Name: _____

USC netID (e.g., `ttrojan`): _____

CS 455 Midterm Exam 1
Fall 2015 [Bono]

Thursday, Oct. 1, 2015

There are 5 problems on the exam, with 58 points total available. There are 10 pages to the exam (5 pages double-sided), including this one; make sure you have all of them. If you need additional space to write any answers, pages 9 and 10 are left blank for that purpose. If you use these pages for answers you just need to 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 username (a.k.a., netID) at the top of the exam. Also put your netID at the top right of the front side of each page of the exam. Please read over the whole test before beginning. Good luck!

	value	score
Problem 1	6 pts.	
Problem 2	4 pts.	
Problem 3A	10 pts.	
Problem 3BCD	10 pts.	
Problem 4	8 pts.	
Problem 5	20 pts.	
TOTAL	58 pts.	

`Arrays.toString` method documentation (from Oracle documentation):

`public static String toString(int[] a)`

Returns a string representation of the contents of the specified array. The string representation consists of a list of the array's elements, enclosed in square brackets (`"[]"`). Adjacent elements are separated by the characters `", "` (a comma followed by a space). Elements are converted to strings as by `String.valueOf(int)`. Returns `"null"` if `a` is `null`.

Problem 1 [6 pts.]

Consider the following program:

```
public class Prob1 {
  public static void main(String[] args) {
    int[] myNums = null;
    initArray(myNums, 5);
    System.out.println(Arrays.toString(myNums));
  }

  public static void initArray(int[] nums, int size) {
    nums = new int[size];
    for (int i = 0; i < nums.length; i++) {
      nums[i] = i;
    }
  }
}
```

Part A [4]. 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 `initArray`'s parameters.

Part B [2]. What is printed by the code? (Documentation for `Arrays.toString` appears on the exam cover page)

Problem 2 [4 pts.]

Consider the following static method that returns a `boolean` telling us whether the given day of the week is a work day. Days are represented using the constants below. Hint: it doesn't do what you might think it does.

```
public static final int SUN = 0;       // assume these constants are also
public static final int MON = 1;       // defined in the same class
. . .
public static final int SAT = 6;  // Do not change these constants in part B


/**
   Returns true iff the given day is a work day.
   PRE: day is one of SUN, MON, TUE, WED, THU, FRI, SAT
*/
public static boolean isWorkDay(int day) {


   if ((day != SAT) || (day != SUN)) {


      return true;


   }


   return false;
}
```

Part A. Suppose we use this method (with no changes made to it) to find out which days of the week someone has to work. **How many days a week does the worker have to work?**

Part B. Modify the method above so it gives results consistent with the normal American Monday through Friday work week, or if it already does that, write "No change", below. Make any changes directly in the code above, using arrows to show where your code should be inserted, crossing out code that you would get rid of, etc. (Answers with numbers in them will not receive full credit.)

Problem 3 [20 pts. total]

Part A [10]. Complete the implementation of the class `VendingMachine`, defined on this and the next page.

A `VendingMachine` models the inventory inside a vending machine that can hold three types of candy. One can buy candy, find out how much candy there is, and supply the machine with more candy. (Note: The money part of the transactions are not part of this class.) Here is an example of the use of this class:

```
public static void main(String[] args) {
   VendingMachine vend = new VendingMachine();  // machine is empty
   System.out.println(vend.buy(VendingMachine.HERSHEYS, 3);
                              // no product in the machine, machine gives 0
   vend.supply(VendingMachine.HERSHEYS, 20);    // stock the machine
   vend.supply(VendingMachine.M_AND_MS, 20);
   vend.supply(VendingMachine.STARBURST, 20);
   System.out.println(vend.buy(VendingMachine.HERSHEYS, 3);
                                 // customer buys 3 Hersheys bars (prints 3)
   System.out.println(vend.buy(VendingMachine.HERSHEYS, 18);
                  // customer attempts to buy 18 Hersheys, only receives 17
   System.out.println(vend.buy(VendingMachine.HERSHEYS, 3);      // prints 0
   System.out.println(vend.get(VendingMachine.HERSHEYS));// How many left? 0
   vend.supply(VendingMachine.HERSHEYS, 5); // ran out of HERSHEYS, add more
   System.out.println(vend.buy(VendingMachine.STARBURST, 1);     // prints 1
   System.out.println(vend.buy(VendingMachine.HERSHEYS, 10);     // prints 5
   . . .
}
```

The class interface and method comments appear below and continue onto the next page. We left space so you can add in your code to complete the class:

```
//   VendingMachine models the inventory inside a candy machine.
//   interface invariant: the amount of each kind of candy in the machine
//                        is always non-negative
public class VendingMachine {
   public static final int HERSHEYS = 0;
   public static final int M_AND_MS = 1;
   public static final int STARBURST = 2;
   public static final int NUM_KINDS = 3;









   //   Creates an empty vending machine object
   public VendingMachine () {









   }
```

Problem 3 (cont.)

```
   // PRECONDITION for parameters to get, buy, and supply:
   // -- kind is one of HERSHEYS, M_AND_MS, and STARBURST
   // -- quantity > 0   [applies to buy and supply only]

   // Returns the amount in this machine of the kind of candy specified
   // PRECONDITION: see above
   public int get(int kind) {









   }

   // Attempt to buy the given quantity of the kind of candy specified
   // from this vending machine.  Returns the actual number of pieces
   // given out, based on the current inventory (i.e., it can't give out
   // more than is currently in the machine).
   // PRECONDITION: see above
   public int buy(int kind, int quantity) {









   }

   //  Add to this vending machine the given quantity of the kind of candy
   //     specified.
   // PRECONDITION: see above
   public void supply(int kind, int quantity) {









   }
}
```

Problem 3 (cont.)

Part B [3]. Suppose we now want to add a fourth type of candy in our `VendingMachines`. What parts of your `VendingMachine` class code would have to change to handle the fourth candy type? Circle all that apply below (we started it for you):

(a.) comments

b. constant definitions

c. instance variable definitions

d. constructor

e. `buy` method

f. `supply` method

Part C [2]. Show the variable definition(s) for data structure that would allow us to store information about all the `VendingMachines` in a building:

Part D [5]. Give the code to initialize your data structure from part C so that there are 10 empty vending machines in the building, or say "already done", if your data structure was implicitly initialized this way by the code you gave for part C.

Problem 4 [8 pts. total]

The following method doesn't work. The method comment describes what it is *supposed* to do.

```
/**
   Reads an indeterminate number of non-negative values from the user,
   using -1 as a sentinel, and returns the sum of the numbers
   @param in   the scanner to read from
   @returns    the sum of the non-negative numbers read
*/
public static int sumInts(Scanner in) {

  System.out.print("Enter numbers when prompted.  ");

  System.out.println("Use -1 to indicate you are done.");

  int num = 0;

  int sum = 0;

  while (num != -1) {

    System.out.print("Enter number (-1 to finish): ");

    num = in.nextInt();

    sum += num;

  }

  return sum;

}
```

Part A. Show two example data sets for which the method returns the wrong result and what the method returns in that case. The second one of these must be a boundary case.

	numbers entered by the user	return value of **`sumInts(new Scanner(Sytem.in))`**
a.		
b. (boundary case)		

Part B. Fix the code above. Do not rewrite the whole method, but rather make your changes right into the code above, using arrows to show where your code should be inserted, crossing out code that you would get rid of, etc.

Problem 5 [20 pts]

Write the static Java `int[]` method `divTenFirst` which takes an array of numbers and returns an array with the same values as the original, but with all the numbers that are divisible by 10 appearing before all of the numbers that aren't divisible by 10. Other than that, the numbers may appear in any order. ***The array*** **`nums`** ***should be unmodified by the method.***

Here are some examples (because of the ordering flexibility, your code may get different results):

`nums`	**`divTenFirst(nums)`**
[3, 2, 6, 10]	*[10, 3, 2, 6]*
[0, 10, 30, 0]	*[10, 30, 0, 0]*
[20, 5, 200, 3, 8, 20]	*[20, 200, 20, 5, 3, 8]*
[3, 2, 7, 7, 1]	*[3, 2, 7, 7, 1]*
[20, 20, 20]	*[20, 20, 20]*
[1]	*[1]*
[]	*[]*

```
public static int[] divTenFirst(int[] nums) {
```

Extra space for answers or scratch work.

If you put any of your answers here, please write a note on the question page directing us to look here. Also label any such answers here with the question number and part, and circle the answer.

Extra space for answers or scratch work (cont.)

If you put any of your answers here, please write a note on the question page directing us to look here. Also label any such answers here with the question number and part, and circle the answer.