Problem Set 3, Part I …

Please start your answer to each problem on a new page, as we have done below!

Problem 1: A class that needs your help

1-1) *Revise the code found below:*

```
public class ValuePair {
    private int a;
    private double b;

    public double product() {
        return this.a * this.b;
    }

    // add the new methods here
    public ValuePair(int a, double b) {
        modifyA(a);
        modifyB(b);
    }

    public int getOddInt() {
        return a;
    }

    public double getRealNum() {
        return b;
    }

    public void modifyA(int a) {
        if (a % 2 == 0) {
            throw new IllegalArgumentException();
        } else {
            this.a = a;
        }
    }

    public void modifyB(double b) {
        if (b > 0.0) {
            this.b = b;
        } else {
            throw new IllegalArgumentException();
        }
    }
}
```

1-2)

```
a)    ValuePair vp1 = new ValuePair(7, 15.5);
b)    vp1.a = 25;
c)    double b1 = vp1.getRealNum();
d)    vp2.modifyA(vp2.getOddInt() + 2);
```

Problem 2: Static vs. non-static

2-1)

type and name of the variable	**static or non-static?**	**purpose of the variable, and why it needs to be static or non-static**
double rawScore	non-static	Stores the raw score associated with a given Grade object; needs to be non-static so every Grade object will have its own instance of this variable
int latePenalty	non-static	Stores the late penalty as a percentage associated with a given Grade object; needs to be non-static so every Grade object will have its own instance of this variable
String category	non-static	Stores the category of the grade associated with a given Grade object; needs to be non-static so every Grade object will have its own instance of this variable
int assignmentCount	static	Stores the number of Grade objects created with category “assignment”. This is static so that it is shared across all instances
int quizCount	static	Stores the number of Grade objects created with category “quiz”. This is static so that it is shared across all instances
int examCount	static	Stores the number of Grade objects created with category “exam”. This is static so that it is shared across all instances
double adjustedScore	non-static	Stores the adjusted score associated with a given Grade object; needs to be non-static so every Grade object will have its own instance of this variable

2-2)

a) static or non-static?: non-static
explanation: it modifies the instance-specific **category** field and adjusts the class-wide counts for categories

b) changes it would need to make:
Decrement the **quizCount** static variable
Increment the **examCount** static variable
Update the **category** field of the specific Grade object

2-3)

a) static or non-static?: non-static
explanation: it modifies the instance-specific **latePenalty** field

b) example of calling it:
```
g.waiveLatePenalty();
```

2-4)

a) static or non-static?: static
explanation: it performs a calculation based solely on its parameters and does not depend on any instance-specific fields

b) example of calling it:
```
double rawScore = Grade.computeRaw(pct, possiblePoints);
```

Problem 3: Understanding and using inheritance

3-1) Truck, Vehicle

3-2) Yes, because TractorTrailer inherits from Truck, which in turn inherits from Vehicle.

3-3) *Add your definition of the Limousine class below:*

```
public class Limousine extends Automobile {
	private boolean hasSunRoof;
	private int champagneBottleCapacity;

	public Limousine(String make, String model, int year, int totalSeats, boolean
hasSunRoof, int champagneBottleCapacity) {
		super(make, model, year, totalSeats);
		if  (champagneBottleCapacity < 0) {
			throw new IllegalArgumentException(“Champagne bottle capacity
cannot be negative”);
		}
		this.hasSunRoof = hasSunRoof;
		this.champagneBottleCapacity = champagneBottleCapacity;
	}

	public boolean hasSunRoof() {
		Return hasSunRoof;
	}

	public int getChampagneBottleCapacity() {
		return champagneBottleCapacity;
	}

	public String toString() {
		Return getMake() + “ ” + getModel() + “ “ + “(seats up to “ +
(getTotalSeats() - 2) + “ customers)”;
	}
}
```

Problem 4: Inheritance and polymorphism

4-1) The equals() method that Zoo overrides comes from the Object class in Java which is implicitly inherited by every class. The Object class provides a default implementation of the equals() method which Zoo overrides.

4-2) String c, int x, int y, int a, int b

4-3)

which println statement?	which method is called?	will the call compile (yes/no?)	if the call compiles, which version of the method will be called?
first one	one()	yes	the Yoo version
second one	two()	yes	The Woo version
third one	three()	no	-
fourth one	equals()	yes	The Zoo version
fifth one	toString()	yes	The Woo version

4-4)

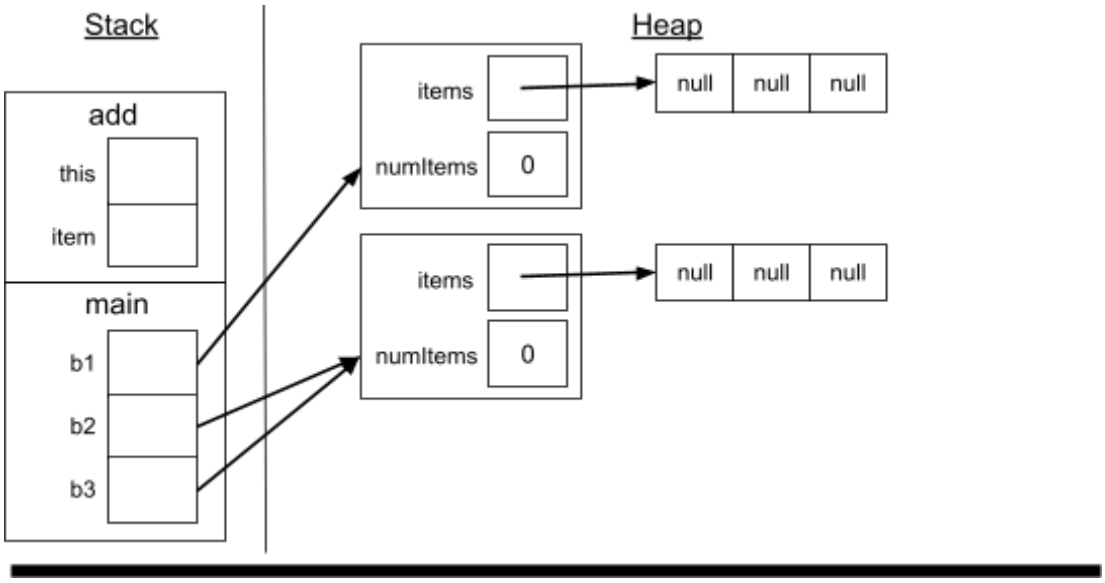
```
class Too extends Zoo {
	private int t;
	private int u;

	public double avg() {
		return (double) (getA() + getT() + getU()) / 3;
	}

	public int getT() {
		return t;
	}

	public int getU() {
		return u;
	}
}
```

4-5)

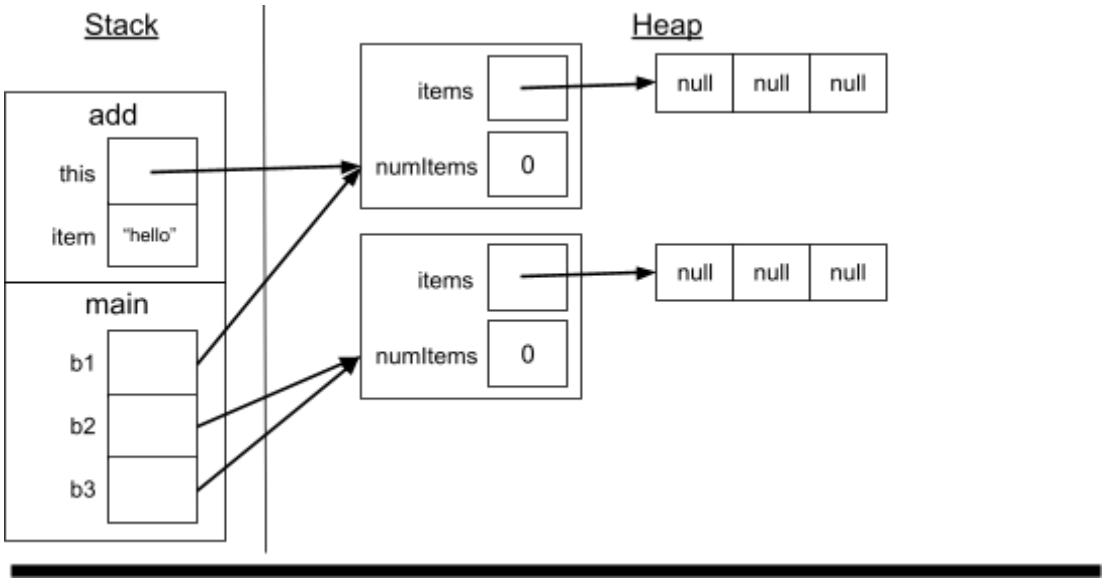
a) Allowed because Too is a subclass of Woo
b) Allowed because Woo is a subclass of Zoo
c) Allowed because Yoo is a subclass of Zoo
d) Not allowed because Zoo is not a subclass of Too

Problem 5: Memory management and the `ArrayBag` class

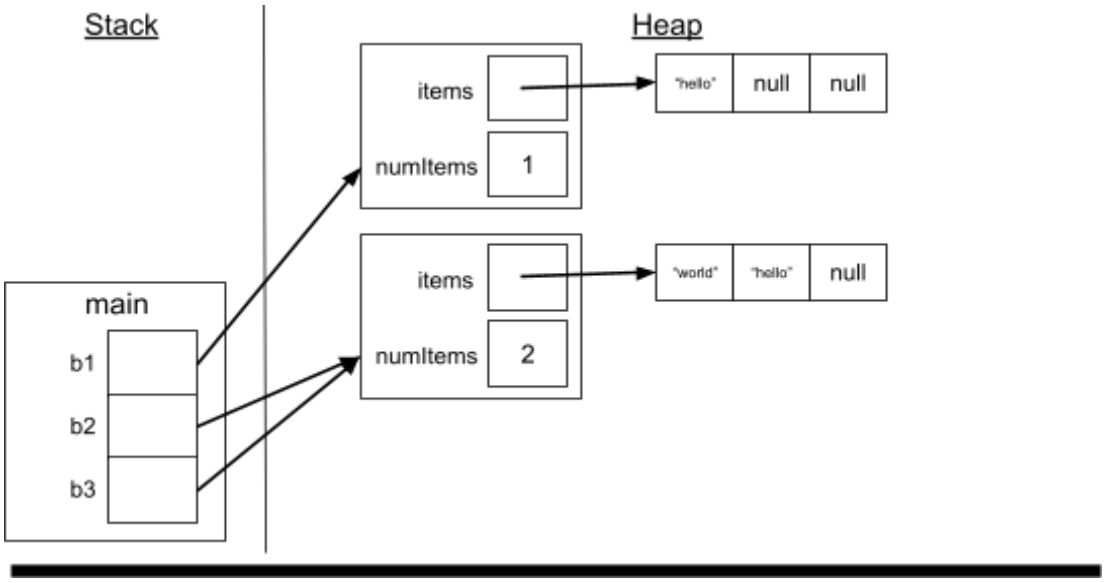
5-1)



5-2)



5-3)



5-4)

"{hello}"
"{world, hello}"
"{world, hello}"

Problem 6: Using recursion to print an array

6-1)

```
public static void print(int[] arr, int start) {
	if (start < 0) {
		throw new IllegalArgumentException();
	}

	if (arr.length == 0||arr == null||arr.length <= start) {
		return;
	} else {
		System.out.println(arr[start]);
		print(arr, start + 1);
	}
}
```

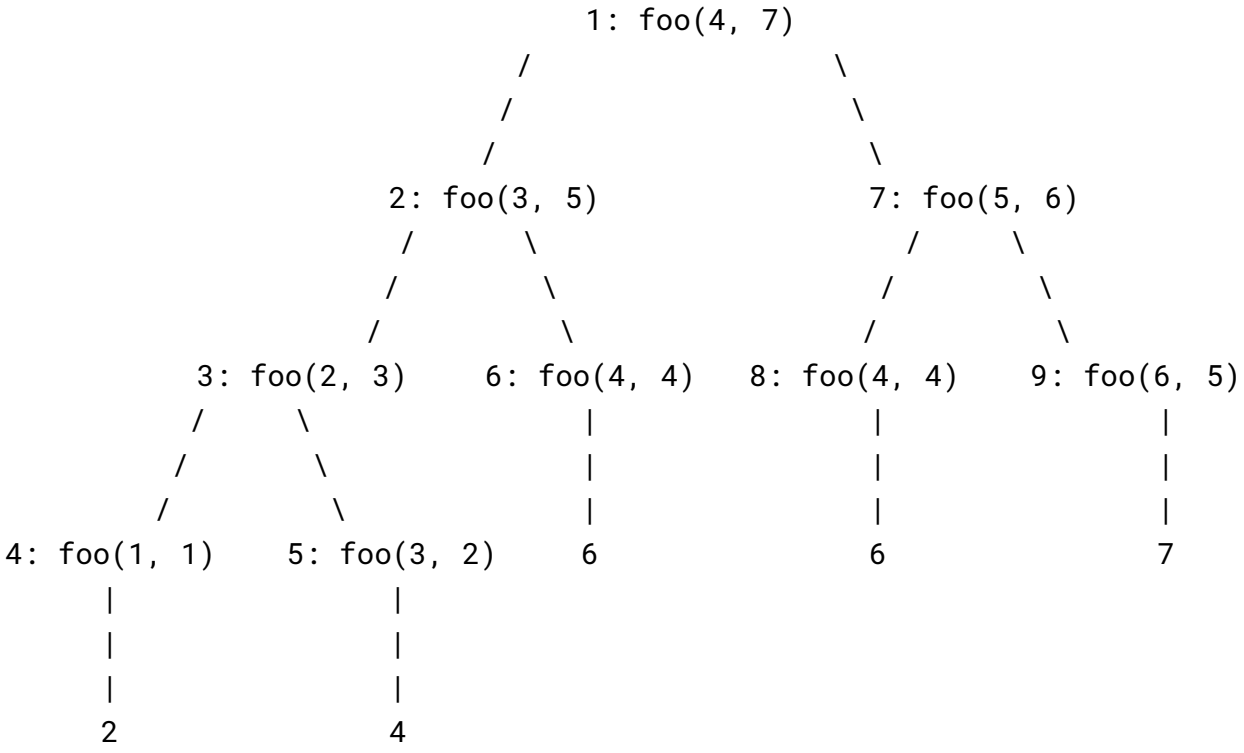
6-2)

```
public static void printReverse(int[] arr, int i) {
	if (i < 0) {
		throw new IllegalArgumentException();
	}

	if (arr.length != 0 && arr != null && arr.length < i) {
		printReverse(arr, i + 1);
		System.out.println(arr[i]);
	}
}
```

6-3) ***initial call:*** `printReverse(arr, 0)`

Problem 7: A method that makes multiple recursive calls

7-1)



7-2)

call 1 (foo(4,7)) returns (foo(3,5) + foo(5,6)) (**return value: 25)**
call 2 (foo(3,5)) returns (foo(2,3) + foo(4,4)) **(return value: 12)**
call 3 (foo(2,3)) returns (foo(1,1) + foo(3,2)) **(return value: 6)**
call 4 (foo(1,1)) returns 2
call 5 (foo(3,2)) returns 4
call 6 (foo(4,4)) returns 6
call 7 (foo(5,6)) returns (foo(4,4) + foo(6,5)) **(return value: 13)**
call 8 (foo(4,4)) returns 6
call 9 (foo(6,5)) returns 7