1 Unusual programs

1. Consider the following Java program :

```
public class Thread2 extends Thread {

        public int id;
        public Thread2 bro;
        public Thread2(int id, Thread2 bro) {
                this.id = id; this.bro = bro; }

        public void run(){
                if (this.id == 0) System.out.print("3");
                else {
                        bro.join();
                        System.out.print("4");
                }
                System.out.print("0");
        }
}

public class Thread1 extends Thread{

        public int id;
        public Thread1(int id){ this.id = id; }

        public void run(){
                if (this.id == 0){

                        Thread t = new Thread2 (1, null);
                        t.bro = new Thread2(0, t);

                        t.bro.start();
                        t.start();

                }

                else System.out.print("2");
        }

public static void main(String[] args){
        new Thread1(0).start();
        new Thread1(1).start();

        }
}
```

Questions What are the possible outputs ?

- 32040 Y N
- 34002 Y N
- 30402 Y N
- 23040 Y N
- 40302 Y N

2. Consider the following Java program:

```
public class Thread1 extends Thread{

        public int id;
        public int result;
        public Thread1 bro;


        public Thread1(int id, Thread1 bro){
          this.id = id; this.result = -1; this.bro = bro;
        }

        public void run(){
                System.out.println(1 - this.id);
                if (id == 0){
                        bro.join();
                        System.out.println(bro.result);
                }
                else this.result = 2;
                System.out.println("Bye");
        }


        public static void main(String[] args){
                Thread1 t1 = new Thread1(0,null);
                Thread1 t2 = new Thread1(1,t1);

                t1.bro = t2;

                System.out.println("Hello");

                t1.start();
                t2.start();

        }

}
```

Questions Give 3 valid outputs of the program.

3. Consider the following Java program:

```
public class Example extends Thread{

        public int counter;
        public int l_i;

        public Example(int cpt, int l_i){
          this.counter = cpt; this.l_i = l_i;
        }


        public void run(){
          int i ;
          for (i = l_i;i<2;i++){
            System.out.println("counter = " + this.counter);
            (new Example(this.counter +1,i + 1)).start();
            this.counter++;
          }

          System.out.println("counter = " + this.counter);

        }


        public static void main(String[] args){

                Example e = new Example(0,0);
                e.start();

        }
}
```

Questions A. What are the possible values of `counter` printed in the first line?

B. What are the possible values of `counter` printed in the last line?

C. How many times could the value of `counter` be printed ?

2 Concurrency problems

4.

```
public class LinkedList {

        private LinkedList next;
        private final ReentrantLock l_add = new ReentrantLock();
        private final ReentrantLock l_remove = new ReentrantLock();

        public LinkedList(){
                this.next = null;
        }

        public LinkedList add(){
                l_add.lock();
                LinkedList res = new LinkedList();
                res.next = this;
                l_add.unlock();
                return res;
        }

        public LinkedList remove(){
                l_remove.lock();
                if (this == null) return null;
                LinkedList res = this.next;
                this.next = null;
                l_remove.unlock();
                return res;
        }

}
```

Questions

The implementation may deadlock	Y	N
The implementation may crash	Y	N
The function *add* always add at the beginning of the list	Y	N
The fuction *remove* always remove the first element of the list	Y	N
Using only one lock is possible	Y	N

5. This exercice is here to show you what to NEVER do.

```
public class BankAccount {
    private int balance = 0;
    private Lock lk = new Lock();

    int getBalance() {
        lk.lock();
        int ans = balance;
        lk.unlock();
        return ans;
    }
    void setBalance(int x) {
        lk.lock();
        balance = x;
        lk.unlock();
    }
    void withdraw(int amount) {
        lk.lock();
        int b = getBalance();
        if(amount > b) {
            lk.unlock();
            throw new WithdrawTooLargeException();
        }
        setBalance(b - amount);
        lk.unlock();
    }
}
```

Questions

1. Explain why, if the lock is not re-entrant, this implementation will deadlock.
2. Still considering this is not a re-entrant lock, you replace the function withdraw by :

```
void withdraw(int amount) {
  lk.lock();
  lk.unlock();
  int b = getBalance();
  lk.lock();
  if(amount > b) {
    lk.unlock();
    throw new WithdrawTooLargeException();
  }
  lk.unlock();
  setBalance(b - amount);
  lk.lock();
  lk.unlock();
}
```

Does it work now ? Can it be improved ?

6. We implemented a new Bank Account class trying to solve the problem.We use a class Account to manage the bank account and a class Transaction class to manage the use of Account functions.

```
  public class Account {
    double balance;
    int id;
    Lock lock = new ReentrantLock();

    Account(int id, double balance) {
        this.id = id;
        this.balance = balance;
    }

    boolean withdraw(double amount) {
        if (this.lock.tryLock()) {
            balance -= amount;
            this.lock.unlock();
            return true;
        }
        return false;
    }

    boolean deposit(double amount) {
        if (this.lock.tryLock()) {
            balance += amount;

            this.lock.unlock();
            return true;
        }
        return false;
    }

    public boolean tryTransfer(Account account, double amount) {
        if (this.withdraw(amount)) {
            if (account.deposit(amount)) {return true;}
            // else destination account busy, refund source account.
            this.deposit(amount);
            }

        return false;
    }

    public static void main(String[] args) {
        Account a1 = new Account(1, 500d);
        Account a2 = new Account(2, 500d);

        new Thread(new Transaction(a1, a2, 10d), "tr-1");).start();
        new Thread(new Transaction(a2, a1, 10d), "tr-2");).start();
    }
}
class Transaction implements Runnable {
    private Account source, destination;
    private double amount;

    Transaction(Account source, Account destination, double amount) {
        this.source = source;
        this.destination = destination;
```

```
        this.amount = amount;
    }

    public void run() {
        while (!source.tryTransfer(destination, amount));
}
```

Questions

1. Unfortunately, this code is flawed. Can you explain the problem ?
2. We change the class Transaction by

```
class Transaction implements Runnable {
    private Account sourceAccount, destinationAccount;
    private double amount;
    private final static Lock l = new Lock();

    Transaction(Account source, Account dest, double amount) {
        this.source = source;
        this.destination = dest;
        this.amount = amount;
    }

    public void run() {
        if (!source.tryTransfer(destination, amount)){
          l.lock();
          while (!source.tryTransfer(destination, amount));
          l.unlock();
        }
    }
}
```

   Does it work now ? If yes, explain the difference. If not, tell what is missing.