Segundo Certamen
Tiempo 120 min. Al terminar, entregar sus respuestas vía AULA.

Usted deberá desarrollar los programas mostrados en la Figura 1.

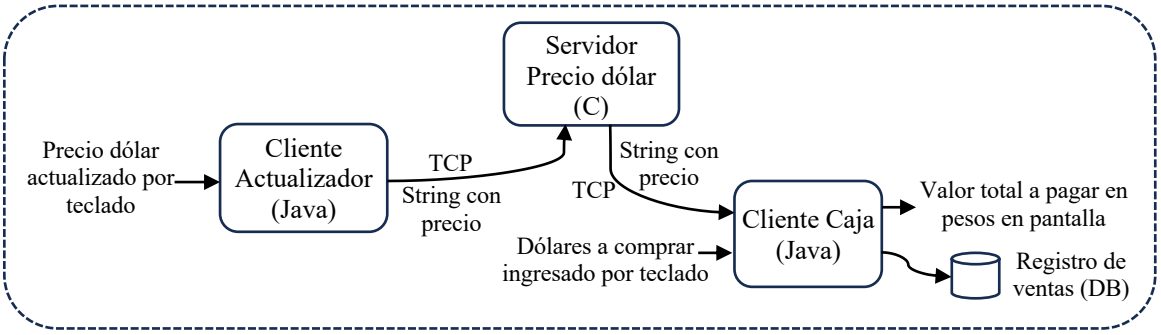

Figura 1: Programas involucrados en la situación del certamen

Idea general: Una empresaria tiene una caja de cambios. Para informar el valor de venta de dólares, ella usa un programa cliente para actualizar el precio de venta de dólares almacenado en un servidor central. Una caja de cambios recibe las actualizaciones del precio del dólar tan pronto éstas se producen en el servidor. Así la caja maneja una copia local para el precio, la cual usa cada vez que se hace una venta de dólares. El orden de ejecución es: Servidor precios, Cliente actualizador, Cliente caja.

Pregunta 1 (20 pts): Programe en Java el cliente Actualizador.java
Sintaxis: $ java Actualizador <Servidor precio> <puerto escucha de este servidor>
Descripción: Actualizador crea un **socket TCP** y luego lee y envía string ingresados por teclado con actualizaciones del precio del dólar. Este valor es inmediatamente reportado al servidor de precio. Actualizador termina al ingresar un precio negativo.
Sólo un cliente Actualizador correo por vez.
Suba a AULA Actualizador.java y eventualmente otras clases que usted haya creado.

R: (ver aquí)

```
// $ java Actualizador <Servidor precio> <puerto escucha de este servidor>
import java.net.*;
import java.io.*;
import java.util.Scanner;

public class Actualizador {
  public static void main(String[] args) throws IOException {
    int port=Integer.parseInt(args[1]);
    Socket socket = new Socket(args[0], port);
    Scanner in = new Scanner(System.in);
    try {
      PrintWriter out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(
                                                    socket.getOutputStream())),true);
      while(true) {
        double precio = in.nextDouble();
        if (precio < 0) System.exit(0);
        out.println(precio);
      }
    } finally {
      System.out.println("closing...");
      socket.close();
    }
  }
}
```

8 pts. Creación y conexión de socket

8 pts. Lectura y envío de precio del dólar
4pts. Condición de término

Pregunta 2 (30 pts): Programe en lenguaje C el Servidor Precio Dólar, servidorPrecio.c
Sintaxis: $ ./servidorPrecio <puerto TCP de escucha Actualizador> <puerto TCP escucha caja>
Descripción: En "puerto TCP de escucha Actualizador", este servidor atiende a un único cliente Actualizador. En "puerto TCP de escucha caja", este servidor atiende a una única caja. ServidorPrecio recibe y mantiene un string con la copia del precio del dólar de manera de reportarlo a la caja cliente tan pronto ésta se conecta. ServidorPrecio hace llegar a la Caja actualizaciones posteriores del precio tan pronto son informadas por Actualizador.
Suba su respuesta a AULA con nombre servidorPrecio.c

R: Solución utilizando select (se requiere concurrencia para aceptar nuevos actualizaciones del precio mientras se conecta la Caja) (ver aquí)

```c
/*
SYNOPSIS:    servidorPrecio <puerto TCP de escucha Actualizador> <puerto TCP escucha caja>
*/

#include <stdio.h>
#include <netdb.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>

void reusePort(int sock);

int main(int argc,  char *argv[]) {
  fd_set readfds, readfdsCopy;
  int n;
  int actSock=0, cajaSock=0;
  struct addrinfo hints, *result, *rp;
  int as, cs;
  struct sockaddr_storage from_addr;
  socklen_t from_addr_len;
  char precio[50];

  memset(&hints, 0, sizeof(hints));

  hints.ai_family = AF_INET; //IPv4
  hints.ai_socktype = SOCK_STREAM; //TCP
  hints.ai_flags = AI_PASSIVE; //For wildcard IP address

  if ((getaddrinfo(NULL, argv[1], &hints, &result)) != 0)   {
    fprintf(stderr, "getaddrinfo error\n");
    exit(EXIT_FAILURE);
  }

  for (rp = result; rp != NULL; rp = rp -> ai_next)  {
    as = socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol);
    reusePort(as);
    if (as == -1) continue;
    if (bind(as, rp -> ai_addr, rp -> ai_addrlen) == 0)  break; //Success
    close(as);
  }
  memset(&hints, 0, sizeof(hints));

  hints.ai_family = AF_INET; //IPv4
  hints.ai_socktype = SOCK_STREAM; //TCP
  hints.ai_flags = AI_PASSIVE; //For wildcard IP address

  if ((getaddrinfo(NULL, argv[2], &hints, &result)) != 0)   {
    fprintf(stderr, "getaddrinfo error\n");
    exit(EXIT_FAILURE);
  }
```

```
  for (rp = result; rp != NULL; rp = rp -> ai_next)  {
    cs = socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol);
    reusePort(cs);
    if (cs == -1)  continue;
    if (bind(cs, rp -> ai_addr, rp -> ai_addrlen) == 0) break; //Success
    close(cs);
  }
  freeaddrinfo(result); //No longer needed
  listen(as,4);
  listen(cs,4);
  from_addr_len = sizeof(from_addr);
  FD_ZERO(&readfdsCopy);
  FD_SET(as,&readfdsCopy);
  FD_SET(cs,&readfdsCopy);

  for(;;){
    memcpy(&readfds, &readfdsCopy, sizeof(fd_set));
    n = select(FD_SETSIZE, &readfds, (fd_set *) 0, (fd_set *) 0, NULL);
    if (n > 0) {
      if (FD_ISSET(as, &readfds)) {
        printf("Accepting an Actualizador...\n");
        actSock = accept(as, (struct sockaddr *) &from_addr, &from_addr_len);
        FD_SET(actSock, &readfdsCopy);
      }
      if (FD_ISSET(cs, &readfds)) {
        printf("Accepting a Caja...\n");
        cajaSock = accept(cs, (struct sockaddr *) &from_addr, &from_addr_len);
        write(cajaSock, precio, strlen(precio));
      }
      if (FD_ISSET(actSock, &readfds)) {
        if ((n=read(actSock, precio, sizeof(precio)))>0) {
          precio[n]='\0';
          if (cajaSock > 0) write(cajaSock, precio, strlen(precio));
          printf("precio: %s", precio);
        } else {
          close(actSock);
          FD_CLR(actSock, &readfdsCopy);
        }
      if (FD_ISSET(cajaSock, &readfds))
        if ((n=read(cajaSock, precio, sizeof(precio)))<=0) {
          close(cajaSock);
          FD_CLR(cajaSock, &readfdsCopy);
        }
      }
    }
  }
}
void reusePort(int s) {
  int one=1;
  if ( setsockopt(s,SOL_SOCKET,SO_REUSEADDR,(char *) &one,sizeof(one)) == -1 ) {
    printf("error in setsockopt,SO_REUSEPORT \n");
    exit(-1);
  }
}
```

4 pts. Creación del servicio para el Actualizador
4 pts. Creación del servicio para la Caja
6 pts. Programación de concurrencia (llegada actualización y aceptación de cliente Caja)
4 pts. Aceptación de Autenticador
4 pts. Actualización de precio (no requiere acceso exclusivo)
4 pts. Aceptación de Caja
4 pts. Envío de actualizaciones a Caja

Pregunta 3 (50 pts): Programe en lenguaje Java el cliente Caja.java
Sintaxis: $ java Caja <Servidor precio> <Puerto escucha caja>
Descripción: (**40 pts**) Caja.java crea un **socket TCP** y espera la llegada de un primer precio para el dólar, el cual almacena en una variable double. Luego, Caja.java actualiza este valor cada vez

que llega un nuevo valor desde “Servidor precio”. En concurrencia con lo anterior, Caja.java lee por teclado un número de dólares a comprar y muestra por pantalla el valor en pesos para esa compra. Caja termina cuando se ingresa un número negativo.
(10 pts) Al hacer una venta de dólares, Caja la registra en la base de datos Ventas ya existente en MaríaDB, la cual tiene la tabla Valpo con campos enteros Precio_dolar y Pesos_pagados
Suba a AULA su archivo Caja.java y eventualmente otras clases que usted haya creado.

R. (ver aquí)

```
import java.io.*;
import java.net.*;
import java.util.Scanner;
import java.sql.*;

public class Caja {
   public static void main(String[] args ) {
      try {
         int port = Integer.parseInt(args[1]);
         Socket s = new Socket(args[0], port);
         Scanner in =new Scanner(s.getInputStream());
         Cambio cambio = new Cambio(in.nextDouble());

         System.out.println("Cambio inicial="+cambio.getPrecio());
         Venta venta =new Venta(cambio);
         venta.start();
         while(true)
            cambio.setPrecio(in.nextDouble());
      }
      catch (Exception e) {
         e.printStackTrace();
      }
   }
}
class Cambio {
   private double precio;
   public Cambio (double p){
      precio=p;
   }
   public synchronized double getPrecio(){
      return precio;
   }
   public synchronized void setPrecio(double p){
      precio=p;
   }
}
class Venta extends Thread {
   Cambio cambio;
   Scanner in;
   DB db;

   public Venta(Cambio c) {
      cambio = c;
      in = new Scanner (System.in);
      db = new DB();
   }

   public void run() {
      double amount;
      double total;
      while (true) {
         amount = in.nextDouble();
         if (amount <0) System.exit(0);
         total = amount*cambio.getPrecio();
         System.out.println(total);
         db.storeInDB((int)cambio.getPrecio(), (int) total);
      }
   }
}
```

```
class DB {
   private Statement sentencia;
   public DB() {
      Connection conexion;

      try {  // se carga el Driver JDBC
         Class.forName("org.mariadb.jdbc.Driver");
      } catch( Exception e ) {
         System.out.println( "No se pudo cargar el driver" );
         e.printStackTrace();
         System.exit(-1);
      }
      try {
         conexion = DriverManager.getConnection("jdbc:mariadb://localhost/Ventas","root", "");
         System.out.println( "Conexión establecida" );
         sentencia = conexion.createStatement();
      }catch( Exception e ) { System.out.println( e );}
   }
   public void storeInDB(int precio, int total) {
      try {
         sentencia.executeUpdate( "INSERT INTO Valpo " +
                                  "VALUES("+precio+","+total+")" );
      } catch( Exception e ) {System.out.println( e ); return;  }
    }
}
```

5 pts. Conexión a servidor y recepción de primer valor de cambio
10 pts. Creación de hebra para manejar concurrentemente actualizaciones del precio y cálculo de ventas
5 pts. Programación de la actualización del precio.
10 pts. Manejo de acceso exclusivo al precio del dólar cambiado por una hebra y leído por otra (uso de synchronized)
5 pts. Programación de lectura de cantidad de dólares y valor a pagar
5 pts. Acceso a base de datos
5 pts. Registro en base de datos
5 pts. Condición de término