Final Exam
EECS2030
Advanced Object-Oriented Programming
Fall 2021
Professor: Dr. Marzieh Ahmadzadeh

Student Name: ________________________ **Student ID: ______________________**

Section: _______ **Grade: __________ /100**

Please read the following information before you start writing.

Please do not flip this page until you are signaled to do so.
You have 180 minutes to complete the exam.
Raise your hand if you have any question.
No question will be answered in the last 15 minutes of the exam.
This exam has 100 points that accounts for 20% of your total grade.
Write the answers neatly. If your answer is not readable, no mark will be awarded.
The last page of this exam is blank and therefore you can use it as your draft work.
This exam is a closed book exam therefore NO aid including textbook, handout etc. are allowed.

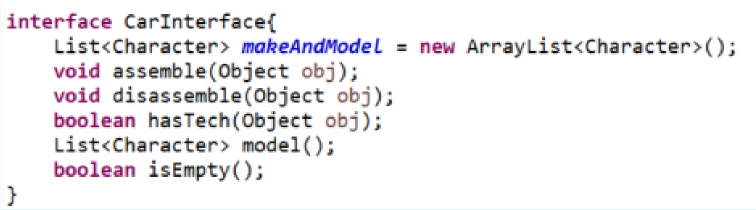
The York University guideline on academic honesty prohibits all forms of academic dishonesty including cheating, and the use of unauthorized aids. As a result, it is expected that you do not consult any unauthorized source including, not limited to, colleagues, handout, mobile phone, and other forms of electronic resources during the examination. Students violating the Code may be subject to penalties up to and including suspension or expulsion from the University.
If you get to this point, it means you have read all the instructions above. Draw a smiley face on top right corner of this page and receive one free point.

GOOD LUCK ON YOUR EXAM

Use this page for your rough notes. Please note that this page will not be marked.

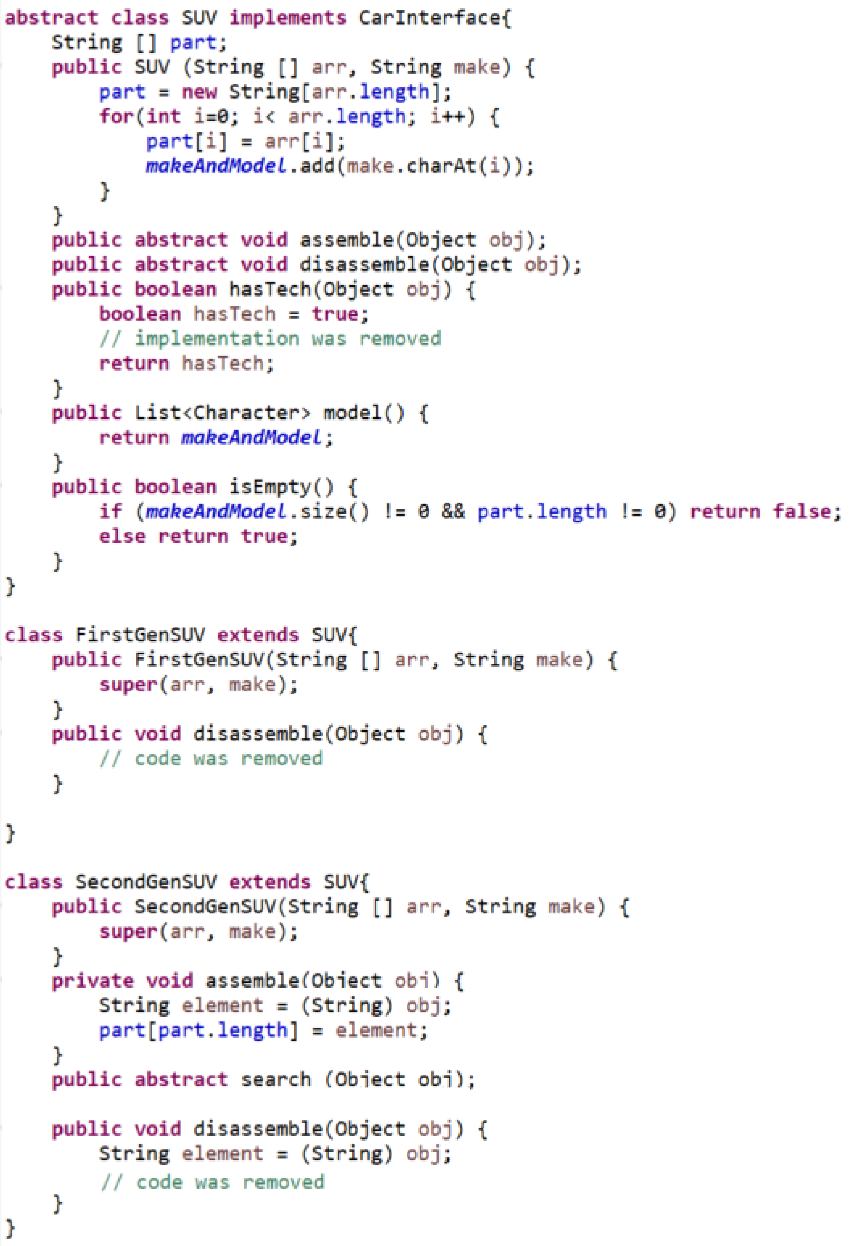
Q1:

An interface called *carInterface* is designed as below:



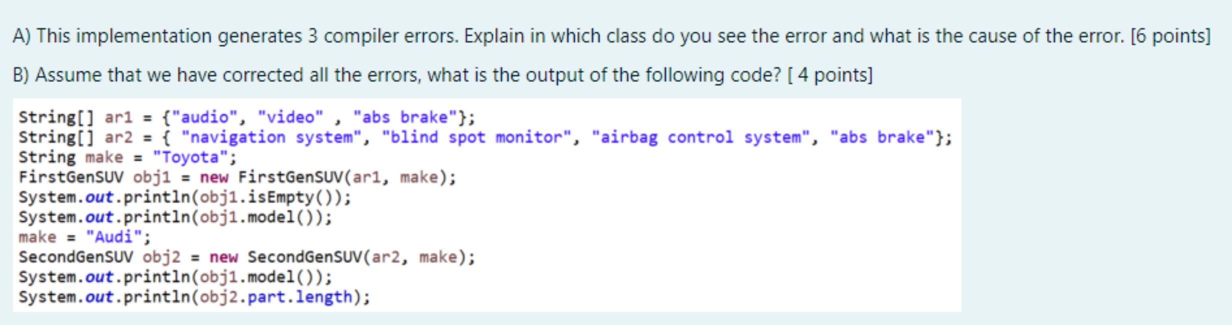
```
interface CarInterface{
    List<Character> makeAndModel = new ArrayList<Character>();
    void assemble(Object obj);
    void disassemble(Object obj);
    boolean hasTech(Object obj);
    List<Character> model();
    boolean isEmpty();
}
```

Using this interface, the following hierarchy of inheritance is created.



```
abstract class SUV implements CarInterface{
    String [] part;
    public SUV (String [] arr, String make) {
        part = new String[arr.length];
        for(int i=0; i< arr.length; i++) {
            part[i] = arr[i];
            makeAndModel.add(make.charAt(i));
        }
    }
    public abstract void assemble(Object obj);
    public abstract void disassemble(Object obj);
    public boolean hasTech(Object obj) {
        boolean hasTech = true;
        // implementation was removed
        return hasTech;
    }
    public List<Character> model() {
        return makeAndModel;
    }
    public boolean isEmpty() {
        if (makeAndModel.size() != 0 && part.length != 0) return false;
        else return true;
    }
}

class FirstGenSUV extends SUV{
    public FirstGenSUV(String [] arr, String make) {
        super(arr, make);
    }
    public void disassemble(Object obj) {
        // code was removed
    }

}

class SecondGenSUV extends SUV{
    public SecondGenSUV(String [] arr, String make) {
        super(arr, make);
    }
    private void assemble(Object obj) {
        String element = (String) obj;
        part[part.length] = element;
    }
    public abstract search (Object obj);

    public void disassemble(Object obj) {
        String element = (String) obj;
        // code was removed
    }
}
```

Use this page for your rough notes. Please note that this page will not be marked.



A) This implementation generates 3 compiler errors. Explain in which class do you see the error and what is the cause of the error. [6 points]

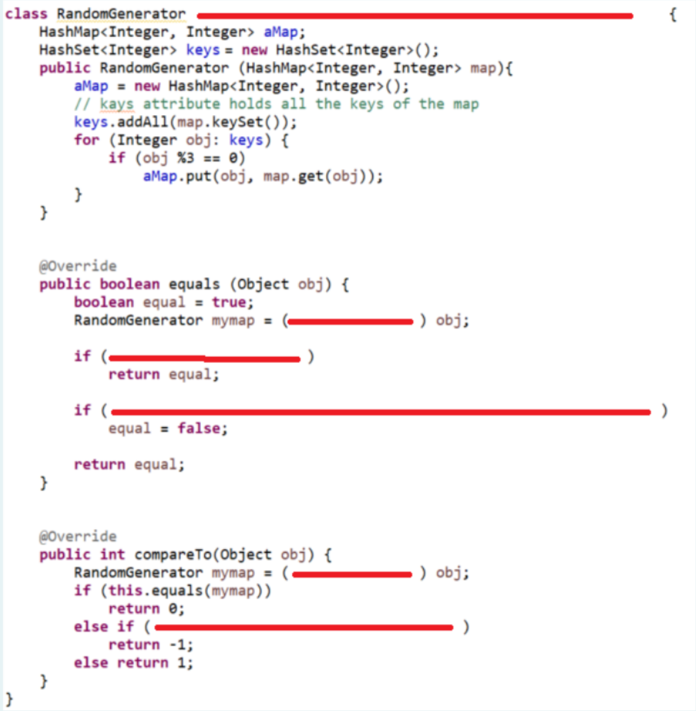
B) Assume that we have corrected all the errors, what is the output of the following code? [4 points]

```
String[] ar1 = {"audio", "video" , "abs brake"};
String[] ar2 = { "navigation system", "blind spot monitor", "airbag control system", "abs brake"};
String make = "Toyota";
FirstGenSUV obj1 = new FirstGenSUV(ar1, make);
System.out.println(obj1.isEmpty());
System.out.println(obj1.model());
make = "Audi";
SecondGenSUV obj2 = new SecondGenSUV(ar2, make);
System.out.println(obj1.model());
System.out.println(obj2.part.length);
```

Use this page for your rough notes. Please note that this page will not be marked.

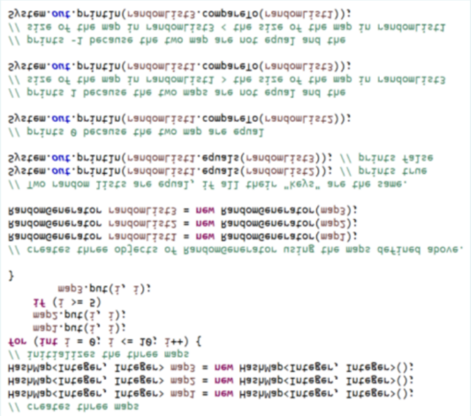
Q2:

RandomGenerator is a HashMap that extracts the pairs of <key, value> whose keys are divisible by 3 from a given map, and stores them in its instance variable called aMap. The definition of this class is as follow:



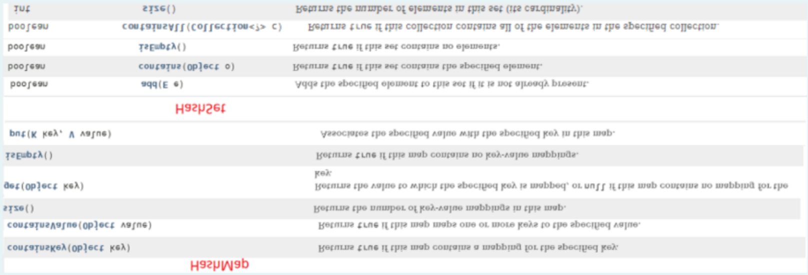
```
class RandomGenerator ______________________________        {
    HashMap<Integer, Integer> aMap;
    HashSet<Integer> keys = new HashSet<Integer>();
    public RandomGenerator (HashMap<Integer, Integer> map){
        aMap = new HashMap<Integer, Integer>();
        // kays attribute holds all the keys of the map
        keys.addAll(map.keySet());
        for (Integer obj: keys) {
            if (obj %3 == 0)
                aMap.put(obj, map.get(obj));
        }
    }

    @Override
    public boolean equals (Object obj) {
        boolean equal = true;
        RandomGenerator mymap = (__________) obj;

        if (______________ )
            return equal;

        if (__________________________________________ )
            equal = false;

        return equal;
    }

    @Override
    public int compareTo(Object obj) {
        RandomGenerator mymap = (__________) obj;
        if (this.equals(mymap))
            return 0;
        else if (______________________ )
            return -1;
        else return 1;
    }
}
```

The following code is a correct code that is executed without any problem. The comments show what each method does and what is outputted. Fill in the gaps in the code above, so that the following code runs as expected.



```
\\ creates three maps
HashMap<Integer, Integer> map1 = new HashMap<Integer, Integer>();
HashMap<Integer, Integer> map2 = new HashMap<Integer, Integer>();
HashMap<Integer, Integer> map3 = new HashMap<Integer, Integer>();
\\ initializes the three maps
for (int i = 0; i <= 10; i++) {
    map1.put(i, i);
    map2.put(i, i);
    if (i >= 5)
        map3.put(i, i);
}

\\ creates three objects of RandomGenerator using the maps defined above.
RandomGenerator randomList1 = new RandomGenerator(map1);
RandomGenerator randomList2 = new RandomGenerator(map2);
RandomGenerator randomList3 = new RandomGenerator(map3);

\\ Two random lists are equal, if all their "keys" are the same.
System.out.println(randomList1.equals(randomList2)); \\ prints true
System.out.println(randomList1.equals(randomList3)); \\ prints false

\\ prints 0 because the two map are equal
System.out.println(randomList1.compareTo(randomList2));

\\ prints 1 because the two maps are not equal and the
\\ size of the map in randomList1 > the size of the map in randomList3
System.out.println(randomList1.compareTo(randomList3));

\\ prints -1 because the two map are not equal and the
\\ size of the map in randomList3 < the size of the map in randomList1
System.out.println(randomList3.compareTo(randomList1));
```

Some of the methods of HashMap and hashSet that you may find useful to understand the code and answer this question is given below.



HashMap

`containsKey(Object key)`	Returns `true` if this map contains a mapping for the specified key.
`containsValue(Object value)`	Returns `true` if this map maps one or more keys to the specified value.
`size()`	Returns the number of key-value mappings in this map.
`get(Object key)`	Returns the value to which the specified key is mapped, or `null` if this map contains no mapping for the key.
`isEmpty()`	Returns `true` if this map contains no key-value mappings.
`put(K key, V value)`	Associates the specified value with the specified key in this map.

HashSet

boolean	`add(E e)`	Adds the specified element to this set if it is not already present.
boolean	`contains(Object o)`	Returns `true` if this set contains the specified element.
boolean	`isEmpty()`	Returns `true` if this set contains no elements.
boolean	`containsAll(Collection<?> c)`	Returns `true` if this collection contains all of the elements in the specified collection.
int	`size()`	Returns the number of elements in this set (its cardinality).

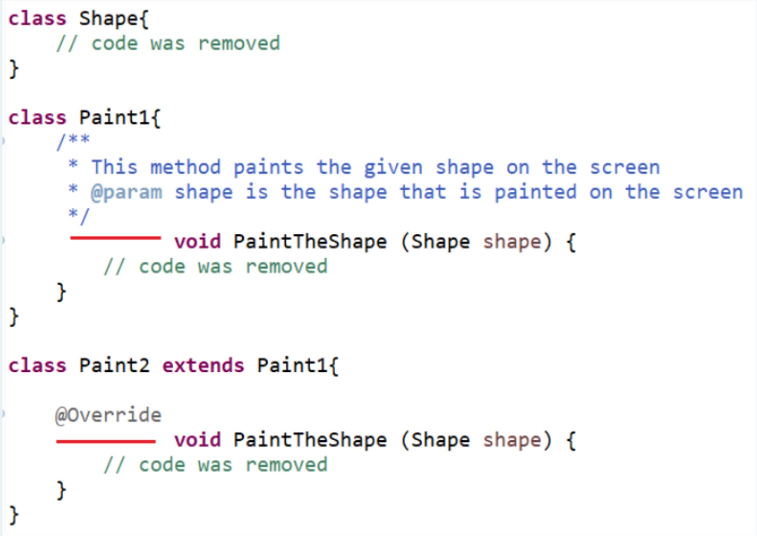
Use this page for your rough notes. Please note that this page will not be marked.

Answer to q2:

Use this page for your rough notes. Please note that this page will not be marked.

Q3:

A set of invariants, preconditions and postconditions are defined for the hierarchy of the classes defined below.



```
class Shape{
    // code was removed
}

class Paint1{
    /**
     * This method paints the given shape on the screen
     * @param shape is the shape that is painted on the screen
     */
     ________ void PaintTheShape (Shape shape) {
        // code was removed
    }
}

class Paint2 extends Paint1{

    @Override
    ________ void PaintTheShape (Shape shape) {
        // code was removed
    }
}
```

A) Specify, which precondition, postcondition and invariant associates with method *paintTheShape* in each class.

Precondition:

- The color of the shape should be red.
- The color of the shape can be anything except red.

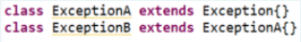
Postcondition:

- draws the shape.
- draws the shape and outline it.

Invariant:

- The content of the page before painting remains as the background of the shape.
- The content of the page before painting remains as the background of the shape. The color of the background remains the same.

B) An exception is thrown by each of *paintTheShape* methods in case the precondition was not met. The exception are as follows.



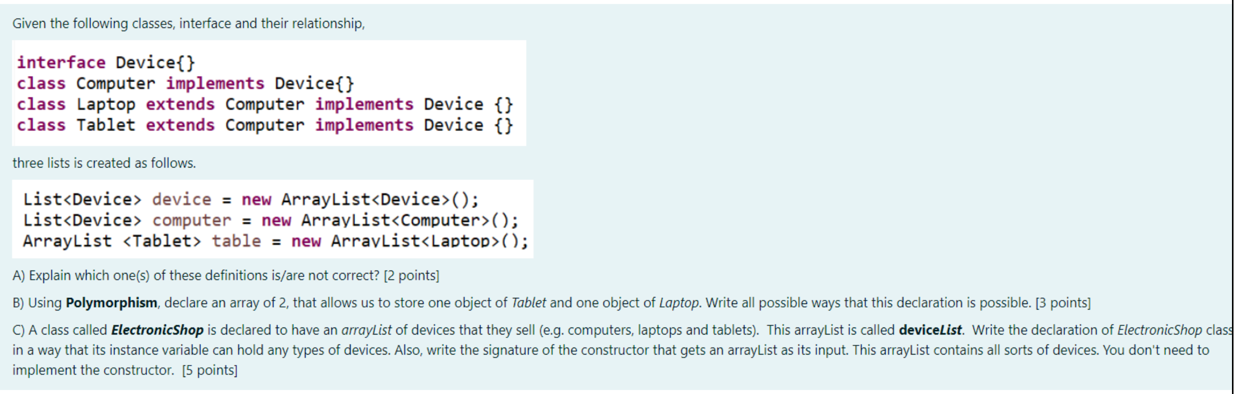
```
class ExceptionA extends Exception{}
class ExceptionB extends ExceptionA{}
```

Specify which exception is thrown by which method.

C) If the access modifier for *paintTheShape* methods can be either protected or public, specify which method gets the protected and which gets the public access modifier.

Use this page for your rough notes. Please note that this page will not be marked.

Q4:



Given the following classes, interface and their relationship,

```
interface Device{}
class Computer implements Device{}
class Laptop extends Computer implements Device {}
class Tablet extends Computer implements Device {}
```

three lists is created as follows.

```
List<Device> device = new ArrayList<Device>();
List<Device> computer = new ArrayList<Computer>();
ArrayList <Tablet> table = new ArrayList<Laptop>();
```

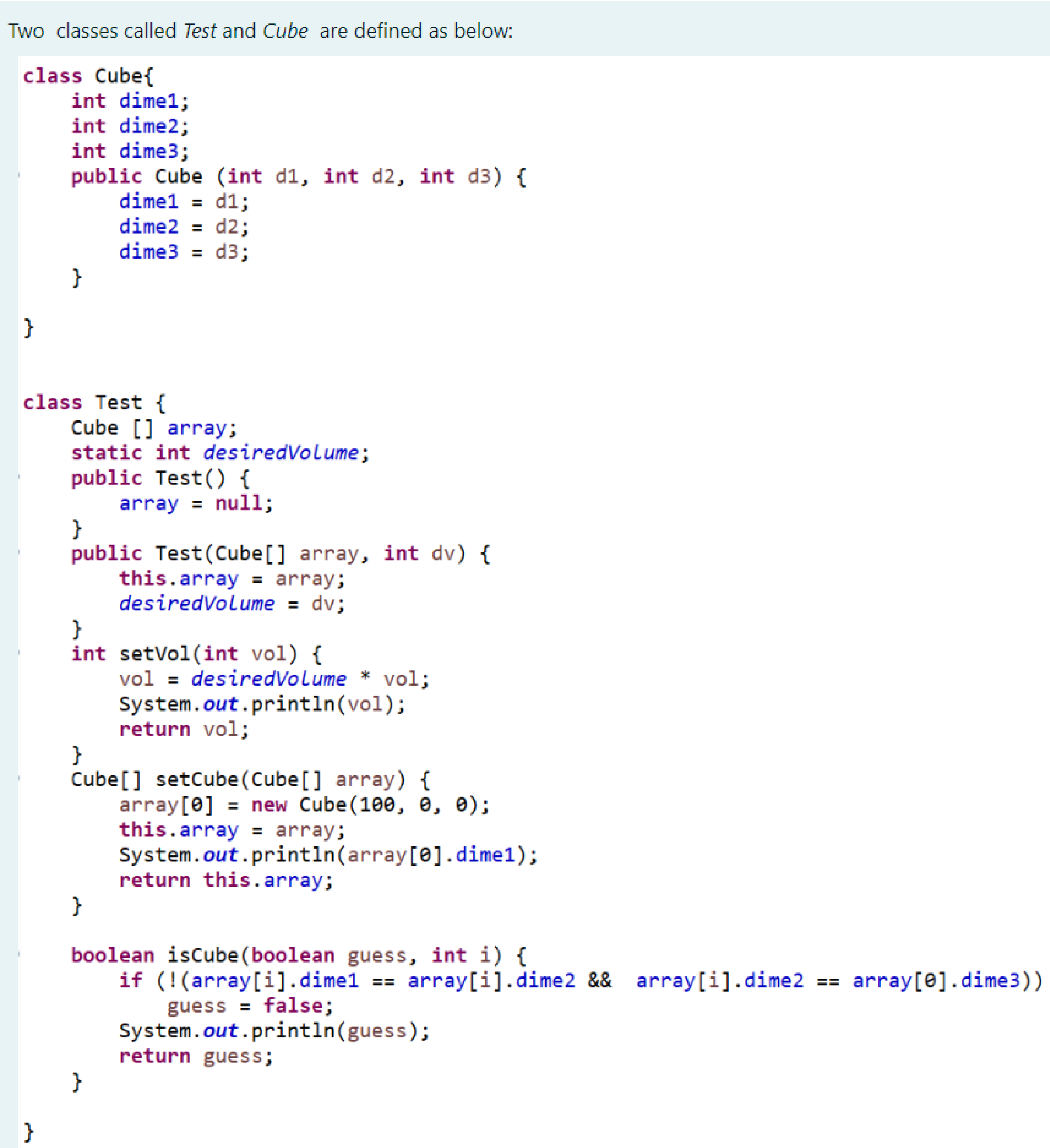
A) Explain which one(s) of these definitions is/are not correct? [2 points]

B) Using **Polymorphism**, declare an array of 2, that allows us to store one object of *Tablet* and one object of *Laptop*. Write all possible ways that this declaration is possible. [3 points]

C) A class called ***ElectronicShop*** is declared to have an *arrayList* of devices that they sell (e.g. computers, laptops and tablets). This arrayList is called **device*List***. Write the declaration of *ElectronicShop* class in a way that its instance variable can hold any types of devices. Also, write the signature of the constructor that gets an arrayList as its input. This arrayList contains all sorts of devices. You don't need to implement the constructor. [5 points]

Use this page for your rough notes. Please note that this page will not be marked.

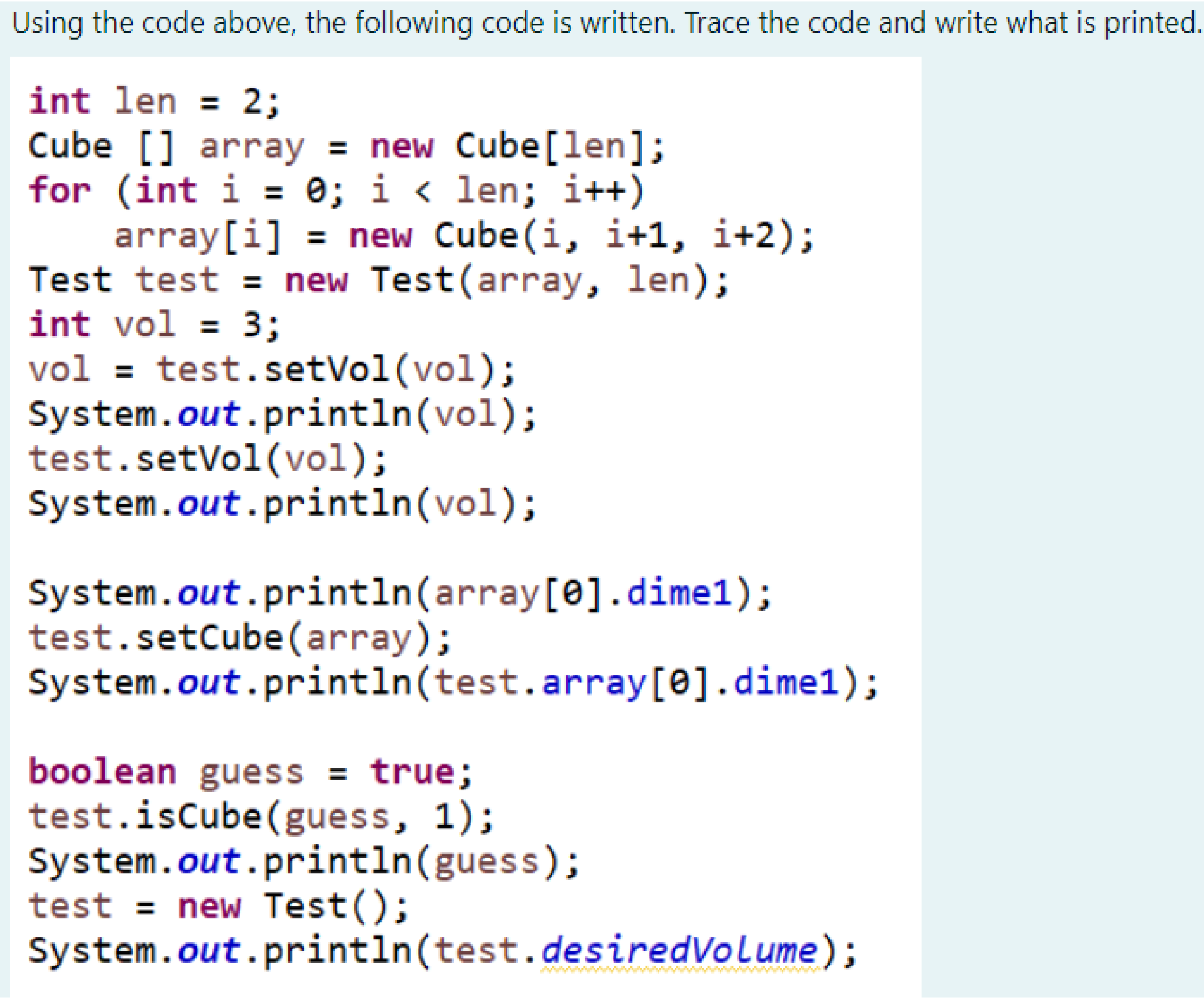
Q5:



Two classes called *Test* and *Cube* are defined as below:

```java
class Cube{
    int dime1;
    int dime2;
    int dime3;
    public Cube (int d1, int d2, int d3) {
        dime1 = d1;
        dime2 = d2;
        dime3 = d3;
    }

}


class Test {
    Cube [] array;
    static int desiredVolume;
    public Test() {
        array = null;
    }
    public Test(Cube[] array, int dv) {
        this.array = array;
        desiredVolume = dv;
    }
    int setVol(int vol) {
        vol = desiredVolume * vol;
        System.out.println(vol);
        return vol;
    }
    Cube[] setCube(Cube[] array) {
        array[0] = new Cube(100, 0, 0);
        this.array = array;
        System.out.println(array[0].dime1);
        return this.array;
    }

    boolean isCube(boolean guess, int i) {
        if (!(array[i].dime1 == array[i].dime2 &&  array[i].dime2 == array[0].dime3))
            guess = false;
        System.out.println(guess);
        return guess;
    }

}
```

Use this page for your rough notes. Please note that this page will not be marked.

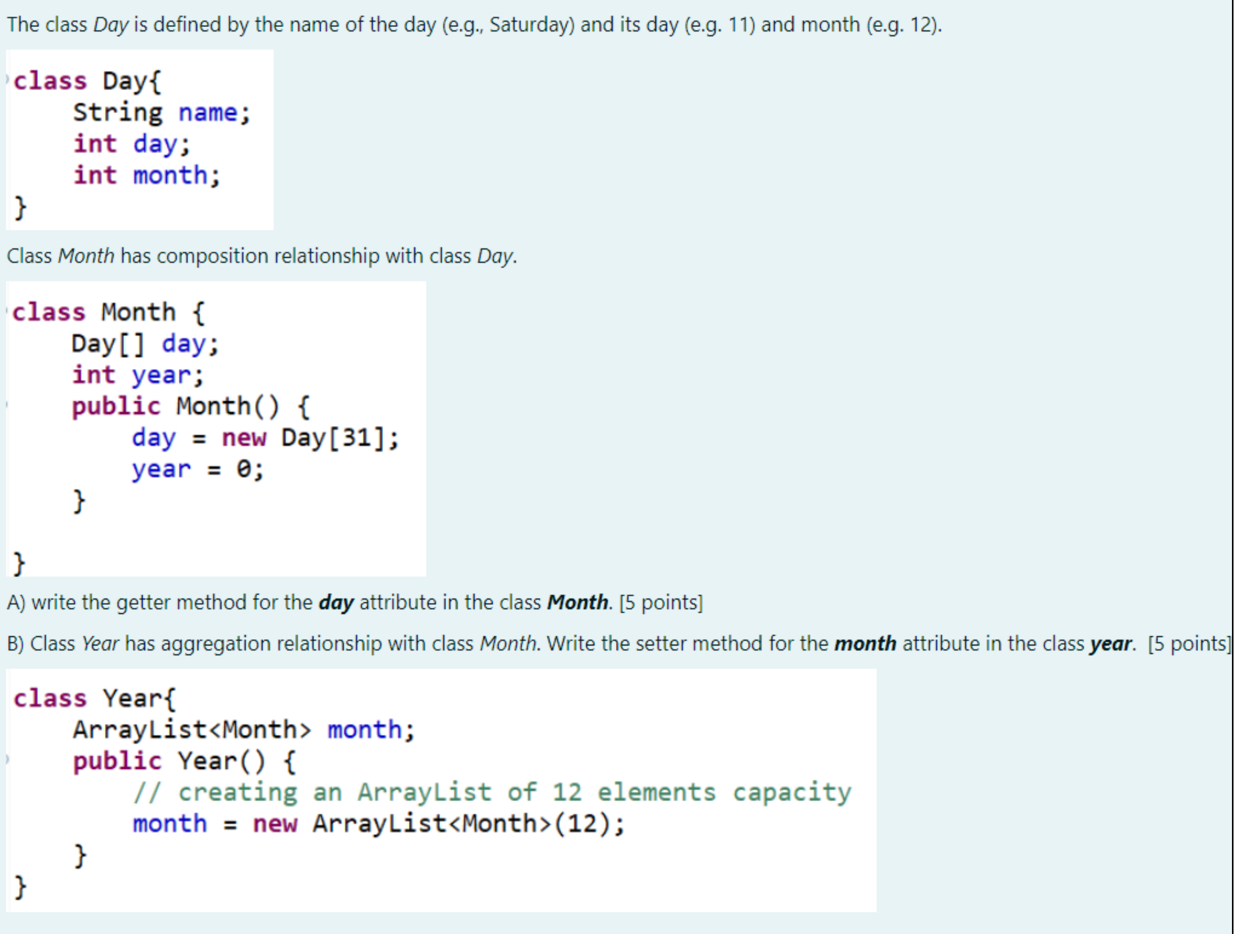


Using the code above, the following code is written. Trace the code and write what is printed.

```java
int len = 2;
Cube [] array = new Cube[len];
for (int i = 0; i < len; i++)
    array[i] = new Cube(i, i+1, i+2);
Test test = new Test(array, len);
int vol = 3;
vol = test.setVol(vol);
System.out.println(vol);
test.setVol(vol);
System.out.println(vol);

System.out.println(array[0].dime1);
test.setCube(array);
System.out.println(test.array[0].dime1);

boolean guess = true;
test.isCube(guess, 1);
System.out.println(guess);
test = new Test();
System.out.println(test.desiredVolume);
```

Use this page for your rough notes. Please note that this page will not be marked.

Q6:



The class *Day* is defined by the name of the day (e.g., Saturday) and its day (e.g. 11) and month (e.g. 12).

```
class Day{
    String name;
    int day;
    int month;
}
```

Class *Month* has composition relationship with class *Day*.

```
class Month {
    Day[] day;
    int year;
    public Month() {
        day = new Day[31];
        year = 0;
    }

}
```

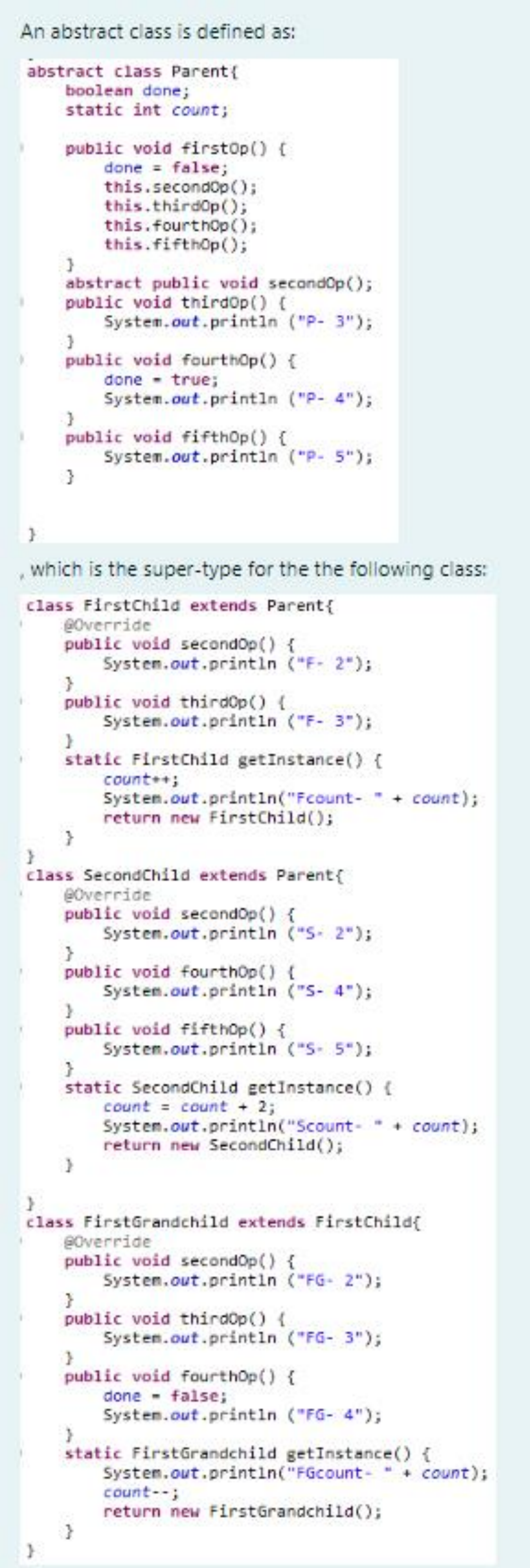
A) write the getter method for the ***day*** attribute in the class ***Month***. [5 points]

B) Class *Year* has aggregation relationship with class *Month*. Write the setter method for the ***month*** attribute in the class ***year***. [5 points]

```
class Year{
    ArrayList<Month> month;
    public Year() {
        // creating an ArrayList of 12 elements capacity
        month = new ArrayList<Month>(12);
    }
}
```

Use this page for your rough notes. Please note that this page will not be marked.

Q7:



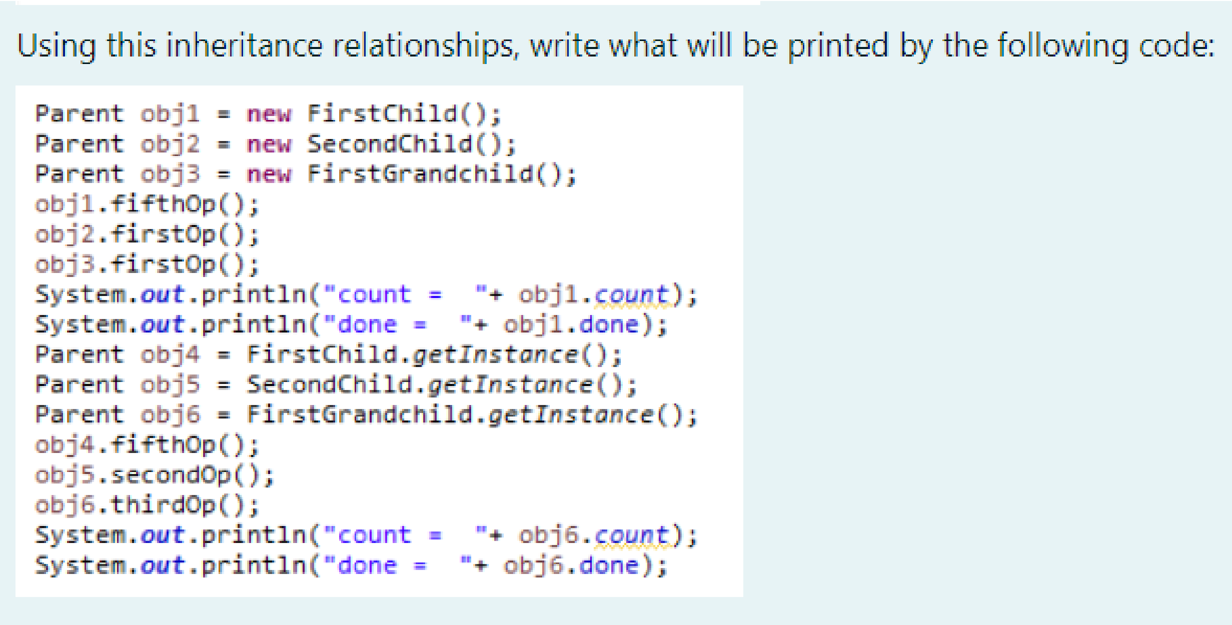
An abstract class is defined as:

```
abstract class Parent{
    boolean done;
    static int count;

    public void firstOp() {
        done = false;
        this.secondOp();
        this.thirdOp();
        this.fourthOp();
        this.fifthOp();
    }
    abstract public void secondOp();
    public void thirdOp() {
        System.out.println ("P- 3");
    }
    public void fourthOp() {
        done = true;
        System.out.println ("P- 4");
    }
    public void fifthOp() {
        System.out.println ("P- 5");
    }


}
```

, which is the super-type for the the following class:

```
class FirstChild extends Parent{
    @Override
    public void secondOp() {
        System.out.println ("F- 2");
    }
    public void thirdOp() {
        System.out.println ("F- 3");
    }
    static FirstChild getInstance() {
        count++;
        System.out.println("Fcount- " + count);
        return new FirstChild();
    }
}
class SecondChild extends Parent{
    @Override
    public void secondOp() {
        System.out.println ("S- 2");
    }
    public void fourthOp() {
        System.out.println ("S- 4");
    }
    public void fifthOp() {
        System.out.println ("S- 5");
    }
    static SecondChild getInstance() {
        count = count + 2;
        System.out.println("Scount- " + count);
        return new SecondChild();
    }

}
class FirstGrandchild extends FirstChild{
    @Override
    public void secondOp() {
        System.out.println ("FG- 2");
    }
    public void thirdOp() {
        System.out.println ("FG- 3");
    }
    public void fourthOp() {
        done = false;
        System.out.println ("FG- 4");
    }
    static FirstGrandchild getInstance() {
        System.out.println("FGcount- " + count);
        count--;
        return new FirstGrandchild();
    }
}
```

Use this page for your rough notes. Please note that this page will not be marked.

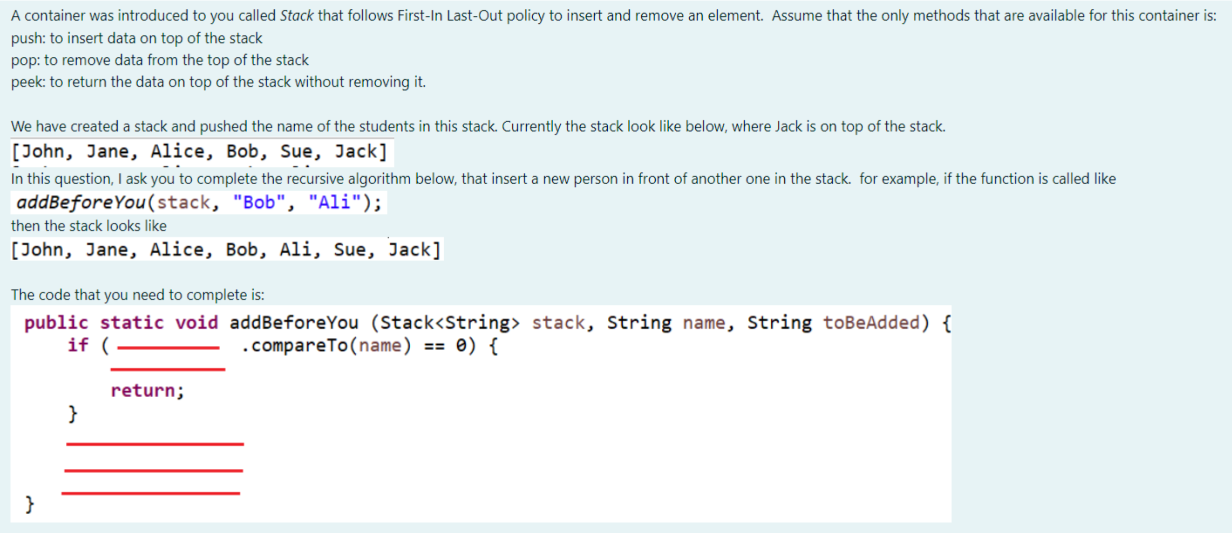


Using this inheritance relationships, write what will be printed by the following code:

```
Parent obj1 = new FirstChild();
Parent obj2 = new SecondChild();
Parent obj3 = new FirstGrandchild();
obj1.fifthOp();
obj2.firstOp();
obj3.firstOp();
System.out.println("count =  "+ obj1.count);
System.out.println("done =  "+ obj1.done);
Parent obj4 = FirstChild.getInstance();
Parent obj5 = SecondChild.getInstance();
Parent obj6 = FirstGrandchild.getInstance();
obj4.fifthOp();
obj5.secondOp();
obj6.thirdOp();
System.out.println("count =  "+ obj6.count);
System.out.println("done =  "+ obj6.done);
```

Use this page for your rough notes. Please note that this page will not be marked.

Q8:



A container was introduced to you called *Stack* that follows First-In Last-Out policy to insert and remove an element. Assume that the only methods that are available for this container is:
push: to insert data on top of the stack
pop: to remove data from the top of the stack
peek: to return the data on top of the stack without removing it.

We have created a stack and pushed the name of the students in this stack. Currently the stack look like below, where Jack is on top of the stack.

```
[John, Jane, Alice, Bob, Sue, Jack]
```

In this question, I ask you to complete the recursive algorithm below, that insert a new person in front of another one in the stack. for example, if the function is called like

```
addBeforeYou(stack, "Bob", "Ali");
```

then the stack looks like

```
[John, Jane, Alice, Bob, Ali, Sue, Jack]
```

The code that you need to complete is:

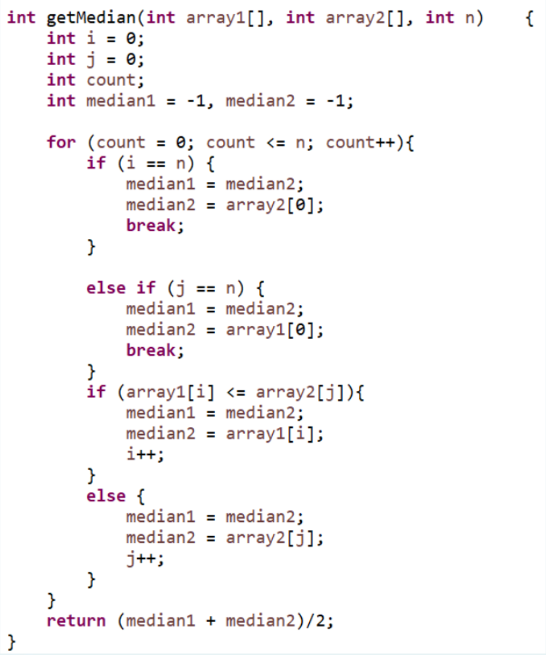
```
public static void addBeforeYou (Stack<String> stack, String name, String toBeAdded) {
    if ( ________  .compareTo(name) == 0) {
        ________
        return;
    }
    ________
    ________
    ________
}
```

Use this page for your rough notes. Please note that this page will not be marked.

Q9:

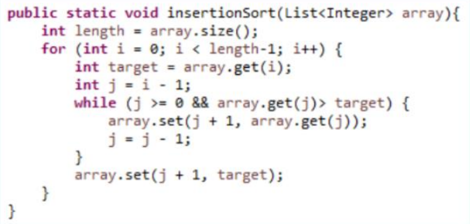
For the given algorithms, specify what is the time complexity of the algorithms using big-O notation. You do not need to write your computations. [8 points]

A)



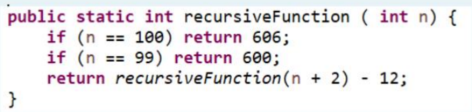
```
int getMedian(int array1[], int array2[], int n)    {
    int i = 0;
    int j = 0;
    int count;
    int median1 = -1, median2 = -1;

    for (count = 0; count <= n; count++){
        if (i == n) {
            median1 = median2;
            median2 = array2[0];
            break;
        }

        else if (j == n) {
            median1 = median2;
            median2 = array1[0];
            break;
        }
        if (array1[i] <= array2[j]){
            median1 = median2;
            median2 = array1[i];
            i++;
        }
        else {
            median1 = median2;
            median2 = array2[j];
            j++;
        }
    }
    return (median1 + median2)/2;
}
```

B)



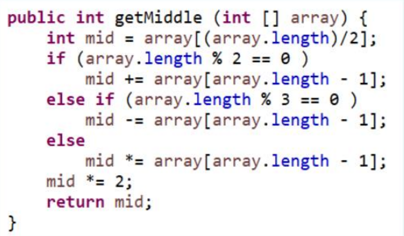
```
public static void insertionSort(List<Integer> array){
    int length = array.size();
    for (int i = 0; i < length-1; i++) {
        int target = array.get(i);
        int j = i - 1;
        while (j >= 0 && array.get(j)> target) {
            array.set(j + 1, array.get(j));
            j = j - 1;
        }
        array.set(j + 1, target);
    }
}
```

C)



```
public static int recursiveFunction ( int n) {
    if (n == 100) return 606;
    if (n == 99) return 600;
    return recursiveFunction(n + 2) - 12;
}
```

D)



```
public int getMiddle (int [] array) {
    int mid = array[(array.length)/2];
    if (array.length % 2 == 0 )
        mid += array[array.length - 1];
    else if (array.length % 3 == 0 )
        mid -= array[array.length - 1];
    else
        mid *= array[array.length - 1];
    mid *= 2;
    return mid;
}
```

Order the following functions by asymptotic growth rate. Show the power using ^ (i.e. 2^10) and ignore writing the base for the logarithm (i.e. log n), if you are not comfortable using the editor's subscript and superscript's features. [8 points]

$3n^3 + \frac{\pi}{2} n \log_2 n$ $\quad$ $1000n + 2000$ $\quad$ $n^2 log_2^n + n\, log_2 n$ $\quad$ $2^n + n + 1$

If the running time of an algorithm is computed as $T(n) = 7n^3 + 2n^2 - 2n + 1$ prove that $T(n) \in O(n^3) \quad for \quad \forall\ n \in N$ [4 points]

Use this page for your rough notes. Please note that this page will not be marked.

Answer to Q9: