1 Problem solving with arrays and numbers

Please go through the concept of "violation" algorithms vs. "validation" algorithms

1. Write a method that when passed an integer, returns the last digit. You may assume that the integer is positive.

Solution:

```
int lastDigit(int n) {
        return n % 10;
}
```

2. Write a method that when passed an integer, returns the sum of its digits. You may assume that the integer is positive.

Solution: Iterative:

```
int sumDigits(int n) {
        int result = 0;
        while(n > 0) {
                result+=n%10;
                n = n / 10;
        }
        return result;
}
```

Recursive:

```
int sumDigits(int n) {
        if(n == 0) {
                return 0;
        }
        return n % 10 + sumDigits(n / 10);
}
```

3. Write a method that when passed two integers, returns their greatest common divisor (largest integer that divides both). For example, gcd of 75 and 30 is 15 while gcd of 8 and 15 is 1. You may assume both integers passed are positive.

Solution:

```
int gcd(int a, int b) {
        int smaller;
```

```
        if(a < b)
                smaller = a;
        else
                smaller = b;
        for(int i=smaller; i > 1; i--)
                if(a%i == 0 && b%i == 0)
                        return i;
        return 1;
}
```

4. Write a method that when passed three integers, returns the highest of the three.

Solution: Cheeky answer: (because I didn't say that you cannot use built-in methods)

```
        int highest(int a, int b, int c) {
                return Math.max(a, Math.max(b, c));
        }
```

From scratch:

```
int highest(int a, int b, int c) {
        if(a >= b && a >= c)
                return a;
        if(b >= c)
                return b;
        return c;
}
```

Or:

```
int highest(int a, int b, int c) {
        if(a >= b)
                if(a >= c)
                        return a;
                else
                        return b;
        else
                if(b >= c)
                        return b;
                else
                        return c;
}
```

5. Write a method that when passed an array of integers, returns the number of negative numbers in the array. You may assume that the array is not `null`.

Solution:

```
int countNegatives(int[] arr) {
        int count = 0;
        for(int i=0; i < arr.length; i++) {
                if(arr[i] < 0) {
                        count++;
                }
        }
        return count;
}
```

Using built-in iterator:

```
int countNegatives(int[] arr) {
        int count = 0;
        for(int item: arr) {
                if(item < 0) {
                        count++;
                }
        }
        return count;
}
```

6. Write a method that when passed an array of `double` values and another `double` value (*key*), returns the number of times *key* exists in the array. You may assume that the array is not `null`.

Solution:

```
int countOccurrences(double[] arr, double key) {
        int count = 0;
        for(double item: arr) {
                if(item == key) {
                        count++;
                }
        }
        return count;
}
```

7. Write a method that when passed two arrays of `double` values, returns `true` if they are identical, and `false` otherwise. You may assume that each array has at least one item.

Solution:

```
boolean identical(double[] a, double[] b) {
        if(a.length != b.length)
                return false;

        for(int i=0; i < a.length; i++) {
                if(a[i] != b[i]) {
                        return false;
                }
        }

        return true;
}
```

8. (Challenging) Write a method that when passed an array of integers, return `true` if there is no item in the array that occurs more than once, and `false` otherwise.

Solution: Solution 1:

```
boolean noDuplicates(int[] arr) {
        for(int i=0; i < arr.length; i++) {
                for(int k=i+1; k < arr.length; k++) {
                        if(arr[i] == arr[k]) {
                                return false;
                        }
                }
        }
        return true;
}
```

Solution 2 (delegated):

```
int countOccurrences(int[] arr, int key) {
        int count = 0;
        for(int item: arr) {
                if(item == key) {
                        count++;
                }
        }
        return count;
}
```

boolean noDuplicates(int[] arr) for(int item: arr) if(countOccurrences(arr, item) ¿ 1) return false; return true;

2 Classes and Objects

9. Write a class definition, including instance variables, getters and setters, default and parameterized constructors, area(), and compareTo method to define a Circle, as represented by its radius.

Solution:

```
public class Circle {
        private double radius;

        public void setRadius(double r) {
                if(r < 0)
                        radius = 0;
                else
                        radius = r;
        }

        public double getRadius() {
                return radius;
        }

        public Circle() {
                setRadius(0);
        }

        public Circle(double r) {
                setRadius(r);
        }

        public double area() {
                return Math.PI * radius * radius;
        }

        public int compareTo(Circle other) {
                if(radius > other.radius)
                        return 1;
                if(radius < other.radius)
                        return -1;
                return 0;
        }
}
```

10. Write a class definition, including instance variables, getters and setters, default and parameterized constructors, perimeter(), and compareTo method to define a Rectangle, as represented by its width and height.

Solution:

```
public class Rectangle {
        private double width, height;

        public void setWidth(double w) {
                if(w < 0) {
                        w*=-1;
                }
                width = w;
        }

        public void setHeight(double h) {
                if(h < 0) {
                        h*=-1;
                }
                height = h;
        }

        public double getWidth() {
                return width;
        }

        public double getHeight() {
                return height;
        }

        public Rectangle(double w, double h) {
                setWidth(w);
                setHeight(h);
        }

        public Rectangle() {
                setWidth(0);
                setHeight(0);
        }

        public double area() {
                double result = width * height;
                return result;
        }

        public boolean isSquare() {
                if(width == height)
                        return true;
                else
```

```
                return false;
        }

        public int compareTo(Rectangle other) {
                if(area() > other.area())
                        return 1;
                if(area() < other.area())
                        return -1;
                return 0;
        }
}
```

11. Write a class definition, including instance variables, getters and setters, default and parameterized constructors, totalStockPrice(), and compareTo method to define a StockItem, as represented by its unit price and quantity left. Total stock price is defined as the product of unit price and quantity left.

Solution:

```
public class StockItem {
        private double price;
        private int quantity;

        public double getPrice() {
                return price;
        }

        public void setPrice(double price) {
                this.price = Math.abs(price);
        }

        public int getQuantity() {
                return quantity;
        }
        public void setQuantity(int quantity) {
                this.quantity = Math.abs(quantity);
        }

        public StockItem(String n, double p, int q) {
                setName(n);
                setPrice(p);
                setQuantity(q);
        }
```

```
        public String toString() {
                return name+" priced at "+price+", quantity left "+quantity;
        }

        public double totalStockPrice() {
                // TODO Auto-generated method stub
                return price * quantity;
        }

        public int compareTo(StockItem other) {
                double total1 = this.totalStockPrice();
                double total2 = other.totalStockPrice();

                if(total1 > total2)
                        return 1;
                if(total1 < total2)
                        return -1;
                return 0;
        }
}
```

12. Consider the following class definition,

```
class Box {
   private double width, breadth, height;

   //assume getters, setters
   public Box(double w, double b, double h) {
          setWidth(w);
          setBreadth(b);
          setHeight(h);
   }

   public double volume() {
          return width * breadth * height;
   }

   public int compareTo(Box other) {
          if(volume() > other.volume())
                 return 1;
          if(volume() < other.volume())
             return -1;
          return 0;
   }
}
```

Outside the class, declare and instantiate an object `myBox` of class `Box` with width = 6.4, breadth = 9.6, and height = 1.5.

Solution:

```
Box myBox = new Box(6.4, 9.6, 1.5);
```

Assume a second `Box` object `yourBox`. Store the outcome of comparing `myBox` (as the calling object) with `yourBox` (as the parameter object) in a variable `status`.

Solution:

```
int status = myBox.compareTo(yourBox);
```

Assume the following code outside the class `Box`

```
Box[] boxes = new Box[100];
for(int i=0; i < boxes.length; i++) {
        double w = nextInt(10);
        double b = nextInt(10);
```

```
            double h = nextInt(10);
            boxes[i] = new Box(w, b, h);
        }
```

Add some code after the given code that displays the breadth of boxes with width more than 7.

Solution:

```
for(int i=0; i < boxes.length; i++) {
        if(boxes[i].getWidth() > 7) {
                System.out.println(boxes[i].getBreadth());
        }
}
```

Or (using the built-in iterator):

```
for(Box b: boxes) {
        if(b.getWidth() > 7) {
                System.out.println(b.getBreadth());
        }
}
```

13. Consider the following class `Service`.

```
class Service {
        private int n;

        //assume setters, getters, constructors

        public int doubleUp() {
                setN(2*n); //setter is assumed to be there
        }

        public static int multiply(int a, int b) {
                return a * b;
        }
}
```

Further assume an object `myService` of class `Service` for which $n = 5$.

What's the correct way to call the method `doubleUp` - `myService.doubleUp()`, or `Service.doubleUp()`?

Solution: `myService.doubleUp()`

What about `multiply` - `myService.multiply(2, 4)`, or `Service.multiply(2, 4)`?

Solution: `Service.multiply(2, 4)`

3 Searching and Sorting

14. Write a method that when passed an integer array, sorts it in ascending order using insertion sort.

15. Write a method that when passed an integer array, sorts it in ascending order using selection sort.

16. Write a method that when passed an integer array, assumed to be sorted in ascending order, and an integer `key`, returns the index at which `key` is found, using binary search.

17. For which algorithm, is the central principle, "for each item of the array, move the current item into its rightful place, *so far*"

    Solution: insertion sort

18. For which algorithm, is the central principle, "for each item of the array, swap it with the smallest item in the remaining array"

    Solution: selection sort

19. For which algorithm, is the central principle (loosely speaking), "split the array in half and the target is either at that index, or if there, it must be to the left (if smaller than median), or to the right (if higher than median)"

    Solution: binary search

4 Recursion

20. What are the values of `result1, result2, result3` when the following code is executed? What does that tell you about the method `foo`?

```
public class Client {

        public static int foo(int n) {
                if(n <= 1)
                        return 0;
                return 1 + foo(n/2);
        }

        public static void main(String[] args) {
                int result1 = foo(32);
                int result2 = foo(64);
                int result3 = foo(4);
        }
}
```

Solution:

```
result1:

foo(32) = 1 + foo(16)
foo(16) = 1 + foo(8)
foo(8) = 1 + foo(4)
foo(4) = 1 + foo(2)
foo(2) = 1 + foo(1)
foo(1) = 0
Passing back,
foo(2) = 1 + 0 = 1
foo(4) = 1 + 1 = 2
foo(8) = 1 + 2 = 3
foo(16) = 1 + 3 = 4
foo(32) = 1 + 4 = 5

result1 = 5

Similarly,

result2 = 6
result3 = 2

It tells you that foo is a log (base 2) computing function.
```

21. What is the value of `result` when the following code is executed? Are there any values for the parameter to `bar` when the method will executed indefinitely causing `StackOverflowError`?

```
public class Client {

        public static int bar(int n) {
                if(n == 1)
                        return 0;
                return n + bar(n - 3);
        }

        public static void main(String[] args) {
                int result = bar(10);
        }
}
```

Solution: bar(10) = 10 + bar(7) = 10 + 7 + bar(4) = 10 + 7 + 4 + bar(1) = 10 + 7 + 4 + 0 = 21

yes, it will execute forever for any n for which remainder by 3 is NOT 1.

22. Write a **recursive** method that when passed an integer (assume it is more than 0), returns the number of digits in the integer. Hint: you get the last digit of an integer n by $n\%10$ and the rest of the number by $n/10$.

23. Write a **recursive** method that when passed an integer (assume it is more than 0), returns the sum of digits in the integer.

24. Write a **recursive** method that when passed an integer (assume it is more than 0), returns the product of digits in the integer.

25. Write a **recursive** method that when passed an integer (assume it is more than 0), returns the highest digit in the integer.

26. Write a **recursive** method that when passed two integers (assume both are more than 0), returns their *greatest common divisor* (gcd). Euclid's algorithm states that $gcd(a, b) = gcd(b, a\%b)$ if $b \neq 0$ and $gcd(a, 0) = a$.

    Thus,

```
gcd(80, 56)
=gcd(56, 24)
=gcd(24, 8)
=gcd(8, 0)
=8
```

27. Write a **recursive** method that when passed a String (assume not `null`), returns the reverse of the String.

28. Write a **recursive** method that when passed a String (assume not `null`), returns `true` if it's a palindrome (exactly the same when reversed), and `false` if it's not a palindrome. Any String with 0 or 1 characters (like "" or "A") is a palindrome)

5 Lists

29. What are the contents of `list` when the following code is executed?

```
ArrayList<Integer> list = new ArrayList<Integer>();

for(int i=0; i < 10; i++) {
    if(i % 2 == 0) {
        list.add(2*i + 1);
    }
    else {
        list.add(2*i - 1);
    }
}

list.remove(4);

list.set(7, list.get(7) * 10);

for(int i=0; i < list.size(); i++) {
    if(i % 2 == 0) {
        list.set(i, list.get(i) + 1);
    }
}
```

30. Complete the method `countOdds`, that when passed an `ArrayList` of integer objects assumed to be not `null`), returns the number of items that are odd (not divisible by 2).

```
public static int countOdds(ArrayList<Integer> list)
```

31. Write a method that returns `true` an `ArrayList` of doubles is sorted in ascending order, `false` otherwise

32. Write a method that removes all negative items from an `ArrayList` of doubles