*Instituto Politécnico Nacional.*

*Escuela Superior de Cómputo.*

*Practica 1*

*Alumnos:* ALLAN ULISES ZEPEDA IBARRA

JONATHAN ARCOS AYALA

*3CM4*

**Objetivo**

El estudiante implementará una aplicación de carrito de compra para la selección y adquisición de artículos, generación de recibo de compra y el envío de múltiples objetos serializados a través de la red haciendo uso de sockets de flujo.

**Introducción**

**Kryonet**

Kryonet es un centro de información de Java que proporciona una API ordenada y comprensible para TCP dinámico y UDC comunicación de red cliente / servidor utilizando NIO. Kryonet utiliza la biblioteca de serialización Kyro a cambiar de forma rápida y significativamente gráficos de objetos en todo el sistema de conexiones. Es un arquetipo para cualquier aplicación cliente / servidor.

Al programar es posible que nos veamos con la necesidad de realizar una tarea cada cierto tiempo. Por ejemplo, si queremos pintar un reloj en pantalla, cada segundo debemos mover o repintar el segundero. Quizás deseemos ver cuándo se crea un fichero, con lo que podemos, por ejemplo, cada diez segundos ver si existe. En fin, hay un montón de posibles aplicaciones en las que podemos necesitar realizar tareas periódicamente.

**Timer**

Java nos proporciona dos clases distintas que nos ayudan a esta tarea. Son las clases **javax.swing.Timer** y **java.util.Timer**. A ambas clases les decimos cada cuánto queremos el aviso (por ejemplo, un aviso cada segundo en el caso del reloj) y ellas se encargan de llamar a un método que nosotros hayamos implementado. El resultado es que ese método (el que pinta el segundero en la pantalla), se llamará cada cierto tiempo (una vez por segundo en el caso del reloj).

**Hibernate**

Hibernate es un potente, objeto de alto rendimiento / persistencia relacional y servicio de consulta. Hibernate le permite desarrollar clases persistentes siguiente lenguaje orientado a objetos - incluyendo la asociación, herencia, polimorfismo, composición y colecciones. Hibernate le permite expresar consultas en su propia extensión de SQL portátil (HQL), así como en SQL nativo, o con un criterio orientado a objetos

**Desarrollo**

Se comenzo por hacer la base de datos y su conexión usando Hibernate Para esto configuramos el servidor que usaremos junto con clases de hibernate y este crea un objeto articulo que es hacia donde mapeara todas las consultas de la base de datos, el objeto que usaremos es:

package HBJ;  
// Generated 22/03/2016 01:58:58 PM by Hibernate Tools 4.3.1  
  
  
  
/\*\*  
 \* Articulo generated by hbm2java  
 \*/  
public class Articulo implements java.io.Serializable {  
  
  
 private String nombre;  
 private Double presio;  
 private Integer cantidad;  
 private String ruta;  
  
 public Articulo() {  
 }  
  
   
 public Articulo(String nombre) {  
 this.nombre = nombre;  
 }  
 public Articulo(String nombre, Double presio, Integer cantidad, String ruta) {  
 this.nombre = nombre;  
 this.presio = presio;  
 this.cantidad = cantidad;  
 this.ruta = ruta;  
 }  
   
 public String getNombre() {  
 return this.nombre;  
 }  
   
 public void setNombre(String nombre) {  
 this.nombre = nombre;  
 }  
 public Double getPresio() {  
 return this.presio;  
 }  
   
 public void setPresio(Double presio) {  
 this.presio = presio;  
 }  
 public Integer getCantidad() {  
 return this.cantidad;  
 }  
   
 public void setCantidad(Integer cantidad) {  
 this.cantidad = cantidad;  
 }  
 public String getRuta() {  
 return this.ruta;  
 }  
   
 public void setRuta(String ruta) {  
 this.ruta = ruta;  
 }  
  
 @Override  
 public String toString()  
 {  
 return "Nombre: " + this.nombre + "\nPresio: " + this.presio + "\nCantidad: " + this.cantidad + "\nRuta: " + this.ruta + "\n\n";  
 }  
  
  
}  
Posteriormente creamos el servidor y su comunicacion TCP y UDP correspondiente. Los servicios que se levantaron son:

Pedir los productos al servidor: En este se hace la peticion a la base de datos y posterriormente los datos se cargan en una lista de objetos articulo para ser enviados al cliente.

public void connected(Connection c)  
 {  
 BufferedImage img;  
 ByteArrayOutputStream baos;  
 System.out.println("Recibiendo conexion de: "+c.getRemoteAddressTCP().getHostName());  
 stockUsuarios.put(c.getID(), new HashMap<String, Integer>());  
 miSesion = HibernateUtilArticulo.getSessionFactory().getCurrentSession();  
 miSesion.beginTransaction();  
 listaArticulo men = new listaArticulo();  
 imagen im = new imagen();  
 Query qr = miSesion.createQuery("SELECT a FROM Articulo a");  
 List<Articulo> articulos = qr.list();  
 men.articulos = articulos;  
 men.bandera=0;  
 try {  
 Thread.sleep(1000);  
 } catch (InterruptedException ex) {  
 Logger.getLogger(servidor.class.getName()).log(Level.SEVERE, null, ex);  
 }  
 c.sendTCP(men);  
 /\*for (Articulo art :articulos) {  
 im.Nombre=art.getNombre();  
 try{  
 img = ImageIO.read(new File(art.getRuta()));  
 baos = new ByteArrayOutputStream();  
 ImageIO.write( img, "jpg", baos );  
 baos.flush();  
 im.img = baos.toByteArray();  
 baos.close();  
 }catch(IOException e){  
 System.out.println(e.getMessage());  
 }   
 c.sendTCP(im);  
 }\*/  
 System.out.println("si envio todo");  
 miSesion.getTransaction().commit();  
 }

Dar de alta elemento del carrito: En este se toma en cuanta los elementos incluidos en el carrito de compras, donde cada unidad de cada articulo incrementa la variable Stock que es la que tiene el control de cuantos elementos estan apartados.

case 0:  
 System.out.println("Opcion 0: Alta de articulo en el carrito");  
 System.out.println(packet.art);  
 aux = (int)stockGeneral.get(packet.art);  
 System.out.println(aux);  
 System.out.println(packet.cantidad);  
 if(aux>=packet.cantidad){  
 stockGeneral.put(packet.art, (aux-packet.cantidad));  
 stockUsuarios.get(c.getID()).put(packet.art, packet.cantidad);  
 packet.art=null;  
 packet.bandera=0;  
 packet.cantidad=0;  
 c.sendTCP(packet);  
   
 }else{  
 packet.art="Existencias insuficientes";  
 packet.bandera=2;  
 packet.cantidad=aux-packet.cantidad;  
 c.sendTCP(packet);  
 }  
 System.out.println(packet.bandera);  
 break;

Quitar elemento del carrito: En el que se disminulle el contado stock y se actualizan las existencias.

case 1:  
 System.out.println("Opcion 1: Baja de articulo en el carrito");  
 aux = (int)stockGeneral.get(packet.art);  
 stockGeneral.put(packet.art, (aux+packet.cantidad));  
 stockUsuarios.get(c.getID()).remove(packet.art);  
 packet.art=null;  
 packet.bandera=1;  
 packet.cantidad=0;  
 c.sendTCP(packet);  
 break;

Cambiar cantidad de elemento en el carrito: en el que se cambia el numero de productos seleccionados en el carrito y se actualizan las existencias del mismo.

case 2:  
 System.out.println("Opcion 2: Cambios de cantidad de articulos en el carrito");  
 aux = (int)stockGeneral.get(packet.art);  
 int auxStockCli = ((int)stockUsuarios.get(c.getID()).get(packet.art) - packet.cantidad);  
 if(auxStockCli<0){  
 if(aux>=packet.cantidad){  
 stockGeneral.put(packet.art, (aux+auxStockCli));  
 stockUsuarios.get(c.getID()).put(packet.art, packet.cantidad);  
 packet.art=null;  
 packet.bandera=3;  
 packet.cantidad=0;  
 c.sendTCP(packet);  
 }else{  
 packet.art="Existencias insuficientes";  
 packet.bandera=2;  
 packet.cantidad=aux-packet.cantidad;  
 c.sendTCP(packet);  
 }  
 }else{  
 stockGeneral.put(packet.art, (aux+packet.cantidad));  
 stockUsuarios.get(c.getID()).put(packet.art,packet.cantidad);  
 packet.art=null;  
 packet.bandera=3;  
 packet.cantidad=0;  
 c.sendTCP(packet);  
 }  
 break;

Confirmar compra: en el que todos los elementos del carrito son confirmados para comprar y estos son descontados directamente de la base de datos.

case 3:  
 System.out.println("Opcion 3: Confirmar Compra");  
 boolean compra=true;  
 while (compra){  
 if(permiso==null){  
 permiso=c;  
 Map stock = stockUsuarios.get(c.getID());  
 miSesion = HibernateUtilArticulo.getSessionFactory().getCurrentSession();  
 miSesion.beginTransaction();  
   
 for(Articulo art :server.articulos){  
 if(stock.containsKey(art.getNombre())){  
 art.setCantidad((art.getCantidad())-(int)stock.get(art.getNombre()));  
 miSesion.update(art);  
 }  
 }  
 miSesion.getTransaction().commit();  
 stockUsuarios.put(c.getID(), new HashMap<String, Integer>());  
 packet.art=null;  
 packet.bandera=4;  
 packet.cantidad=0;  
 c.sendTCP(packet);  
 permiso=null;  
 compra=false;  
 }  
 }  
   
 break;

Para finalizar se impementaron todas las peticiones del cliente de cada uno de estos servicios.

/\*  
 \* To change this license header, choose License Headers in Project Properties.  
 \* To change this template file, choose Tools | Templates  
 \* and open the template in the editor.  
 \*/  
package ver;  
import com.esotericsoftware.kryo.Kryo;  
import com.esotericsoftware.kryonet.\*;  
import java.io.IOException;  
import javax.swing.JOptionPane;  
  
/\*\*  
 \*  
 \* @author allan  
 \*/  
public class clie extends Listener{  
 Client clienteKryo = null;  
 Kryo kryo;  
 String cont=null;  
 int tcp=3041;  
 int udp=3042;  
   
 clie(){  
 clienteKryo = new Client();  
 kryo = clienteKryo.getKryo();   
 kryo.register(listaArticulo.class);  
 kryo.register(articuloUnitario.class);  
 kryo.register(java.util.ArrayList.class);  
 kryo.register(Articulo.class);  
 kryo.register(imagen.class);  
 kryo.register(byte[].class);  
 }  
  
 clie(String ip) throws IOException{  
 clienteKryo = new Client();  
 kryo = clienteKryo.getKryo();   
 kryo.register(listaArticulo.class);  
 kryo.register(articuloUnitario.class);  
 kryo.register(java.util.ArrayList.class);  
 kryo.register(Articulo.class);  
 kryo.register(imagen.class);  
 kryo.register(byte[].class);  
 clienteKryo.start();  
 clienteKryo.connect(5000, ip , tcp, udp);  
 clienteKryo.addListener(new clie());  
 }  
 public void crear\_Socket(String ip) throws IOException  
 {  
 clienteKryo.start();  
 clienteKryo.connect(5000, ip , tcp, udp);  
 clienteKryo.addListener(new clie());  
 }  
 @Override  
 public void disconnected(Connection c)  
 {  
 JOptionPane.showMessageDialog(null, "Error de Conexion");  
 }  
 @Override  
 public void received(Connection c,Object p)  
 {  
 System.out.println(c);  
 if(p instanceof listaArticulo)  
 {  
 listaArticulo packet = (listaArticulo) p;  
 System.out.println(packet.articulos);  
 switch(packet.bandera){  
 case 0:  
 cliente.articulos = packet.articulos;  
 cliente.cargarCompleta=false;  
 break;  
 default:  
 break;  
 }  
 }else if(p instanceof articuloUnitario){  
 articuloUnitario packet = (articuloUnitario) p;  
 switch(packet.bandera){  
 case 0:  
 cliente.altaCompleta=false;  
 break;  
 case 1:  
 cliente.bajaCompleta=false;  
 break;  
 case 2:  
 System.out.println("Error en la peticion");  
 cliente.e = new Exception(packet.art);  
 cliente.altaCompleta=false;  
 cliente.cambioCompleta=false;  
 case 3:  
 cliente.cambioCompleta=false;  
 break;  
 case 4:  
 cliente.compraCompleta=false;  
 break;  
 case 5:  
 cliente.borrarCompleta=false;  
 break;  
 default:  
 break;  
 }  
 }  
 }  
}

**Conclusiones**

**Arcos Ayala Jonathan:** En un carrito de compra se comparte información entre varios usuarios y se debe de llevar un control de los datos ya que se puede duplicar las peticiones por mas de un usuario al mismo tiempo y tenemos que ser capases de controlar el flujo de información para que no se genere una incongruencia de datos. Lo que encontramos en esta practica es el control de la existencias por medio de un servidor que controla los registros a la base de datos y podemos ver como se bloquean los servicios al momento de la compra.

**Zepeda Ibarra Allan Ulises:** Al administrar varias peticiones al mismo tiempo es necesario el uso de sockets bloqueantes para conservar la consistencia de la información, ya que si se procesa indiscriminadamente las peticiones podemos caes como es el caso de una tienda en línea en vender un producto del que ya no tenemos existencias.

**Bibliografía**

<http://www.kryonet.com/>

<http://www.chuidiang.com/java/timer/timer.php>

<http://facestutorials.icefaces.org/tutorial/hibernate-tutorial.html>