Name: _____

USC loginid (e.g., ttrojan): _____

CS 455 Midterm Exam 1
Spring 2012 [Bono]

Feb. 16, 2012

There are 4 problems on the exam, with 45 points total available. There are 7 pages to the exam, including this one; make sure you have all of them. There is also a one-page code handout that accompanies the exam. If you need additional space to write any answers, you may use the backs of exam pages (just direct us to look there).

Note: if you give multiple answers for a problem, we will only grade the first one. Avoid this issue by labeling and circling your final answers and crossing out any other answers you changed your mind about (though it's fine if you show your work).

Put your name and USC loginid at the top of the exam. Please read over the whole test before beginning. Good luck!

	value	score
Problem 1	10 pts.	
Problem 2	10 pts.	
Problem 3	10 pts.	
Problem 4	15 pts.	
TOTAL	45 pts.	

Problem 1 [10 pts.]

Consider the following program:
(Note: more about Point class on code handout.)

```
public class Prob1 {
  public static void foo(Point a) {
    a = new Point(a.getX() + 5, a.getY() + 10);
  }
  public static void main(String[] args) {
    Point p = new Point(5, 10);
    Point r = p;
    p.translate(1,1);
    foo(r);
    System.out.println(p + " " + r);
  }
}
```

Part A [6]. In the space below, draw a box-and-pointer diagram (a.k.a., memory diagram) showing all object variables, objects, and their state as they have changed during the code sequence. This includes showing `foo`'s parameter `a`.

Part B [2]. What is printed by the code? For the purpose of this problem assume a `Point` is printed as follows: `[x, y]`

Part C [2]. How many `Point` objects are created by the code above?

Problem 2 [10 pts.]

Implement the class `DigitExtractor`, which breaks up an integer into its individual digits. After being initialized with a positive integer, each call to `nextDigit()` returns the next digit in the integer, starting from the rightmost digit; i.e., a sequence of such calls returns the digits in *reverse* order.

Here is an example of code to use the class. It extracts all of the digits of 27,054:

```
DigitExtractor extractor = new DigitExtractor(27054);

while (extractor.hasNextDigit()) {

  System.out.println(extractor.nextDigit());

}
```

Corresponding output:

```
4
5
0
7
2
```

Hint: use the modulus operator (`%`) and integer division.

The modulus operator gives the remainder of dividing two integers. For example, the result of

```
17 % 3
```

is `2` (17 divided by 3 has a remainder of 2).

Reminder: Integer division is done with the / operator performed on two integers. For example, the result of

```
17 / 3
```

is `5`.

[Do not write your answer here. The class interface and space for your answer is provided on the next page.]

Problem 2 (cont.)

Complete the implementation of `DigitExtractor` – details of this problem are on the previous page.

```
/**
 *  DigitExtractor breaks up a positive integer into its individual digits
 *  in reverse order.
 */
public class DigitExtractor{

   /**
    * Creates digit extractor for the given integer.
    * @param anInt the integer to extract from
    *  PRE: anInt > 0
   */
   public DigitExtractor(int anInt) {

   }

   /**
    * Returns true iff there are more digits left to extract.
   */
   public boolean hasNextDigit() {

   }

   /**
    * Extracts the the "next" digit in the integer (starts from rightmost
    * digit, and goes leftward)
    * PRE: hasNextDigit()
    * @return the digit
   */
    public int nextDigit() {

    }
}
```

Problem 3 [10 pts.]

Implement the static method `printAddProblem`, which takes as its parameter an array of integers and prints out the values as an addition problem. When given an array of length 0 it should print nothing. See examples below for the exact format to use.

Ex1: Output of a call to `printAddProblem` on the array `[3, 7, 2]`

```
3 + 7 + 2
```

Ex2: Output of a call to `printAddProblem` on the array `[3]`

```
3
```

Ex3: Output of a call to `printAddProblem` on the array `[3, -7, 0, 2]`

```
3 + -7 + 0 + 2
```

```
/**
   Prints out the given sequence of numbers as an addition problem.
   @param nums the numbers to make the addition problem from
 */
public static void printAddProblem(int[] nums)
```

Problem 4 [15 pts. total]

Implement the static method `pack`, which takes out all of the zeroes from an array, packing the resulting values to the left . This method does not do the packing in-place, but puts the packed version in another array, `packed`, leaving the original array unchanged. While the array we start out with has zero and/or non-zero values throughout, the resulting array is a partially filled array, which can be described by the array itself (2nd parameter) plus an integer size (return value), which describes which part of the array is being used. Note: this problem doesn't involve any array resizing.

In the following examples, assume `arr.length` and `packed.length` are 5 (although in general they can have any length, as long as `packed` starts out as big as `arr`). The `?` means that the value is unknown (i.e., not significant).

	arr:	after call to **pack(arr, packed):** **packed:**	return value:
Ex1:	5 0 0 2 4	5 2 4 ? ?	3
Ex2:	0 0 0 0 0	? ? ? ? ?	0
Ex3:	5 0 0 3 0	5 3 ? ? ?	2
Ex4:	5 0 0 0 0	5 ? ? ? ?	1
Ex5:	0 0 0 0 5	5 ? ? ? ?	1

One more example example, but where `arr.length` and `packed.length` are `10`:

	arr:	after call to **pack(arr, packed):** **packed:**	return value:
Ex6:	0 2 3 4 0 6 5 0 0 1	2 3 4 6 5 1 ? ? ? ?	6

For full credit, your answer must be efficient: for example, it shouldn't take many steps to do something that can be done in one step.

Method interface:

```
/**
 * Removes all 0 values from the array, packing the resulting values to the
 * left -- the changed version is in "packed" (original array unchanged).
 * @param arr The array to pack (unchanged by method)
 * @param packed The resulting packed array.
 *  PRE: packed.length >= arr.length
 * @return the "size" of the resulting partially filled array -- the valid
 *  values are now in packed[0] through packed[size-1]
 */
public static int pack(int[] arr, int[] packed) {
```

[This header and space for your answer is provided on the next page.]

Problem 5 (cont.)

```
/**
 * Removes all 0 values from the array, packing the resulting values to the
 * left -- the changed version is in "packed" (original array unchanged).
 * @param arr The array to pack (unchanged by method)
 * @param packed The resulting packed array.
 *  PRE: packed.length >= arr.length
 * @return the "size" of the resulting partially filled array -- the valid
 *  values are now in packed[0] through packed[size-1]
 */
public static int pack(int[] arr, int[] packed) {
```