Task 1

What is printed out when the code is executed?

```
public class Class1 {
	public static void main(String[] args) {
		int array[] = {14,5,7};

		for (int counter1 = 0; counter1 < array.length; counter1++) {
			for (int counter2 = counter1; counter2 > 0; counter2--) {
				if (array[counter2 - 1] > array[counter2]) {
					int variable1 = array[counter2];
					array[counter2] = array[counter2 - 1];
					array[counter2 - 1] = variable1;
				}
			}
		}

		for (int counter3 = 0; counter3 < array.length; counter3++)
			System.out.println(array[counter3]);
	}
}
```

Answer:

5, 7, 14 (insertion sort)

Task 2

What is printed out when the code is executed?

```
public class Class2 {

	public static void main (String args[]){
	    int [] variable1 = new int[128];
	    Class3 variable2 = new Class3(variable1, 0);

	    variable2.method1(7);
	    variable2.method1(2);
	    variable2.method1(8);

	    System.out.println(variable2.method2());
	    System.out.println(variable2.method2());
	    System.out.println(variable2.method2());
	}
}

public class Class3 {

	private int [] array;
	private int index;

	public Class3(int[] array, int index) {
		this.array = array;
		this.index = index;
	}

	 public void method1 (int variable3) {
		array[index] = variable3;
		index++;
	}

	public int method2 () {
```

```
                index--;
                return array[index];
        }

        public int methode3(int index) {
                return array[index];
         }
}
```

Answer:
8, 2, 7 (Class3 implements a Stack. Numbers are pushed and then popped.)

Task 3

The data structure in Classe3 implements a Stack. The implementation is very simple and can lead to problems.

a) Which poblems does that implementation have?

```
public class Stack {

        private int [] elements;
        private int index;

        public Class3(int[] elements, int index) {
                this.elements = elements;
                this.index = index;
        }

         public void push (int item) {
                elements[index] = item;
                index++;
        }

        public int pop () {
                index--;
                return elements[index];
        }

        public int top () {
                return elements[index];

}
```

b) How would you improve the implementation? (verbal description suffices; no need to write source code).

Answer:

a) No checks whether stack is full or empty; passed index is not checked

   Furthermore, a list would be better than an array.

b) Include security checks; use list instead of array.

Task 4

The source code shows a linked list. What is printed out when the code is executed?

```
public class AbstractElement implements Comparable<Object> {
```

```
	private String name;
	private int attribute;
	public AbstractElement(String n) {
		this.name = n;
	}
	public AbstractElement(String name, int attr) {
		this.attribute = attr;
		this.name = name;
	}
	public int compareTo(Object o1) {
		return this.name.compareTo(((AbstractElement) o1).name);
	}
	public String toString() {
		return this.name;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public int getAttribute() {
		return attribute;
	}
	public void setAttribute(int attribute) {
		this.attribute = attribute;
	}
}

public class LinkedList {
	Node head;
	public LinkedList() {
		this.head = new Node();
	}
	public Node getElementAt(int pos) {
		Node current = this.head;
		for (int i = 0; i <= pos; i++) {
			if (current.getNext() != null) {
				current = current.getNext();
			} else {
				return null;
			}
		}
		return current;
	}
	public void insert(AbstractElement e) {
		Node newNode = new Node(e);
		newNode.setNext(this.head.getNext());
		head.setNext(newNode);
	}
	public void insertAlgorithm(AbstractElement ae){
		Iterator iter = this.iterator();
		Node previous = iter.next();
		if (ae.compareTo(previous.getElement()) < 0) {
			insert(ae);
			return;
		}
		while (iter.hasNext()) {
			Node tmp = iter.next();
			if (ae.compareTo(tmp.getElement()) < 0) {
				Node newNode = new Node(ae);
				newNode.setNext(tmp);
				previous.setNext(newNode);
				break;
			}
			previous = tmp;
		}
```

```
        if (ae.compareTo(previous.getElement()) > 0) {
                Node newNode = new Node(ae);
                previous.setNext(newNode);
                return;
        }
}
public String removePos(int pos) {
        Node previous = null;
        Node current = this.head;
        Node delete = null;
        for (int i = 1; i <= pos; i++) {
                previous = current;
                if (current.getNext() != null) {
                        current = current.getNext();
                } else {
                        return null;
                }
                delete = current;
        }
        previous.setNext(current.getNext());
        return delete.getElement().getName();
}
public int size() {
        Node iterator = head;
        int size = 0;
        while (iterator.getNext() != null) {
                iterator = iterator.getNext();
                size++;
        }
        return size;
}
public final void sortBubbleSort() {
        AbstractElement[] array = new AbstractElement[this.size()];
        Node current = this.head;
        int i = 0;
        array[i] = current.getElement();
        while ((current != null) && ((current = current.getNext()) != null)) {
                array[i++] = current.getElement();
        }
        sort(array, this.size());
        i = 0;
        current = head;
        while ((current != null) && ((current = current.getNext()) != null)) {
                current.setElement(array[i++]);
        }
}
public final void sortPerformance() {
        long t1 = System.currentTimeMillis();
        sortBubbleSort();
        long t2 = System.currentTimeMillis();
        t2 = t2 - t1;
        System.out.println("Sortierung benˆtigte: " + t2 + " Milisekunden");
}
public void sort() {
        Runtime r = Runtime.getRuntime();
        long mem1, mem2;
        mem1 = r.totalMemory();
        mem2 = r.freeMemory();
        sortPerformance();
        mem1 = r.totalMemory() - mem1;
        mem2 = r.freeMemory() - mem2;
        System.out.println("Verwendeter Speicher: " + mem1
                        + " ; ƒnderung der Speicherauslastung: " + mem2);
}
private void sort(AbstractElement[] array, int size) {
        boolean swapped;
        do {
```

```
                        swapped = false;
                        for (int i = 0; i < size - 1; i++) {
                                if (array[i + 1].compareTo(array[i]) < 0) {
                                        swap(array, i, i + 1);
                                        swapped = true;
                                }
                        }
                } while (swapped);
        }
        private void swap(AbstractElement[] array, int index1, int index2) {
                AbstractElement swapTmp = array[index1];
                array[index1] = array[index2];
                array[index2] = swapTmp;
        }
        public String toString() {
                String result = "";
                Iterator it = this.iterator();
                while (it.hasNext()) {
                        result += it.next().toString() + "\r\n";
                }
                return result;
        }
        public Iterator iterator(){
                return new Iterator(this, 0);
        }
}

public class Node {
        private AbstractElement element;
        private Node next;
        public Node() {
                this.element = null;
                this.next = null;
        }
        public Node(AbstractElement e) {
                this.element = e;
                this.next = null;
        }
        public AbstractElement getElement() {
                return this.element;
        }
        public void setNext(Node n) {
                this.next = n;
        }
        public Node getNext() {
                return this.next;
        }
        public void setElement(AbstractElement e) {
                this.element = e;
        }
}

public class ListMain {
        public static void main(String args[]) {
                LinkedList list = new LinkedList();
                AbstractElement a1 = new AbstractElement("A1");
                AbstractElement a2 = new AbstractElement("A2");
                AbstractElement a3 = new AbstractElement("A3");
                AbstractElement a4 = new AbstractElement("A4");
                AbstractElement a5 = new AbstractElement("A5");

                list.insert(a5);
                list.insert(a4);
                list.insert(a3);
                list.insert(a2);
                list.insert(a1);
```

```
            System.out.println(list.remove(0));
        }
}
```

Answer:
A NullPointerException, because in the method remove, a != null check is missing. The loop starts with 1, but is called with 0.

Task 5

The source code shows a linked list. What is printed out when the code is executed?
(Line 1 to 15 are the same as in the last task)?

```
        public static void main(String args[]) {
                LinkedList list = new LinkedList();
                AbstractElement a1 = new AbstractElement("A1");
                AbstractElement a2 = new AbstractElement("A2");
                AbstractElement a3 = new AbstractElement("A3");
                AbstractElement a4 = new AbstractElement("A4");
                AbstractElement a5 = new AbstractElement("A5");

                list.insert(a5);
                list.insert(a4);
                list.insert(a3);
                list.insert(a2);
                list.insert(a1);

                Iterator iter = list.iterator();
                while (iter.hasNext()) {
                        System.out.println(iter.next().getElement());
                }

        }
```

Answer:
First A1, then A5; method `insert` inserts elements at the front of the list.

Task 6

What is printed out, if the method is extended as shown? (Line 1 to 20 are the same as for the last task)

```
        public static void main(String args[]) {
                LinkedList list = new LinkedList();
                AbstractElement a1 = new AbstractElement("A1");
                AbstractElement a2 = new AbstractElement("A2");
                AbstractElement a3 = new AbstractElement("A3");
                AbstractElement a4 = new AbstractElement("A4");
                AbstractElement a5 = new AbstractElement("A5");

                list.insert(a5);
                list.insert(a4);
                list.insert(a3);
                list.insert(a2);
                list.insert(a1);

                Iterator iter = list.iterator();
```

```
		while (iter.hasNext()) {
			System.out.println(iter.next().getElement());
		}

		list.insertAlgorithm(new AbstractElement("A6"));

		iter = list.iterator();
		while (iter.hasNext()) {
			System.out.println(iter.next().getElement());
		}
	}
```

Lösung:
Answer for Task 5 + A6; `insertAlgorithm` inserts elements in lists sorted in ascending order.

Task 7

The source code shows a linked list. What is printed out when the code is executed? (Line 11 to 15 insert elements in another order; A0 is added).

```
	public static void main(String args[]) {
		LinkedList list = new LinkedList();
		AbstractElement a1 = new AbstractElement("A1");
		AbstractElement a2 = new AbstractElement("A2");
		AbstractElement a3 = new AbstractElement("A3");
		AbstractElement a4 = new AbstractElement("A4");
		AbstractElement a5 = new AbstractElement("A5");

		list.insert(a1);
		list.insert(a2);
		list.insert(a3);
		list.insert(a4);
		list.insert(a5);

		list.insertAlgorithm(new AbstractElement("A0"));

		iter = list.iterator();
		while (iter.hasNext()) {
			System.out.println(iter.next().getElement());
		}
	}
```

Answer:
Inserting sorted works only with lists sorted in ascending order. Thus, the answer is: A5, A0, A4, A3, A2, A1

Task 8
Implement a method to insert elements in a list sorted in descending order.
Example
Sorted list:
A6 A4 A3 A2 A1

When A5 is inserted, the order should be kept:
A6 A5 A4 A3 A2 A1

Answer:

Use insertAlgorithm and change the comparisons; i.e., > changes to <, and < changes to >. Another way is to call insert and then adjust and use method bubbleSort.

Task 9

What is printed out when the code is executed?

```
public class Test
    public static void main(String[] args){
       String ausgabe = null;
       if(args[0].equals("Hello"))  {
              ausgabe = args[0] + " World";
       }
       System.out.println(ausgabe);
    }
}
```

Answer:
Depending on whether "Hello" is passed as command-line parameter:
Yes: Hello World
No: NullPointerException

Task 10
You see a program for the manipulation of multi-media data on mobile devices. You can manage photos in albums, set photos as favorites, and sort photos.
However, when you add a picture to an album, a NullPointerException is thrown. What is the problem?

Answer:
Variable `photopathtxt` in class `AddPhotoToAlbum` is initialized with null.