Sarajevo School of Science and Technology
CS/IS 160 Programming and Problem Solving II
April 20, 2015

# Midterm Exam Solutions

**(Midterm exam: 20% of the final grade, 90 minutes)**

## Problem 1 (10 points, each part 2.5 points).

Answer the following questions.

(a) What is the difference between a local variable and an instance variable?

*Solution*: Local variables are defined in a code block or a method and have a local scope, while instance variables are defined inside of a class definition. Thus, the main difference is in the scope of the variables.

(b) Briefly explain what is a package in Java? Why are they useful?

*Solution*: The packages are used to organize and package code with similar functionality. They are useful since they organize the code and provide unique namespaces for the packaged methods and classes.

State whether each of the following statements is true or false. If false, explain why.

(c) An array index should normally be of type `double`.

*Solution*: False. An array index should be of an integer type.

(d) Reference-type instance variables are initialized by default to the value of `0`.

*Solution*: By default, reference-type instance variables are initialized to `null.`

## Problem 2: Simple statements and declarations (10 points, each part 2.5 points).

(a) Declare and initialize a private string array `cities` with enough space for 10 strings.

*Solution*: `private String[] cities = new String[10];`

(b) Declare and initialize a public two-dimensional double array named `matrix` with 1000 rows and 10 columns.

*Solution*: `public double[][] matrix = new double[1000][10];`

(c) Declare and initialize a local integer array `niz` that can hold 100 integers. Now write a for-loop that populates this array with integers from 1 to 100.

*Solution*:

```
int[] niz = new int[100];

for( int i=0; i < 100; i++ ) {
    niz[i] = i+1;
}
```

(d) Write a for-loop that iterates over the array `niz` and doubles each of its elements. Therefore, at the end, the array `niz` will hold integers 2, 4, 6, 8…

*Solution*:

```
for( int i=0; i < 100; i++ ) {
    niz[i] *= 2;
}
```

**Problem 3: Sum of diagonal entries of a square matrix (20 points).**

Implement a static method `diagSum`, which returns the sum of the diagonal entries of a square matrix. For example, if the input matrix is given as

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix},$$

then the sum of its diagonal entries is 1 + 5 + 9 = 15.

Use the following method prototype, where `matrix` is an input integer square matrix:

```
public static int diagSum( int[][] matrix )
```

*Solution*:

```
public static int diagSum( int[][] matrix ) {

      int sum = 0;

      for( int i=0; i < matrix.length; i++ ) {
            sum += matrix[i][i];
      }

      return sum;
}
```

**Problem 4: Array intersection (20 points).**

We are given two integer arrays `a` and `b`. Write a static method `arrayIntersect`, which will return true if the array `b` completely appears (in order) in the array `a`, and will otherwise return false. Therefore, this method checks if the array `b` intersects `a`.

As an example, the method should return true if the array `a` is (1,2,3,4,5), and `b` is (2,3). The method should return false if the array `a` is (1,2,3) and `b` is (2,4).

Use the following method prototype:

```
public static boolean arrayIntersect( int[] a, int[] b )
```

*Solution*: In this problem, we need to loop over all possible sub-arrays of the first input array and see if any of those sub-arrays are exactly the same as the second input array.

```
public static boolean arrayIntersect( int[] a, int[] b ) {

    int i, j;

    for( i=0; i < a.length; i++ ) {

        for( j=0; j < b.length && (i+j) < a.length; j++ ) {
            if( a[i+j] != b[j] ) {
                break;
            }
        }

        if( j == b.length ) {
            return true;
        }
    }

    return false;
}
```

**Problem 5: Filter small entries of an array (20 points).**

Implement a static method `filterLessThan`, which returns the elements of an array that are smaller than an input threshold value as a new array.

For example, if the input array is (1, 6, 3, 4, 10, 5) and the filter threshold value is 5, then the returned array is (1, 3, 4).

Use the following method prototype, where `array` is an input integer array and `val` is the threshold value:

```
public static int[] filterLessThan( int[] array, int val )
```

*Solution*:

```
public static int[] filterLessThan(int[] array, int val) {

        int count = 0;

        // count number of filtered items
        for (int i = 0; i < array.length; i++) {
                if (array[i] < val)
                        count++;
        }

        // initialize filtered array and populate it
        int[] result = new int[count];
        for (int i = 0, j = 0; i < array.length; i++) {
                if (array[i] < val)
                        result[j++] = array[i];
        }

        return result;
}
```

**Problem 6: Meal class (20 points, each part 5 points).**

We define a class `Meal` with two private instance variables: `name` (String) and `price` (double), which represents the name and base price of a meal in KM. Note that the price cannot be set to zero or a negative number. Therefore, print out an error message if price is set to an invalid number.

We use the following Java class definition:

```
// class definition
public class Meal {

     // instance variables
     private String name;
     private double price;

     // getters and setters (part a and b)
     public String getName() {}
     public double getPrice() {}

     public void setName(String inputName) {}
     public void setPrice(double inputPrice) {}

     // public instance methods (part c and d)
     public boolean isExpensive() {}
     public double getDiscountPrice() {}

}
```

(a) Implement the getter method for the private variable `price.`

*Solution*:
```
     public double getPrice() {
          return price;
     }
```

(b) Implement the setter method for the private variable `price.`

*Solution*:

```
     public void setPrice(double inputPrice) {
          if(inputPrice > 0) {
             price = inputPrice;
          }
          else {
             System.out.println("Error: Price must be positive.");
          }
     }
```

(c) Implement the public instance method `isExpensive`, which returns true if the meal price is more than 50 KM.

*Solution*:

```
     public boolean isExpensive() {
          return (price > 50);
     }
```

(d) Implement the public instance method `getDiscountPrice`, which returns a 10% discounted price for the meal. *Note*: Here we want to return a price with 10% taken off.

*Solution*:

```
     public double getDiscountPrice() {
          return 0.9*price;
     }
```