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;
}
```

Ala,

(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)
{
    int temp = number;

    while (temp != 0)
    {
        int digit = temp % 10;
        temp = temp / 10;
        if (digit == 0)
            return false;
        if (number % digit != 0)
            return false;
    }

    return true;
}
```

Part (b) begins on page 6.

Hla2

(b) Write method `firstNumSelfDivisors`, which takes two positive integers as parameters, represe a start value and a number of values. Method `firstNumSelfDivisors` returns an array of size r 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 v 11, 12, and 15, because the first three self-divisors that are greater than or equal to 10 are 11, 12, and 1

In writing `firstNumSelfDivisors`, assume that `isSelfDivisor` works as specified, regard 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)
{
   int[] array = new int[num];
   int index = 0;
   for (int i = start; index < num; i++)
   {
      if (isSelfDivisor(i))
      {
         array[index] = i;
         index++;
      }
   }
   return array;
}
```

GO ON TO THE NEXT PAGE

(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)
{
  String Num = N umber;
  int count=0;
  while(count< Num.length())
{ stringcheck = Num.substring(count, count+1);
  if(check.equals("0"))
    return false;
  if(!(Num/check).equals((double)(Num/check)))
  return false;
  count++;
}

  return true;
```

Part (b) begins on page 6.

A1b2

(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)
{ int[] selfDivisor = new int[num];
  int check= start;
  int count = 0;
  while(count<num)
  { if( isSelfDivisor(check))
    { selfDivisor[count] = check;
      count++; }
    check++; }
  return selfDivisor[];
}
```

GO ON TO THE NEXT PAGE.

A1 c,

(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)
{
  if(number != 0 && number > 0){
    int h = number / 100;
    int t = (number - (h * 100)) / 10;
    int o = number - (h * 100) - (t * 10);
    if(number % h = 0 && number % t = 0 && number % o = 0)
    {
      return true;
    }
  }
  return false;
}
```

Part (b) begins on page 6.

A1c2

(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)
{  int count = 0;
   int[] numbers;
   int r = num
   while(count < num)
   {
      r++;
      if( isSelfDivisor(r))
      {
         count++;
         numbers.add(r);
      }
   }
return numbers;
}
```

GO ON TO THE NEXT PAGE.

AP® COMPUTER SCIENCE A
2007 SCORING COMMENTARY

Question 1

Overview

This question focused on algorithm design and implementation, as well as array manipulation. A property of integers was described, that of being a self-divisor (when every digit of the number evenly divides the number). In part (a) students were required to implement the `isSelfDivisor` method for determining whether a given number is a self-divisor. This could be accomplished in numerous ways, for example, by repeatedly extracting the rightmost digit (using the remainder operator) and then dividing by 10, or by converting the number to a string and then extracting digits as characters. In part (b) students were required to implement a method for finding and collecting self-divisors in an array. The `firstNumSelfDivisors` method had two parameters, the starting number in a range and the number of desired self-divisors. The method was intended to find self-divisors from the range, place them in an array, and return that array. This involved looping through numbers, starting at the start number, and calling the `isSelfDivisor` method from part (a) to identify self-divisors.

Sample: A1a
Score: 9

The solution is completely correct for both parts (a) and (b) and was awarded all points for this question. In part (a) the code correctly loops over the digits of `number`. The student sets the variable `temp` equal to `number` before entering the loop and then mods `temp` by 10 within the loop. The student then does integer division on `temp` so that the loop terminates when all digits of `temp` have been processed. The student correctly checks to see if `digit` is zero, returning `false` if it is. The student checks the original number mod `digit` to determine if it is not a divisor of `number`. The student correctly returns `false` if `digit` does not divide evenly into `number`. The student returns `true` at the end of the loop correctly coding the fact that if none of the values of `digit` failed to divide evenly into `number` then `number` is a self-divisor.

In part (b) the code correctly constructs `array` of the correct size and constructs and initializes `index` as an index variable for `array`. The student then initializes index variable, `i`, of the `for` loop with `start` and correctly increments `i` each time through the loop. The student also uses the correct check, `index < num`, to stop the loop, thereby earning the loop and store correct number of values in the array with this `for` loop. The student then checks the loop control variable, `i`, to determine if it is a self-divisor. If `i` is a self-divisor, it is correctly added to `array`. The student then correctly returns `array` after the loop is complete.

AP® COMPUTER SCIENCE A
2007 SCORING COMMENTARY

Question 1 (continued)

Sample: A1b
Score: 7

In part (a) the student's code assigns the number to a `String` incorrectly thereby losing the ability to access the individual digits within the loop. Because of this error the student lost the two ½ points awarded for accessing the individual digits of `number`. The student then constructs a loop, which gives the ability to access all of the digits within the string. The student correctly checks for zero using the `equals` method to compare two string values. The student then tries to ascertain whether `number` divided by the digit has a remainder but does it using two different types and therefore lost the test if divisible ½ point. The student incorrectly compares the two values when returning `false` on a nonself-divisor, so the return false ½ point was lost. The student returns `true` after the loop completed, correctly coding the fact that if each digit divided evenly into `number` then `number` is a self-divisor.

For part (b) the code correctly initializes an array of the correct size and initializes an index variable for this array. The student then loops the correct number of times to store the correct number of values in the array. The student had preset the variable `check` to `start` before starting the loop and correctly increments it through each iteration of the loop and thus earned the iterate through numbers beginning with `start` ½ point. Within the loop the student calls the method `isSelfDivisor` on `check` and if it is a self-divisor, correctly adds it to the array `selfDivisor`. The student correctly returns the array `selfDivisor` after the loop has completed.

Sample: A1c
Score: 3½

In part (a) the student does not use a loop and therefore lost the 2 points allocated for looping over digits. The student does not check to see if the potential divisor is zero, so the ½ point awarded for that check was lost. The student correctly checks to see if a digit divides evenly into `number` and was awarded that ½ point. Because the student does not use a loop, the conditional statement at the end of the code is in effect an `if then ... else` statement. Therefore the student lost both ½ points for the return statements.

In part (b) the student incorrectly initializes the array `numbers` and lost the initialize array ½ point. However, the student correctly initializes the index variable, `count`, for this array and received the initialize index ½ point. Because the student initializes `r` with `num` rather than `start`, the iterate beginning with `start` ½ point was lost. The student correctly calls `isSelfDivisor` on `r` and earned that ½ point. Because the student uses `add` to place `r` into the array `numbers`, the point for correctness was lost. The student loops the correct number of times and attempts to store the correct number of values in the array and earned both the attempt and the correctness ½ points. The student then returns the array after the loop and earned that ½ point.