Objects and Classes

1. Complete statements (a) and (b), and answer questions (c) and (d):
   
   (a) An object is an instance of a
       Answer: class 2 points
   
   (b) The collection of an object’s instance variables is known as its
       Answer: state 2 points
   
   (c) In Java, objects are created with what operator?
       Answer: new 2 points
   
   (d) In Java, a class may have instance variables and methods. It may also have named blocks of code, at most one of which is executed when an object is created. What are these named blocks of code called?
       Answer: constructors 2 points

   **Primitive Variables: Declaration, Assignment, Operations**

2. For each of the following program fragments, what is the value of the variable x after the statements execute?

   (a) `int y = 10;
       int x = y;
       y = y * 3;`
       Answer: 10 2 points

   (b) `int x = 10;
       x = x / 2;
       x = x + 3 * x - 3;`
       Answer: 17 2 points

   (c) `double x = 7 / 2;`
       Answer: 3.0 2 points

   (d) `boolean x = ( 2 < 3 );`
       Answer: true 2 points

   **Logical Expressions**

3. What is the value of each of the following valid expressions?

   (a) `(2 <= 2) && (2 >= 2)`
       Answer: true 1 points

   (b) `(!true) || (!false)`
       Answer: true 2 points

   (c) `false == false`
       Answer: true 2 points
Constructors

4. Given the following DrJava interactions, write code for the Gadget class that works correctly and uses standard coding style conventions. Each constructor should initialize all instance variables.

```java
> Gadget g1 = new Gadget();
> g1.getNumber()
0
> g1.getStatus()
false
>
> Gadget g4 = new Gadget(5, true);
> g4.getNumber()
5
> g4.getStatus()
true

Answer:

```java
public class Gadget{
    private int number;
    private boolean status;

    public Gadget(){
        number = 0;
        status = false;
    }

    public Gadget(int num, boolean status){
        number = num;
        this.status = status;
    }

    public int getNumber() { return number; }
    public boolean getStatus() { return status; }
}
```
Conditionals (if statements)

5. Here we have a Student class which is used to create Students who can take a midterm. Complete the `takeMidterm` method which updates the state of Student’s score and behaves according to the rules listed below. Assume `takeMidterm` is called just called once per object (the method can assume that the student has not been given a score yet).

   (a) A student who is a genius will always get 100 on any midterm.
   (b) A student who studies for a midterm will get 100 if the exam’s difficulty rating is less than or equal to 7, otherwise they will get an 80.
   (c) A student who does not study will get a 50 on an exam with a difficulty rating less than 5, otherwise they will get a 0.

```java
public class Student{
    private int score;
    private boolean genius;
    private String name;

    public Student(String name, boolean genius){
        this.name = name;
        this.genius = genius;
        score = -99; // exam not taken yet
    }

    public int getScore() { return score; }

    public void takeMidterm(int difficulty, boolean studied){
        if (genius){
            score = 100;
            return;
        }
        // student is not a genius
        if (studied){
            if (difficulty <= 7)
                score = 100;
            else
                score = 80;
            return;
        }
        // student did not study
        if (difficulty < 5)
            score = 50;
        else
            score = 0;
        return;
    }
}
```
6. Each code fragment below compiles without error. However each has one or more problems. Fix the code so that it works according to the supplied documentation (in comments) or DrJava interactions.

(a) public class Holder {
    private int value;
    public Holder(int initialValue) {
        initialValue = value; // value = initialValue;
    }
    public int getValue() {
        return value;
    }
    public void addToValue(int toAdd) {
        int value += toAdd; // value += toAdd;
    }
}

Interactions with a correct implementation of Holder:
> Holder n = new Holder(4);
> n.getValue()
4
> n.addToValue(5);
> n.getValue()
9

(b) /* If data has a non-zero length, prints each element of data, * one per line. Otherwise, prints nothing.
*/
public void test1(int[] data){
    // if (data == null return;)
    for(int i = 0; i <= data.length; i++){ // i < data.length
        System.out.println(data[i]);
    }
}

(c) /* Returns sum of all of the even numbers from 0 to limit inclusive.
Assumes limit is positive.
*/
public int addEvens(int limit){
    int i = 0;
    int sum = 0;
    while(i != limit){ // while (i <= limit)
        sum += i;
        i+=2;
    }
    return sum;
}
A TangoDancer has a name and may have a partner who is also a TangoDancer. If TangoDancer A chooses TangoDancer B to be a partner, the operation succeeds if neither A nor B already has a partner, and A is not B. On the next page, write the `choosePartner` method and additional methods as desired so that the following DrJava interactions work:

```java
> TangoDancer rita = new TangoDancer("senorita rita");
> TangoDancer tony = new TangoDancer("mr tony");
> rita.getName()
"senorita rita"
> rita.hasAPartner()
false
> tony.hasAPartner()
false
> rita.choosePartner(tony)
true
> rita.hasAPartner()
true
> rita.getPartner().getName()
"mr tony"
> tony.hasAPartner()
true
> tony.getPartner().getName()
"senorita rita"
> tony.choosePartner(rita)
false
> TangoDancer elana = new TangoDancer("ms elana");
> elana.choosePartner(elana)
false
```
public class TangoDancer{
    private String name;
    private TangoDancer partner;

    public TangoDancer(String name){
        this.name = name;
        partner = null;
    }

    public String getName() { return name; }
    public TangoDancer getPartner() { return partner; }
    public boolean hasAPartner() { return partner != null; }

    public boolean choosePartner(TangoDancer other){
        if (other == this)
            return false;
        if (partner != null)
            return false;
        if (other.hasAPartner())
            return false;
        partner = other;
        other.setPartner(this);
        return true;
    }

    void setPartner(TangoDancer chooser){
        partner = chooser;
    }
}
Loops

8. For each code segment below, determine how many times the body of the loop is executed. Write one of the following answers after each: 0, 1, infinite, or > 1. Note that ”> 1” means more than once but not infinite.

(a) for(int x=1; x<10; x++){  
    System.out.println(x);  
}  
Answer: > 1

(b) int x=1;  
    while(x<10){  
        System.out.println(x);  
    }  
Answer: infinite

(c) int x=1;  
    do{  
        x = x*2;  
    } while(x>=8);  
Answer: 1

(d) int x=10;  
    while(x<10){  
        System.out.println(x);  
        x=x-1;  
    }  
Answer: 0

(e) int x=1;  
    while(x!=10){  
        x = x*2;  
    }  
Answer: infinite
Arrays

9. (a) Given the following array declaration:

```java
double [] a = new double[5];
```

What is the type of each of the following expressions?:

i. `a.length`
   - Answer: `int`

ii. `a[0]`
   - Answer: `double`

iii. `a`
   - Answer: `double[]`

What is the value of each of the following expressions?

i. `a.length`
   - Answer: `5`

ii. `a[3]`
   - Answer: `0`

iii. `new double[5]`
   - Answer: *the address where the array of doubles is stored*

(b) Complete the method `min` that takes an array of integers as input and returns the smallest item in the array. Assume that the array supplied as an argument has at least one element. Example:

```java
> ArrayTool tool = new ArrayTool();
> tool.min(new int[] { 20, 40, 60, -80, 100})
-80
```

```java
public class ArrayTool{

    public int min(int[] data){
        int smallest = data[0];
        for (int i = 1; i < data.length; i++)
            if (data[i] < smallest){
                smallest = data[i];
            }
        return smallest;
    }
}
```
(c) Complete the method `sum` that takes two arrays of integers as arguments. It returns null if either argument is null or if the arrays are not the same length. Otherwise it returns a new array, each of whose elements equals the sum of the corresponding elements in the input arrays. The input arrays should not be changed by the method. For example:

Answer:

```java
public int[] sum(int[] one, int[] two){
    if (one == null || two == null)
        return null;
    if (one.length != two.length)
        return null;
    int[] result = new int[one.length];
    for (int i = 0; i < one.length; i++)
        result[i] = one[i] + two[i];
    return result;
}
```

Answer:

```java
> ArrayTool tool = new ArrayTool();
> int[] first = new int[]{1, 2, 3, 4};
> int[] second = new int[]{10, 20, 30, 40};
> int[] result = tool.sum(first, second);
> for (int i = 0; i < result.length; i++)
    System.out.print(result[i] + " ");
11 22 33 44
> for (int i = 0; i < first.length; i++)
    System.out.print(first[i] + " ");
1 2 3 4
> for (int i = 0; i < second.length; i++)
    System.out.print(second[i] + " ");
10 20 30 40
```