Section 1

Question	Question title	Marks	Question type
ℹ	Examination Informtion		Information or resources
1	Question 1: Object-Oriented Programming	10	True / False
2	Question 2: Exceptions and Exception Handling	10	True / False
3	Question 3: Programming Paradigms	10	True / False
4	Question 4: Java Declarations	10	True / False
5	Question 5: Multi-Threading	10	True / False

Section 2

Question	Question title	Marks	Question type
6	Question 6: Object Oriented Design (1)	10	True / False
7	Question 7: Object-Oriented Design (2)	10	Upload Assignment
8	Question 8: Requirements Analysis	10	Upload Assignment
9	Question 9: Code Writing and Interpretation	10	Text area
10	Question 10a: Problem Solving	5	Programming
11	Question 10b: Problem Solving	10	Programming
12	Question 10c: Problem Solving	5	Programming

13	Question 11: Threads	10	Text area

7 Question 7: Object-Oriented Design (2)

A library has a stock of books. Some of the books are on the shelves in the public area of the library; a smaller number are kept in a store that the public do not have access to. For each book in the library's stock there are one or more copies which may be kept in different locations, i.e. some copies may be in the store and a few on the shelves in the public area. Each book has a bibliographic record, and each physical copy of the book has a record of the date it was purchased as well as its location.

Draw a UML class diagram that represents the library system described above. The relationships between classes and multiplicities should be clear.

Upload your file here. Maximum one file.

All file types are allowed. Maximum file size is **1 GB**

Select file to upload

Maximum marks: 10

Attaching sketches to this question?
Use the following code:

X X X X X X X

8 Question 8: Requirements Analysis

You have been asked to create a use case diagram as part of the requirements analysis for an online forum. The specification states that there are two kinds of users that interact in the forum with different responsibilities: Users and Administrators. Both must log in to the system before they can read forum messages and reply to them. Part of logging in requires the use of an authentication process, and both types of use need to have registered with the system, which also uses the authentication process. All authenticated users can post new messages to the forum, though new users may only post 10 messages per week until the Administrators remove the limit. Only Administrators can check statistics and create new threads. Users other than administrators are able to send private messages to other users.

Draw a UML Use Case diagram containing Actors, Use Cases and their relationships for the scenario described above.

Upload your file here. Maximum one file.

All file types are allowed. Maximum file size is **1 GB**

Select file to upload

Maximum marks: 10

Attaching sketches to this question?
Use the following code:

X X X X X X X

9 Question 9: Code Writing and Interpretation

The code displayed to the left compiles and runs without error.

Write out the output of the program when it is executed on the command line.

(NB %d and %-10s are placeholders in the format string. %d prints an integer. %-10s aligns a string to the left hand side of a space that is 10 spaces wide.)

Write your answer in the box below. Changes are saved automatically.

Maximum marks: 10

Attaching sketches to this question?
Use the following code:

X X X X X X X

10 Question 10a: Problem Solving

A developer has started to write an array-based class that implements the java.util.List interface. You have been given the code that has been written so far that is in the PDF on the left. The developer's notes are in the Javadoc comments in the code.

The developer has marked the boolean remove(Object o) method with the comment // FIXME.

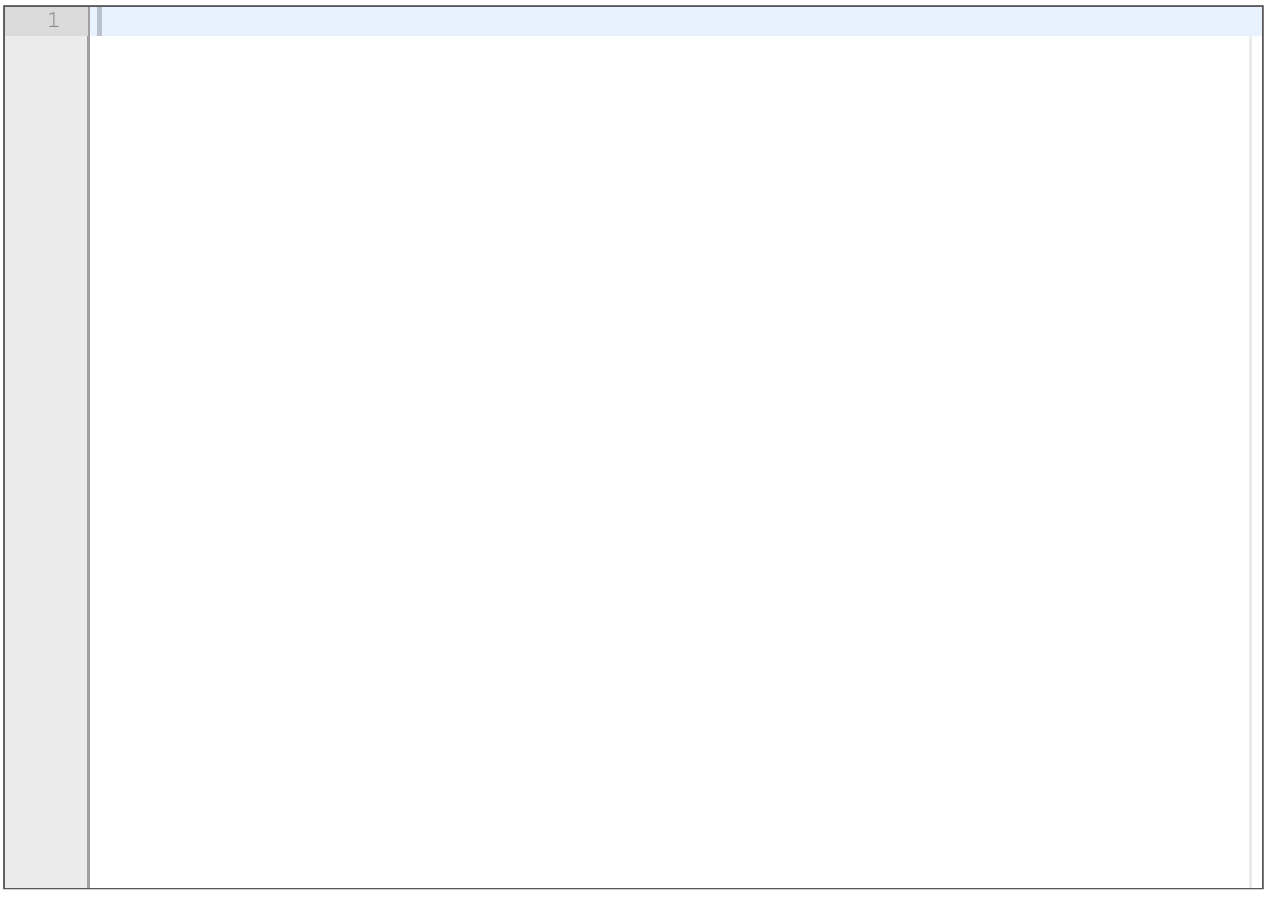
A summary of the Java documentation that specifies the method is as follows:

boolean remove(Object o)
Removes the first occurrence of the specified element from this list, if it is present. If this list does not contain the element, it is unchanged. More formally, removes the element with the lowest index i such that Objects.equals(o, get(i)) (if such an element exists). Returns true if this list contained the specified element (or equivalently, if this list changed as a result of the call).

Throws
NullPointerException if the specified element is null and this list does not permit null elements (optional)
Implement the boolean remove(Object o) method to meet the above specification.

Implement the boolean remove(Object o) method to meet the above specification.

Write your answer in the box below. Changes are saved automatically.



Maximum marks: 5

Attaching sketches to this question?
Use the following code:

X X X X X X X

11 Question 10b: Problem Solving

The Java documentation specifies the following method:

void add(int index, E element)

Inserts the specified element at the specified position in this list. Shifts the element currently at that position (if any) and any subsequent elements to the right (adds one to their indices).

Parameters:

index - index at which the specified element is to be inserted
element - element to be inserted

Throws (two other exceptions ignored):

NullPointerException - if the specified element is null and this list does not permit null elements
IllegalArgumentException - if some property of the specified element prevents it from being added to this list
IndexOutOfBoundsException - if the index is out of range (index < 0 || index > size())

Implement the method void add(int index, E element) for the ArrayList class in the PDF document to the left according to the specification above.

Write your answer in the box below. Changes are saved automatically.

```
1
```

Maximum marks: 10

Attaching sketches to this question?
Use the following code:

X X X X X X X

12

Question 10c: Problem Solving

While adding functionality to the ArrayList class, you realise that the developer has implemented some operations in linear time (O(n)) that could be implemented as constant time operations (O(1)). Suggest revisions to the class so that operations such as size() might be implemented in constant time. (There is no need to write code to implement your ideas, though you can write fragments of code to illustrate changes.)

Write your answer in the box below. Changes are saved automatically.

```
1
```

Maximum marks: 5

Attaching sketches to this question?
Use the following code:

X X X X X X X

13 # Question 11: Threads

Consider the Java program to the left:

(Please note that the code is syntactically correct, compiles and executes.)

(a) Write out the output from the program (3 points)

Write your answer in the box below. Changes are saved automatically.

(b) Explain your answer to part (a) and justify why the program output is as you describe (4 points)

Write your answer in the box below. Changes are saved automatically.

(c) Explain how the BankingTask class above might be improved (3 points) (You should describe the revisions required; there is no need to write code.)

Write your answer in the box below. Changes are saved automatically.

Maximum marks: 10

Attaching sketches to this question?
Use the following code:

X X X X X X X

Question 10
Attached

```java
import java.util.List;

/**
   A class that implements the java.util.List interface using an array.

   The class should have a default capacity of twenty elements. Null
   elements are not permitted.  null is used to represent an empty
   slot in the array and the array is always filled from the left
   (lower indices), so that any emply slots are always at higher
   indices (to the right) than the last occupied slot in the
   array. i.e. 5 elements occupy positions 0-4.
 */
public class ArrayList<E> implements List<E> {
    private E[] store;

    @SuppressWarnings("unchecked")
    public ArrayList () {
        store = (E[]) new Object[20];
    }

    @Override
    public boolean isEmpty() {
        return store[0] == null;
    }

    @Override
    public int size() {
        int count = 0;
        while (store[count] != null) {
            count++;
        }
        return count;
    }


    @Override
    public E get(int index) {
        return store[index];
    }


    @override
    public int indexOf(Object o) {
        int index = 0;
        while (store[index] != null) {
            if (store[index].equals(o)) {
                return index;
            }
            index++;
        }
        return -1;
    }

    @Override
    public E remove(int index) {
```

```
56          if (index < 0 || size() < index) {
57              throw new ArrayIndexOutOfBoundsException();
58          }
59
60          E element = store[index];
61
62          while (store[index] != null) {
63              store[index] = store[++index];
64          }
65
66          return element;
67      }
68
69
70      @Override
71      public boolean remove(Object o) {
72          // FIXME
73          throw new UnsupportedOperationException("remove(Object o) not implemented");
74      }
75
76      @SuppressWarnings("unchecked")
77      public boolean add(E element) {
78          if (element == null) {
79              throw new NullPointerException("null objects cannot be stored in the list!!");
80          }
81
82          int numberOfStoredElements = size();
83          int capacity = store.length;
84          if (capacity == numberOfStoredElements) {
85              E[] newstore = (E[]) new Object[20 + capacity];
86              for (int index = 0; index < store.length; index++) {
87                  newstore[index] = store[index];
88              }
89              store = newstore;
90          }
91
92          store[numberOfStoredElements] = element;
93
94          return true;
95      }
96
97  }
```

Question 11
Attached

```java
import java.util.List;

/**
   A class that implements the java.util.List interface using an array.

   The class should have a default capacity of twenty elements. Null
   elements are not permitted.  null is used to represent an empty
   slot in the array and the array is always filled from the left
   (lower indices), so that any emply slots are always at higher
   indices (to the right) than the last occupied slot in the
   array. i.e. 5 elements occupy positions 0-4.
 */
public class ArrayList<E> implements List<E> {
    private E[] store;

    @SuppressWarnings("unchecked")
    public ArrayList () {
        store = (E[]) new Object[20];
    }

    @Override
    public boolean isEmpty() {
        return store[0] == null;
    }

    @Override
    public int size() {
        int count = 0;
        while (store[count] != null) {
            count++;
        }
        return count;
    }


    @Override
    public E get(int index) {
        return store[index];
    }


    @override
    public int indexOf(Object o) {
        int index = 0;
        while (store[index] != null) {
            if (store[index].equals(o)) {
                return index;
            }
            index++;
        }
        return -1;
    }

    @Override
    public E remove(int index) {
```

```
56          if (index < 0 || size() < index) {
57              throw new ArrayIndexOutOfBoundsException();
58          }
59
60          E element = store[index];
61
62          while (store[index] != null) {
63              store[index] = store[++index];
64          }
65
66          return element;
67      }
68
69
70      @Override
71      public boolean remove(Object o) {
72          // FIXME
73          throw new UnsupportedOperationException("remove(Object o) not implemented");
74      }
75
76      @SuppressWarnings("unchecked")
77      public boolean add(E element) {
78          if (element == null) {
79              throw new NullPointerException("null objects cannot be stored in the list!!");
80          }
81
82          int numberOfStoredElements = size();
83          int capacity = store.length;
84          if (capacity == numberOfStoredElements) {
85              E[] newstore = (E[]) new Object[20 + capacity];
86              for (int index = 0; index < store.length; index++) {
87                  newstore[index] = store[index];
88              }
89              store = newstore;
90          }
91
92          store[numberOfStoredElements] = element;
93
94          return true;
95      }
96
97  }
```

Question 12
Attached

```java
import java.util.List;

/**
   A class that implements the java.util.List interface using an array.

   The class should have a default capacity of twenty elements. Null
   elements are not permitted.  null is used to represent an empty
   slot in the array and the array is always filled from the left
   (lower indices), so that any emply slots are always at higher
   indices (to the right) than the last occupied slot in the
   array. i.e. 5 elements occupy positions 0-4.
 */
public class ArrayList<E> implements List<E> {
    private E[] store;

    @SuppressWarnings("unchecked")
    public ArrayList () {
        store = (E[]) new Object[20];
    }

    @Override
    public boolean isEmpty() {
        return store[0] == null;
    }

    @Override
    public int size() {
        int count = 0;
        while (store[count] != null) {
            count++;
        }
        return count;
    }


    @Override
    public E get(int index) {
        return store[index];
    }


    @override
    public int indexOf(Object o) {
        int index = 0;
        while (store[index] != null) {
            if (store[index].equals(o)) {
                return index;
            }
            index++;
        }
        return -1;
    }

    @Override
    public E remove(int index) {
```

```
56          if (index < 0 || size() < index) {
57              throw new ArrayIndexOutOfBoundsException();
58          }
59
60          E element = store[index];
61
62          while (store[index] != null) {
63              store[index] = store[++index];
64          }
65
66          return element;
67      }
68
69
70      @Override
71      public boolean remove(Object o) {
72          // FIXME
73          throw new UnsupportedOperationException("remove(Object o) not implemented");
74      }
75
76      @SuppressWarnings("unchecked")
77      public boolean add(E element) {
78          if (element == null) {
79              throw new NullPointerException("null objects cannot be stored in the list!!");
80          }
81
82          int numberOfStoredElements = size();
83          int capacity = store.length;
84          if (capacity == numberOfStoredElements) {
85              E[] newstore = (E[]) new Object[20 + capacity];
86              for (int index = 0; index < store.length; index++) {
87                  newstore[index] = store[index];
88              }
89              store = newstore;
90          }
91
92          store[numberOfStoredElements] = element;
93
94          return true;
95      }
96
97  }
```

Question 13
Attached

```java
public class Transaction {

    private Account source;
    private Account destination;
    private int amount;  // in öre

    // a simple class to move money between two bank accounts
    public Transaction(int amount, Account source, Account destination) {
        this.amount = amount;
        this.destination = destination;
        this.source = source;
    }

    public int getAmount() { return this.amount;}

    public Account getSource() {return this.source;}

    public Account getDestination() {return this.destination;}



    public static void main(String[] args) {
        Account account1 = new Account("Johan's account", 25000);
        Account account2 = new Account("Emma's account", 40000);

        Transaction transaction1 = new Transaction(4000,account1,account2);
        Transaction transaction2 = new Transaction(5000,account2,account1);

        new BankingTask("transaction 1", transaction1).start();
        new BankingTask("transaction 2", transaction2).start();
    }
}

class Account {
    private String accountName;
    private int balance; // in öre

    Account(String name, int openingBalance) {
        this.accountName = name;
        this.balance = openingBalance;
    }

    String getName() { return this.accountName; }
}

class BankingTask extends Thread {
    private Transaction transaction;

    BankingTask(String name, Transaction transaction) {
        super(name);
        this.transaction = transaction;
    }

    public void run() {
        Account source = transaction.getSource();
        synchronized (source) {
            System.out.println(getName() + " locked " + source.getName());

            try {
                // Simulate some work with the source account
                Thread.sleep(50);
            } catch (InterruptedException e) {
```

```
                e.printStackTrace();
            }

            Account destination = transaction.getDestination();
            System.out.println(getName() + " trying to lock " + destination.getName());

            synchronized (destination) {
                System.out.println(getName() + " locked " + destination.getName());
                try {
                    // Simulate some work with the destination account
                    Thread.sleep(50);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }

    }
}
```

Question 9
Attached

```java
import java.util.List;
import java.util.ArrayList;

public class FruitMachine {
    private Reel one;
    private Reel two;
    private Reel three;

    FruitMachine() {
        one = new Reel(new String[] {"Bell", "Star", "Apple", "Cherry", "Pineapple"});
        two = new Reel(new String[] {"Cherry", "Star", "Bell", "Apple"});
        three = new Reel(new String[] {"Star", "Pineapple", "Cherry", "Bell", "Apple", "Cherry"});
    }

    void play() {
        for (int i = 0; i < 4; i++) {
            System.out.println(String.format("%d: %-10s %-10s %-10s\n",
                                             i+1, one.spin(), two.spin(), three.spin()));
        }
    }

    public static void main(String[] args) {
        FruitMachine machine = new FruitMachine();

        machine.play();
    }
}

class Reel {
    private List<String> values;
    private int step;
    private int index = 0;

    Reel(String[] items) {
        values = new ArrayList<>();
        for( String item : items) {
            values.add(item);
        }

        step = 2 * values.size() % 7;
    }

    String spin() {
        index = (index + step) % values.size();
        return values.get(index);
    }
}
```