Computation structures

Problem-solving lesson 7

1. Consider the following Java class:

```java
public class MyClass {
    public MyClass() {}
    public synchronized void m1() {
        System.out.println("Entering method 1");
        try{Thread.sleep(5000);}catch(Exception e){}
        System.out.println("Exiting method 1");
    }
    public synchronized void m2() {
        System.out.println("Entering method 2");
        try{Thread.sleep(5000);}catch(Exception e){}
        System.out.println("Exiting method 2");
    }
}
```

   What could be the outcome of the following programs?

```java
MyClass o1, o2;
o1 = new MyClass();   o2 = new MyClass();
new Thread() {public void run() {
        o1.m1();}}.start();
new Thread() {public void run() {
        o1.m2();}}.start();
```

```java
MyClass o1, o2;
o1 = new MyClass();   o2 = new MyClass();
new Thread() {public void run() {
        o1.m1();}}.start();
new Thread() {public void run() {
        o2.m1();}}.start();
```

2. An animal shelter has a room to temporarily store animals that transit from their cages to the vet clinic and reversely. Rules are :

   - The room is only used to hold cats or dogs.
   - A cat can never enter the room if it already contains a cat or a dog.
   - A dog can never enter the room if it already contains a cat.
   - There cannot be more than 4 dogs in the room.

   Write a solution to this problem using *synchronized* methods as well as $wait()$, $notify()$ and $notifyAll()$ calls. Use variables *cats* and *dogs* to represent the number of cats and dogs in the room respectively.

3. A bank asks your help to develop a Java program that performs the payments. Bank accounts are stored in objects of class *Account* that advertise three non-atomic methods: *void credit*(*double amount*), *void debit*(*double amount*) and *String getIBAN*() allowing to credit a certain amount to, debit a certain amount from, or show the IBAN of the account, respectively.

   You must write a method called $transfer$(*Account from*, *Account to*, *double amount*) that will be used in the context of multi-threading, and ensure synchronization is performed in such a way as to keep the accounts in a coherent state while avoiding deadlocks.