COMPUTER SCIENCE A
SECTION II

Time—1 hour and 45 minutes
Number of questions—4
Percent of total grade—50

Directions: SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

Notes:
- Assume that the classes listed in the Quick Reference found in the Appendix have been imported where appropriate.
- Unless otherwise noted in the question, assume that parameters in method calls are not `null` and that methods are called only when their preconditions are satisfied.
- In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods may not receive full credit.

1. A positive integer is called a "self-divisor" if every decimal digit of the number is a divisor of the number, that is, the number is evenly divisible by each and every one of its digits. For example, the number 128 is a self-divisor because it is evenly divisible by 1, 2, and 8. However, 26 is not a self-divisor because it is not evenly divisible by the digit 6. Note that 0 is not considered to be a divisor of any number, so any number containing a 0 digit is NOT a self-divisor. There are infinitely many self-divisors.

```
public class SelfDivisor
{
  /** @param number the number to be tested
   *          Precondition: number > 0
   *  @return true if every decimal digit of number is a divisor of number;
   *          false otherwise
   */
  public static boolean isSelfDivisor(int number)
  {  /* to be implemented in part (a) */  }


  /** @param start starting point for values to be checked
   *          Precondition: start > 0
   *  @param num the size of the array to be returned
   *          Precondition: num > 0
   *  @return an array containing the first num integers ≥ start that are self-divisors
   */
  public static int[] firstNumSelfDivisors(int start, int num)
  {  /* to be implemented in part (b) */  }


  // There may be fields, constructors, and methods that are not shown.
}
```

(a) Write method `isSelfDivisor`, which takes a positive integer as its parameter. This method returns `true` if the number is a self-divisor; otherwise, it returns `false`.

Complete method `isSelfDivisor` below.

```
/** @param number the number to be tested
 *          Precondition: number > 0
 *  @return true if every decimal digit of number is a divisor of number;
 *          false otherwise
 */
public static boolean isSelfDivisor(int number)
```

(b) Write method `firstNumSelfDivisors`, which takes two positive integers as parameters, representing a start value and a number of values. Method `firstNumSelfDivisors` returns an array of size `num` that contains the first `num` self-divisors that are greater than or equal to `start`.

For example, the call `firstNumSelfDivisors(10, 3)` should return an array containing the values 11, 12, and 15, because the first three self-divisors that are greater than or equal to 10 are 11, 12, and 15.

In writing `firstNumSelfDivisors`, assume that `isSelfDivisor` works as specified, regardless of what you wrote in part (a).

Complete method `firstNumSelfDivisors` below.

```
/** @param start starting point for values to be checked
 *          Precondition: start > 0
 *  @param num the size of the array to be returned
 *          Precondition: num > 0
 *  @return an array containing the first num integers ≥ start that are self-divisors
 */
public static int[] firstNumSelfDivisors(int start, int num)
```

2. This question involves reasoning about the code from the Marine Biology Simulation case study. A copy of the code is provided as part of this exam.

   A `PounceFish` is a type of fish that looks for prey and then "pounces" on it. A `PounceFish` can see only a limited distance in its forward direction. If the `PounceFish` sees another fish, it rushes forward and eats the nearest one that it sees, ending up in the location where its prey was originally located. If the `PounceFish` does not see another fish, it acts as a `Fish`.

   The `PounceFish` class is shown below.

```
public class PounceFish extends Fish
{
  private int range;   // the distance that a PounceFish can see; range > 0


  /** Looks ahead range locations in current direction
   *  @return the nearest fish in that direction within range (if any);
   *          null if no such fish is found
   */
  private Fish findFish()
  {  /* to be implemented in part (a) */  }


  /** Acts for one step in the simulation
   */
  public void act()
  {  /* to be implemented in part (b) */  }


  // There may be fields, constructors, and methods that are not shown.
}
```

AP® COMPUTER SCIENCE A
2007 SCORING GUIDELINES

Question 1: Self Divisor

Part A: isSelfDivisor **4 points**

+2 loop over digits
- **+1** access digit *in context of loop*
  - **+1/2** attempt (`number % ?` or successfully convert to string represent)
  - **+1/2** correct
- **+1** process all digits
  - **+1/2** attempt (process multiple digits)
  - **+1/2** correct

+2 classify `number`
- **+1/2** return false if find 0 digit
- **+1/2** test if divisible (`number %` digit)
- **+1/2** return false if find non-divisor digit
- **+1/2** return true self divisor

lose both of these if return a value in both cases of an if-else

Part B: firstNumSelfDivisors **5 points**

+1 initialize
- **+1/2** create and initialize array of size `num`
- **+1/2** create and initialize index counter

+3 1/2 loop to find self divisors
- **+1/2** iterate through numbers beginning with `start`
- **+1/2** call `isSelfDivisor` on number
- **+1 1/2** add self divisor to array
  - **+1/2** attempt (store self divisor in some array index)
  - **+1** correct (store in correct index, including increment)
- **+1** loop and store `num` values in array
  - **+1/2** attempt (must reference index counter and `num`)
  - **+1/2** correct

+1/2 return array (lose this if return first time through loop)

AP® COMPUTER SCIENCE A
2007 CANONICAL SOLUTIONS

Question 1: Self Divisor

PART A:

```
public static boolean isSelfDivisor(int number) {
     int n = number;
     while (n > 0) {
         int digit = n % 10;
         if (digit == 0 || number % digit != 0) {
             return false;
         }
         n /= 10;
     }
     return true;
}
```

ALTERNATE SOLUTION:

```
public static boolean isSelfDivisor(int number) {
     String str = "" + number;
     for (int i = 0; i < str.length(); i++) {
         int digit = Integer.parseInt(str.substring(i,i+1));
         if (digit == 0 || number % digit != 0) {
             return false;
         }
     }
     return true;
}
```

PART B:

```
public static int[] firstNumSelfDivisors(int start, int num) {
    int[] selfs = new int[num];
    int numStored = 0;
    int nextNumber = start;
    while (numStored < num) {
        if (isSelfDivisor(nextNumber)) {
            selfs[numStored] = nextNumber;
            numStored++;
        }
        nextNumber++;
    }
    return selfs;
}
```

ALTERNATE SOLUTION:

```
public static int[] firstNumSelfDivisors(int start, int num) {
    int[] selfs = new int[num];
    int numStored = 0;
    int nextNumber = start;
    for (int i = 0; i < num; i++) {
        while (!isSelfDivisor(nextNumber)) {
            nextNumber++;
        }
        selfs[numStored] = nextNumber;
        numStored++;
        nextNumber++;
    }
    return selfs;
}
```