Práctica 5
Comunicación y sincronización II

Objetivo
Revisar el modelo de sincronización de Java.

Actividad 1. Modifique el siguiente código para que el sistema de cuentas bancarias no se corrompa.

```
/**
   Este programa muestra la corrupción de datos cuando múltiples
   threads accesan sin sincronización una estructura de datos.
*/

public class UnsynchBankTest
{
  public static final int NACCOUNTS = 100;
  public static final double INITIAL_BALANCE = 100;

  public static void main(String[] args)
   {
      Bank b = new Bank(NACCOUNTS, INITIAL_BALANCE);
      int i;
      for (i = 0; i < NACCOUNTS; i++)
      {
        TransferRunnable r = new TransferRunnable(b, i, INITIAL_BALANCE);
        Thread t = new Thread(r);
        t.start();
      }
   }
}

/**
   Un banco con cierto número de cuentas.
*/

class Bank
{

   private final double[] accounts;

   /**
      Construye el banco.
      n = número de cuentas
      initialBalance = saldo inicial de cada cuenta
   */

   public Bank(int n, double initialBalance)
   {
      accounts = new double[n];

      for (int i = 0; i < accounts.length; i++)
         accounts[i] = initialBalance;
   }

   /**
      Función que transfiere dinero de una cuenta a otra.
```

```
         from = la cuenta de donde se toma dinero
         to = la cuenta hacia donde se transfiere el dinero
         amount = la cantidad a transferir
      */

      public void transfer(int from, int to, double amount)
      {
         if (accounts[from] < amount) return;
         accounts[from] -= amount;
         System.out.printf("$%10.2f transferido de la cuenta %d a la cuenta %d\n",
amount, from, to);
         accounts[to] += amount;
         System.out.printf(" Saldo total: %10.2f \n", getTotalBalance());
      }

      /**
         Calcula la suma de todos los saldos de las cuentas.
         regresa el saldo total
      */

      public double getTotalBalance()
      {
         double sum = 0;

         for (double a : accounts)
            sum += a;

         return sum;
      }

      /**
        Regresa el número de cuentas
      */
      public int size()
      {
         return accounts.length;
      }
   }

   /**
      Objeto que transfiere dinero de una cuenta a otra dentro del banco.
   */

   class TransferRunnable implements Runnable
   {

     private Bank bank;
     private int fromAccount;
     private double maxAmount;
     private int DELAY = 10;

      /**
         Construye el objeto que hace las transferencias.
         b = el banco que maneja las cuentas
         from = la cuenta de donde se toma el dinero
         max = el monto máximo de dinero en cada transferencia
      */
      public TransferRunnable(Bank b, int from, double max)
      {
         bank = b;
         fromAccount = from;
         maxAmount = max;
      }
```

```
    public void run()
    {
       try
       {
          while (true)
          {
             int toAccount = (int) (bank.size() * Math.random());
             double amount = maxAmount * Math.random();
             bank.transfer(fromAccount, toAccount, amount);
             Thread.sleep((int) (DELAY * Math.random()));
          }
       }
       catch (InterruptedException e) {}
    }
 }
```

Actividad 2. Modifique el código de tal manera que si una cuenta no tiene el dinero suficiente para hacer el retiro correspondiente, en lugar de abortar la transacción, ésta se suspenda hasta que a la cuenta le sea abonada una cantidad que permita proceder al retiro.