# Pre-Midterm #1 Review

In questions 2–5, mark ALL correct answers for each question.

1. Write a method which accepts a string and a character, and determines whether or not the character occurs in the string. (The beginning and the end should be as below, just fill in the body of the method.)

```
static boolean occurs (String st, char ch)
{
    boolean appears = false;

    <BODY>

    return appears;
}
```

2. Consider the following code section:

```
final int LENGTH = 10, SIZE = 25;
int[] myArray = new int[LENGTH];
for (int i = 0; i < LENGTH; i = i + 1)
{
   myArray[i] = (int) (SIZE * Math.random());
   for (int j = 0; j < i; j = j + 1)
      System.out.print (1 /(myArray[i] - myArray[j]) + " , ");
   System.out.println ();
}
```

(a) The code compiles and runs without errors.

(b) The code compiles without errors and may terminate its execution normally. In this case, its output consists of 45 numbers. All these numbers belong to a certain set consisting of 3 elements.

(c) The code compiles without errors. There may be a runtime error, but the code may also run without any problem. If there is an error, then prior to this error there may be output consisting of anywhere between 0 and 44 numbers.

(d) There is a runtime error after the program prints 5 lines of output.

(e) The code cannot run without errors if the initial value of LENGTH is 100 instead of 10.

(f) None of the above.

3. Consider the following code section:

```
static byte length (long number, int base)
{
   byte l = 1;
   while (number >= base)
   {
      number = number / base;
      l = (byte) (l + 1);
   }
   return l;
}
```

(a) For any value of *number* and any $base \geq 2$, the method calculates the length of the representation of *number* in base *base*.

(b) For any value of $number \geq 0$ and any $base \geq 2$, the method calculates the length of the representation of *number* in base *base*.

(c) If we wanted only the length of the representation of $number \geq 0$ in base 10, then we could replace the body of the method by:
$return\ (byte)\ (number\ +\ "").length();$
(where " " is an empty string).

(d) Since the length of the representation of *number* in base *base* is 1 more than that of the representation of the integer value of $number/base$, we could replace the body of the method by:
$return\ (byte)\ (1\ +\ length\ (number\ /\ base,\ base));$

(e) The suggestion in (d) would work correctly if and only if $number \geq base$.

(f) None of the above.

4. Consider the concatenation operation + on data of type String.

(a) The operation is associative but non-commutative.

(b) The equality $st1 + st2 = st2 + st1$ holds if and only if $st1 = st2$.

(c) There exists an infinite set $S$ of strings such that $st1 + st2 = st2 + st1$ for every $st1, st2 \in S$.

(d) There exists an infinite set $S'$ of strings such that $st1 + st2 \neq st2 + st1$ for every $st1, st2 \notin S'$.

(e) If $st1 + st2 = st2 + st1$ and $st1 + st3 = st3 + st1$, then $st2 + st3 = st3 + st2$.

(f) None of the above.

5. Consider the following code section, where *primes* is of type *int*[], and contains all primes up to 1000, and $m$ and $n$ are any two positive integers of type *int*.

```
int gcd = 1;
for (int i = 0; i < primes.length; i = i + 1)
   if (n % primes[i] == 0 & m % primes[i] == 0)
   {
      gcd = primes[i] * gcd;
      m = m / primes[i];
      n = n / primes[i];
   }
```

(a) The code evaluates correctly the greatest common divisor for any $m$ and $n$.

(b) There are many pairs of integers $m$ and $n$ for which the answer will come out wrong.

(c) The code works correctly if and only if $m, n \leq primes[i]$.

(d) There are pairs of integers $m, n \leq primes[i]$ for which the code returns an incorrect answer.

(e) There exist pairs for which the code returns a *gcd* value equal to $gcd(m, n)$, there exist pairs for which *gcd* is smaller than $gcd(m, n)$, and there exist pairs for which *gcd* is larger than $gcd(m, n)$.

(f) None of the above.