CS 11 Spring 2008 — Mid-term exam 1
Answer Key

1. [Question:] (15 points) Consider the code fragment below. **Mark each location** where an *automatic cast* will occur. Also find each location where an *explicit cast* must be inserted for the code to compile successfully, and **correct the line** with that explicit cast included. [Note: Be aware that some lines may require more than one cast!]

```
byte b = 13;
int i = b;
char c = i;
int i2 = c + i;
char c2 = c + 3;
boolean b = (i < c);
```

[Answers and discussion:] People frequently were confused about which casts were automatic and which had to be forced/explicit. Many people indicated that a particular line involved an automatic cast, but they did not make clear *to which portion* of the expression that cast would apply, thus losing some credit. We will address one line at a time:

- `byte b = 13;`
  No cast is performed or needed here. The compiler identifies `13` as a valid `byte` constant, and so this is a case of assigning a `byte` value into a `byte` space. If you believed that the `13` would be considered an `int`, then you would have to insert a *forced cast* to receive partial credit. Claiming an *automatic cast* here is not sensible.
- `int i = b;`
  An **automatic cast** on `b` occurs here, converting it from a `byte` to an `int`.
- `char c = (char)i;`
  A **forced cast** must be inserted on `i` to make that `int` value fit into a `char` space.
- `int i2 = c + i;`
  In order to add a `char` to an `int`, the compiler will perform an **automatic cast** on `c`, converting it to an `int` so that addition is really performed on two `int` values.
- `char c2 = (char)(c + 3);`
  Two casts must occur here. First, addition is only performed on `int` values. So, `c` is **automatically cast** to an `int`. Second, to assign the result of the `int` addition into a `char` space, a **forced cast** must be performed on the addition expression. Many people caught the
- `boolean b = (i < c);`
  Comparisons must be performed on like types. Therefore, `c` is **automatically cast** to an `int` so that it may be compared to `i`. As an alternate answer, one could insert a **forced case** on `i`, making it into a `char`, thus comparing two `char` values.
  Additionally, note that there is an error in this line. In the first line of the problem, `b` was already declared as a `byte`. Obviously, the question should have, on this line, declared `boolean b2`.

2. [Question] (15 points) Consider the Java code below and answer the questions that follow.

```
System.out.println("Enter a value for a: ");
int a = User.readInt();
System.out.println("Enter a value for b: ");
int b = User.readInt();

if (a < 0) {
    a = -a;
}

int i = 0;
while (i < a) {

  int k = 0;
  for (int j = 100; j >= b; j--) {
    k = j * 2;
    System.out.print(k);
    System.out.print('-');
    System.out.println(j);
  }

  k++;
  i = i + 1;

}
```

(a) What is the output of this code if the user enters 2 for `a` and 98 for `b`?

(b) What is the output of this code if the user enters -3 for `a` and 100 for `b`?

[Answer] For each case:

(a)
```
200-100
198-99
196-98
200-100
198-99
196-98
```

(b)
```
200-100
200-100
200-100
```

[Discussion] The most common errors were to get the number of repetitions of any or all of the loops incorrect. (Interestingly, many people terminated the loops too soon, and nobody believed that there were *extra* repetitions.)

3. [Question] (10 points) The following method contains an error that will prevent it from compiling. **Find and correct it**.

```
public static int getPositiveInt () {

    do {

        System.out.print("Enter a positive integer: ");
        int x = Keyboard.readInt();
        if (x <= 0) {
            System.out.println("No good, try again: ");
        }

    } while (x <= 0);

    return x;

}
```

[Answer] The scope of x is too narrow. It ceases to exist at the closing brace before the `while` keyword, and thus the use of it in the continuing condition and in the return statement is invalid. To fix it, `x` must be declared before the *do-while* loop, and its previous declaration must be turned into a simply assignment:

```
public static int getPositiveInt () {

    int x;
    do {

        System.out.print("Enter a positive integer: ");
        x = Keyboard.readInt();
        if (x <= 0) {
            System.out.println("No good, try again: ");
        }

    } while (x <= 0);

    return x;

}
```

[Discussion] Many people found the basic problem with this question. The most common error was to declare `x` before the *do-while*, but then not remove (or not specify the removal of) the existing declaration of that variable. A few people more radically restructured the method, obtaining an initial input from the user before the loop. Although that modification was more extensive than necessary, it received full credit if done correctly.
It is worth noting that `Keyboard.readInt()` differs slightly from our usual `keyboard.nextInt()`. Although the two are different, they perform an identical task, and few people were distracted by this detail. A few focused on it; if you did, be aware that I would not base a question like this one on a that kind of typo. To do so would be to ask a question that reveals nothing about your understanding of programming and algorithms.

4. [Question] (30 points) **Write a method** named `printTable` that accepts a *height* and a *width* as parameters and then prints a table of those dimensions, where the table follows the following pattern:

```
1 2 3 4 5
2 4 6 8 10
3 6 9 12 15
4 8 12 16 20
```

The above table is the result of calling `printTable(4, 5)`.

[Answer] Although there are many ways to write a method that perform this task, here is one of the simplest:

```
public static void printTable (int height, int width) {

  for (int row = 1; row <= height; row++) {
    for (int column = 1; column <= width; column++) {
      System.out.print((row * column) + " ");
    }
    System.out.println();
  }

} // end printTable
```

[Discussion] Many people devised solutions that were anywhere from somewhat to excessively complex. Either through overly involved arithmetic expressions to control loops, or the excessive use of extra, supporting variables, people managed to confuse themselves and construct incorrect answers. In general, correct but modestly more complex answers received full credit, while correct but substantially obfuscated solutions lost a little credit.
A common error was to write an inner loop in which the value to be printed was advanced using an expression something like: `y = y + y`. This expression doubles `y` with each iteration, thus violating the pattern (which increases by a constant across a row).

5. (30 points) We want to write a program that allows the user to enter a list of positive numbers and then prints out the *mean* number from that list. That is, if the user enters the values 7, 4, 3.5, and 11.5, then the mean is $\frac{7+4+3.5+11.5}{4} = \frac{26}{4} = 6.5$.
The `main` method of this program is:

```
public static void main (String[] args) {

    double x = getMeanFromUser();
    System.out.println("Max = " + x);

}
```

Write the method `getMeanFromUser`. The user should be able to enter an arbitrary number of positive values. As soon as the user enters a non-positive value, the program should accept that value as an indication that the user has no more values to enter. After the user has entered all of her numbers, the method should return the mean value entered. Do not worry about a user that enters non-numeric values.

[Answer] Once again, there is more than one way to skin a cat, but here is one straightforward solution:

```
public static double getMeanFromUser () {

  double userEntry = 0.0;
  double sum = 0.0;
  int numberEntries = 0;
  while (userEntry > 0.0) {

    System.out.print("Enter next value (or a negative value if finished): ");
    userEntry = keyboard.nextDouble();
    if (userEntry > 0.0) {
      sum = sum + userEntry;
      numberEntries++;
    }

  }

  return sum / numberEntries;

} // end getMeanFromUser
```

[Discussion] First, note the output typo in which the result is presented as the *max*. That superficial error should not have thrown anyone, given the rest of the text of the question.
Most errors on this problem simply involved a failure to keep initialize and update the correct pieces of information at the correct times. A few emitting output by using `System.out.println` instead of returning the mean. Many included entries of `0.0` itself as valid additions to the sequence of numbers, but the question unambiguous refers to *positive* numbers, where the first *non-positive* value—that is, `0.0` or a negative value—would terminate the entry of new values.