to an array? What is one reason we would prefer an array to a linked list?

   (d) Give two reasons why we might want to use an interface.

   (e) What is the purpose of the `Comparable` interface?

Debugging

3. (7 points) Find and correct 7 errors in the following code. Note: some of the bugs may cause incorrect behavior rather than syntax errors. The line numers are included for your convenience.

```
 1: public class Ouch {
 2:   private int[] arr;
 3:   private string name;

 4:   public Ouch() {
 5:     arr = int[5];
 6:   }

 7:   public DebugMe(int[] arr, String n) {
 8:     arr = arr;
 9:   }

10:   public printMe() {
11:     for (i = 0; i < arr.length; i++) {
12:       System.out.println(name.length() + arr[i]);
13:     }
14:   }
15: }
```

BUG 1: ____________________

BUG 2: ____________________

BUG 3: ____________________

BUG 4: ____________________

BUG 5: ____________________

BUG 6: ____________________

BUG 7: ____________________

Awesome TAs, What Are They Good For?

4. (15 points) The following program prints five lines of output. What are they? Write and circle your answer on the following page.

```
public class Awesome {
  public String thing;
  public double[] lulz;
  public int hi;

  public Awesome(String a, int b)   {
    hi = b;
    a = b + a + hi;
    thing = a;
    b = b * b;
    lulz = new double[b];
    for(int Jeff = 0; Jeff < hi; Jeff++)
      lulz[Jeff] = Jeff + b;
  }

  public static void roffleCopter(String thing, Awesome sauce) {
    Awesome bye = sauce;
    thing = sauce.thing;
    if(sauce.lulz.length >= 2)
      bye.lulz[1] = sauce.hi;
    String t = sauce.thing;
    t = t + t;
    bye.hi = thing.length();
    if(sauce.hi == bye.hi)
      bye.hi++;
  }

  public static void main(String[] args) {
    String a = "cheezburger";
    String b = "fryz";
    Awesome Sam = new Awesome(a, 4);
    Awesome Kat = new Awesome(b, 5);
    System.out.println(1 + ": " + Sam.thing + " " + Sam.hi);
    System.out.println(2 + ": " + Kat.thing + " " + Kat.hi);
    Awesome.roffleCopter("WOMP", Sam);
    Awesome.roffleCopter("WUB", Kat);
    System.out.println(3 + ": " + Sam.thing + " " + Sam.hi);
    System.out.println(4 + ": " + Kat.thing + " " + Kat.hi);
    System.out.println(5 + ": " + a + " " + b);
  }
}
```

Write the five lines that the `Awesome` program prints out here. Circle your answer.

QuickSelect

5. (20 points) Quickselect is an algorithm for finding the k-th element of an array, also referred to as the element of rank k. To understand how quickselect works, assume the value of the first array element is *val*. Quickselect will rearrange the elements of the array so that every element that is less than *val* comes before it in the array (in no particular order), and every element that is larger than it comes after (again, in no particular order). This can be done in one pass over the array that works in from both ends, leaving elements in place when they are at the correct end of the array, and swapping them when they aren't. Finally, the first element (*val*) is swapped into its position in the middle of the array. When this is done, we know exactly how many elements are smaller than *val*, and how many are larger. That means we know whether the k-th element is before or after *val* in the array, and we can recursively call quickselect on that portion of the array. It turns out that quickselect almost always finds the k-th lowest element in $O(n)$ time for an array of n elements, and it is widely used in practice.

Below is a program that takes a list of integers as command line arguments, finds a specific element within the array, and prints it. It also prints the array contents after the selection, and some debugging information during the selection:

```
public class QuickSelect {
  public static void main(String[] args) {
    // take a list of integers as command-line arguments
    // and convert them to an int[] array
    int[] arr = new int[args.length];
    for (int i = 0; i < args.length; i++)
      arr[i] = Integer.parseInt(args[i]);

    // use quick-select to find the (args.length / 2) lowest value
    int retval = select(arr, 0, args.length, args.length / 2);

    // print out the return value and contents of arr
    System.out.println("return value = " + retval);
    for (int i = 0; i < arr.length; i++) System.out.print(arr[i] + " ");
    System.out.println();
  }

  // swap elements a and b of arr
  private static void swap(int[] arr, int a, int b) {
    int tmp = arr[a];
    arr[a] = arr[b];
    arr[b] = tmp;
  }

  /* CONTINUED ON NEXT PAGE */
  /* CONTINUATION OF QuickSelect FROM PREVIOUS PAGE */

  // sort arr just enough to identify and return
  // the rank-lowest value of arr
  //
  // e.g. if rank is 2, sort arr just enough to find an return
  // the second lowest element
  public static int select(int[] arr, int lo, int hi, int rank) {
    // print out lo and hi
    System.out.println(lo + " " +  hi);

    // rearrange elements lo..hi of arr so that
    // all values that are <= arr[lo] are before it
    // in arr, and all elements that are > arr[lo]
    // are after it
    int midval = arr[lo];
    int l = lo + 1, r = hi - 1;
    while (r > l) {
      if (arr[l] > midval) {
        swap(arr, l, r);
        r--;
      } else l++;
    }

    if (arr[l] > midval) l--;
    swap(arr, lo, l);

    // Now the value that used to be in arr[lo] is in its
    // final (sorted) position in the array, even if nothing
    // else is fully sorted.  So if it's at position rank, it's
    // the value we're looking for.  Otherwise we can tell if the
    // value we're looking for is below or above it, and recurse
    // only on that half of the array.
    if (l == rank) return arr[l];
    else if (l < rank) return select(arr, r, hi, rank);
    else return select(arr, lo, r, rank);
  }
}
```

For each invocation of `QuickSelect` below, write exactly what the program will print:

(a) `% java QuickSelect 4 3 2 1 7 6 5`

(b) `% java QuickSelect 3 2 1 6 4 5 7`

Queues on First

6. (32 points) You want to have a conversation with your friend about baseball. Unfortunately, he's a computer science major, and doesn't know a thing about it. But he does know Java, so you can decide to explain the batter's lineup to him in code. The general idea is that there is a line of `Player`s, and that when someone must hit the ball, the person that goes up is the one who has waited the longest in line. When the coach adds someone to the lineup they get in the back of the line and must wait for everyone in front to bat first.

Given the `Player` and `Lineup` classes below, answer the questions on the following pages.

```
public class Player {
  public String name;
  public int rbi;
  public Player next; // the next Player in the lineup
                      // null if there are no players behind this one
}

public class Lineup {
  public Player first; // the first player in the batting lineup
  public Player last;  // the last player in the lineup
  public int    size;  // the number of players in the lineup

  // remove and return the Player at the front of the lineup
  // return null if the lineup is empty
  public Player atBat() { /* ... */ }

  // add a Player to the lineup
  // assumes that p is not already in the lineup
  // does nothing if p == null
  public void addToLineup(Player p) { /* ... */ }
}
```

(a) Implement the `atBat()` and `addToLineup` methods. Make sure that the methods obey the comments in the code extract above, and that `first`, `last`, and `size` are updated correctly.

```
public Player atBat {





}

public void addToLineup(Player p) {





}
```

(b) The team you're talking about is exceptionally rowdy, so they often have players ejected from the game. To explain how this works, you decide to add a third public method, `eject` to the `Lineup` class that take a `Player p`, and removes `p` from the lineup and returns `true` if `p` was in it. If `p` is `null` or not in the lineup, `eject` makes no change and returns `false`. For example, if the lineup was Catherine, Sam, Jeff, and Kat when the umpire ejected Sam, the lineup would become Catherine, Jeff, and Kat, and `eject` would return true. Don't forget to handle `null`s appropriately and to update the `first`, `last`, and `size` variables.

Postscript (extra paper)

Postscript (extra paper)