



PRACTICA NO.1 CARRITO DE COMPRA

Ares Ulises Juárez Martínez
3CV17



15 DE MAYO DE 2022
SANDRA IVETTE BAUTISTA ROSALES
Escuela Superior de Computo

Contenido

Objetivo	2
Introducción	2
Cuando un objeto Serializable se escribe con writeObject, luego se modifica y se escribe por segunda vez, ¿por qué falta la modificación cuando se deserializa la transmisión?	2
Apache PDFBox® - A Java PDF Library	2
Class Desktop	2
Building Java Projects with Maven	2
Desarrollo	3
Cuestionario	17
Conclusiones	18
Referencias	18

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 bloqueantes.

Introducción

Cuando un objeto `Serializable` se escribe con `writeObject`, luego se modifica y se escribe por segunda vez, ¿por qué falta la modificación cuando se deserializa la transmisión?

La clase realiza un seguimiento de cada objeto que serializa y envía solo el identificador si el objeto se escribe en la secuencia una vez posterior. Esta es la forma en que trata con gráficos de objetos. El correspondiente realiza un seguimiento de todos los objetos que ha creado y sus identificadores, de modo que cuando el identificador se vuelve a ver, puede devolver el mismo objeto. Tanto los flujos de salida como los de entrada mantienen este estado hasta que se liberan.

Alternativamente, la clase implementa un método de reinicio que descarta la memoria de haber enviado un objeto, por lo que enviar un objeto nuevamente hará una copia. [1]

Apache PDFBox® - A Java PDF Library

La biblioteca Apache PDFBox® es una herramienta Java de código abierto para trabajar con documentos PDF. Este proyecto permite la creación de nuevos documentos PDF, la manipulación de documentos existentes y la capacidad de extraer contenido de los documentos. Apache PDFBox también incluye varias utilidades de línea de comandos. Apache PDFBox se publica bajo la Licencia Apache v2.0. [2]

Class Desktop

La clase `Desktop` permite que una aplicación Java inicie aplicaciones asociadas registradas en el escritorio nativo para manejar un URI o un archivo.

Las operaciones admitidas incluyen:

- iniciar el navegador predeterminado del usuario para mostrar un URI específico;
- iniciar el cliente de correo predeterminado del usuario con un `mailto` URI opcional;
- iniciar una aplicación registrada para abrir, editar o imprimir un archivo específico.

Esta clase proporciona métodos correspondientes a estas operaciones. Los métodos buscan la aplicación asociada registrada en la plataforma actual y la inician para manejar un URI o archivo. Si no hay una aplicación asociada o la aplicación asociada no se inicia, se genera una excepción. [3]

Building Java Projects with Maven

Apache Maven es una herramienta de comprensión y gestión de proyectos de software. Basado en el concepto de un modelo de objetos de proyecto (POM), Maven puede administrar la construcción, los informes y la documentación de un proyecto desde una pieza central de información. [4]

Desarrollo

Se comenzó programando la clase de productos, definiendo sus atributos e implementando la interfaz de Externizable

```
1 public class Producto implements Externalizable { // Usamos Externalizable para poder escribir manualmente los elementos
2 // del objeto
3     private int idProducto;
4     private String nombre;
5     private int cantidad;
6     private double precio;
7     private String descripcion;
8
9     public Producto(int idProducto, String nombre,
10         int cantidad, double precio, String descripcion) { // Constructor con todos los objetos
11         this.idProducto = idProducto;
12         this.nombre = nombre;
13         this.cantidad = cantidad;
14         this.precio = precio;
15         this.descripcion = descripcion;
16     }
17 }
```

Código 1. Segmento de la definición de Producto

Ahora se procede a definir un método para obtener las imágenes de su producto

```
1 public ArrayList<File> obtenerImagenes() { // Metodo para obtener las imagenes de cada producto, esta fijo a la
2 // cantidad de productos
3     ArrayList<File> imagenes = new ArrayList<File>();
4     Path path = Paths.get("./Imagenes/");
5     File a1, a2, a3, a4;
6     if (Files.exists(path)) {
7         a1 = new File("./Imagenes/" + idProducto + "_1.png");
8         a2 = new File("./Imagenes/" + idProducto + "_2.png");
9         a3 = new File("./Imagenes/" + idProducto + "_3.png");
10        a4 = new File("./Imagenes/" + idProducto + "_4.png");
11    } else {
12        a1 = new File("./" + idProducto + "_1.png");
13        a2 = new File("./" + idProducto + "_2.png");
14        a3 = new File("./" + idProducto + "_3.png");
15        a4 = new File("./" + idProducto + "_4.png");
16    }
17    imagenes.add(a1);
18    imagenes.add(a2);
19    imagenes.add(a3);
20    imagenes.add(a4);
21    return imagenes;
22 }
```

Código 2 Segmento del producto para obtener las imágenes

Después se implementaron los métodos para poder realizar la serialización del producto.

```

1  @Override
2  public void writeExternal(ObjectOutput out) throws IOException { // Metodo para escribir el objeto
3      out.writeInt(idProducto);
4      out.writeUTF(nombre);
5      out.writeUTF(descripcion);
6      out.writeInt(cantidad);
7      out.writeDouble(precio);
8  }
9
10
11  @Override
12  public void readExternal(ObjectInput in)
13      throws IOException, ClassNotFoundException { // Metodo para leer el objeto
14      this.idProducto = in.readInt();
15      this.nombre = in.readUTF();
16      this.descripcion = in.readUTF();
17      this.cantidad = in.readInt();
18      this.precio = in.readDouble();
19  }
20

```

Código 3 Segmento de código de serialización de Productos

A continuación, se realizó la clase de CarritoDeCompra, con sus atributos y de igual manera implementando la interfaz de Externalizable, el cual va a tener un ArrayList con los productos

```

1  public class CarritoDeCompra implements Externalizable { // Usamos Externalizable para poder escribir el ArrayList
2
3      private ArrayList<Producto> productos;
4      private double total;
5
6      public CarritoDeCompra() { // Constructor default
7          productos = new ArrayList<Producto>();
8          total = 0.0;
9      }

```

Código 4 Segmento de código del CarritoDeCompra

Y después se tiene el código de generación de ticket, utilizando la biblioteca de PDFBox, el cual solo genera una página simple de ticket.

```

1 public void generateTicket() throws IOException { // Metodo que genera el ticket del carrito de compra
2     PDDocument document = new PDDocument(); // Creamos un objeto del PDF
3     PDPage pagina = new PDPage(); // Creamos una pagina que agregar al PDF
4     document.addPage(pagina); // Lo agregamos
5     PDPageContentStream contentStream = new PDPageContentStream(
6         document, pagina); // Creamos un stream para la pagina
7     contentStream.beginText(); // Decimos que vamos a empezar a escribir
8     contentStream.setFont(PDType1Font.COURIER, 10); // Configuramos la tipografia y el tamaño de letra
9     contentStream.setLeading(14.5f); // Configuramos el leading
10    contentStream.newLineAtOffset(25, 700); // Le decimos donde vamos a empezar a escribir el texto
11    SimpleDateFormat dsf = new SimpleDateFormat(
12        "EEEE dd MMM yyyy", Locale.getDefault()); // Creamos un DateFormat, solo para el ticket
13    contentStream.showText("Fecha: " + dsf.format(new Date())); // Escribimos la fecha
14    contentStream.newLine(); // Nueva linea
15    contentStream.showText("Producto      Cantidad      Precio      SubTotal"); // Escribimos las cabeceras
16    contentStream.newLine(); // Nueva linea
17    for (Iterator it = productos.iterator(); it.hasNext(); ) { // Iteramos todos los productos
18        Producto p = (Producto) it.next();
19        contentStream.showText(
20            p.getNombre()
21                .substring(0, Math.min(12, p.getNombre().length())) // Ponemos el titulo del producto o solo
22                                                                    // 12 caracteres
23            + " "
24            + p.getCantidad() + " " + p.getPrecio() + " " // Mostramos la cantidad y el precio
25            + (p.getCantidad() * p.getPrecio()); // Y el subtotal del producto
26        contentStream.newLine(); // Nueva linea
27    }
28    contentStream.showText("Total: " + total); // Total
29    contentStream.endText(); // Finalizamos el texto
30    contentStream.close(); // Cerramos el stream
31    dsf = new SimpleDateFormat("yyyy-MM-dd'T'HH-mm-ss"); // Creamos el nuevo formato para guardar el archivo
32    String ruta = "Ticket" + dsf.format(new Date()) + ".pdf"; // Nombre del ticket
33    document.save(ruta); // Lo guardamos
34    document.close(); // Cerramos el documento
35    if (Desktop.isDesktopSupported()) { // Si Desktop esta disponible
36        Desktop.getDesktop().open(new File(ruta)); // Abrimos el PDF
37    }
38
39 }

```

Código 5 Segmento de código que genera tickets

Y después los métodos que modifican los productos del carrito

```

1 public void addProduct(Producto nuevoProducto) { // Agregamos un producto al carrito de compra
2     boolean seActualizo = false;
3     for (int i = 0; i < productos.size(); i++) {
4         Producto p = (Producto) productos.get(i);
5         if (nuevoProducto.getIdProducto() == p.getIdProducto()) { // Checamos si ya esta registrado
6             p.setCantidad(p.getCantidad() + nuevoProducto.getCantidad()); // Actualizamos la cantidad
7             seActualizo = true;
8             break;
9         }
10    }
11    if (!seActualizo) { // Si no lo encontro
12        productos.add(nuevoProducto); // Lo agregamos al ArrayList
13    }
14    actualizaTotal(); // Actualizamos el total
15 }

```

Código 6 Segmento de código que agrega un producto al carrito

```

1 public void removeProduct(int idProducto, boolean reIndex) { // Quitamos el producto del IDProducto y le pasamos si
2 // debemos reindexar
3     for (int i = 0; i < productos.size(); i++) { // Iteramos todos los productos
4         Producto get = (Producto) productos.get(i);
5         if (get.getIdProducto() == idProducto) { // Si el producto es el mismo
6             productos.remove(get); // Lo eliminamos
7             break;
8         }
9     }
10    if (reIndex) // SI hay que reindexar
11        for (int i = 0; i < productos.size(); i++) {
12            Producto get = (Producto) productos.get(i);
13            get.setIdProducto(i + 1); // Cambiamos el ID
14        }
15    actualizaTotal(); // Actualizamos el total
16 }

```

Código 7 Segmento de código que elimina un producto del Carrito

```

1 private void actualizaTotal() { // Metodo para actualizar el total del carrito, iterando todos los elementos
2     total = 0.0;
3     for (Iterator iterator = productos.iterator(); iterator.hasNext(); ) {
4         Producto p = (Producto) iterator.next();
5         total += p.getCantidad() * p.getPrecio();
6     }
7 }

```

Código 8 Segmento de código que actualiza el total del carrito

```

1 public Producto getProducto(int idProducto) { // Obtenemos el producto dado el ID
2     Producto p = null;
3     for (int i = 0; i < productos.size(); i++) {
4         Producto get = (Producto) productos.get(i);
5         if (get.getIdProducto() == idProducto) { // SI el producto es igual al ID
6             p = get;
7             break;
8         }
9     }
10    return p; // Lo regresamos
11 }

```

Código 9 Segmento de código que obtiene un producto del Carrito

Y por último los métodos para serializar el objeto, donde se hace una distinción debido al ArrayList

```

1  @Override
2  public void writeExternal(ObjectOutput out) throws IOException { // Metodo para escribir el objeto
3      out.writeDouble(total);
4      out.writeInt(productos.size());
5      for (Producto producto : productos) {
6          out.writeObject(producto);
7      }
8  }
9  }
10
11 @Override
12 public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException { // Metodo para leer el objeto
13     this.total = in.readDouble();
14     int cantidad = in.readInt();
15     productos = new ArrayList<Producto>();
16     for (int i = 0; i < cantidad; i++) {
17         productos.add((Producto) in.readObject());
18     }
19 }

```

Código 10 Segmento de código que serializa el objeto

Para la gestión de las peticiones y comunicaciones entre el cliente y servidor, se realizó un gestor, donde existen 4 métodos

- Para gestionar el cliente desde el cliente
- Para gestionar el cliente desde el servidor
- Para gestionar el administrador desde el cliente
- Para gestionar el administrador desde el servidor


```

1 public void gestionaCliente(ObjectInputStream ois, ObjectOutputStream oos, BufferedReader br)
2     throws IOException, ClassNotFoundException, CloneNotSupportedException { // Metodo para gestionar el cliente
3     // desde el cliente
4     if (ois.readBoolean()) {
5         CarritoDeCompra existencias = (CarritoDeCompra) ois.readObject(); // Obtenemos el carrito actual
6         CarritoDeCompra carr = new CarritoDeCompra();
7         boolean sePuedeSalir = false;
8         Producto p, copiaProducto;
9         int IdProducto;
10        while (!sePuedeSalir) {
11            System.out.println(" " +
12                "Ingresa la operacion que quieras realizar:\n" +
13                "1)Revisar carrito de compras\n" +
14                "2)Listar productos en tienda\n" +
15                "3)Mostrar detalles del producto\n" +
16                "4)Comprar producto\n" +
17                "5)Modificar carrito\n" +
18                "6)Finalizar compra\n" +
19                "7)Salir\n" +
20                "");
21
22            int opcion = Integer.parseInt(br.readLine());
23            switch (opcion) {
24                case 1: // Revisar carrito
25                    System.out.println(carr);
26                    break;
27                case 2:// Listar productos en tienda
28                    oos.writeInt(4);
29                    oos.flush();
30                    oos.reset(); // Hago un reset para que no tenga cache de objetos
31                    existencias = (CarritoDeCompra) ois.readObject(); // Revisamos las existencias de productos
32                    for (Iterator<Producto> iterator = existencias.getProductos().iterator(); iterator.hasNext();) {
33                        p = iterator.next();
34                        System.out.println("ID: " + p.getIdProducto() + "\tNombre: " + p.getNombre()
35                            + "\tExistencias: " + p.getCantidad());
36                    }
37                    break;
38                case 3:// Mostrar detalles del producto
39                    oos.writeInt(1);
40                    oos.flush();
41                    oos.reset(); // Hago un reset para que no tenga cache de objetos
42                    System.out.print("Ingresa el ID del producto: ");
43
44                    IdProducto = Integer.parseInt(br.readLine());
45                    oos.writeInt(IdProducto); // Enviamos el ID del producto
46                    oos.flush();
47                    oos.reset(); // Hago un reset para que no tenga cache de objetos
48                    p = (Producto) ois.readObject(); // Obtenemos el producto
49                    System.out.println(p);
50
51                    int cantidad = ois.readInt(); // Cantidad de imagenes del producto
52                    byte[] b = new byte[1024]; // Buffer para escribir la imagen
53                    for (int i = 0; i < cantidad; i++) {
54                        int n;
55                        long recibidos = 0, tam = ois.readLong(); // Leemos el tamaño del archivo
56                        File archivo = new File("Producto_" + (i + 1) + ".png"); // Creamos un archivo
57                        try (DataOutputStream dos = new DataOutputStream(
58                            new FileOutputStream(archivo))) {
59                            while (recibidos < tam) { // Escribimos la imagen mientras tengamos que recibir
60                                n = ois.read(b);
61                                dos.write(b, 0, n);
62                                recibidos += n;
63                            }
64                        }
65                        muestraImagen(archivo); // Mostramos la imagen
66                    }
67                    break;

```

Código 11. Segmento de código que gestiona el cliente desde el cliente, parte 1

```

1  case 4:// Comprar producto
2      System.out.print("Ingresa el ID del producto: ");
3
4      IdProducto = Integer.parseInt(br.readLine());
5      p = carr.getProducto(IdProducto); // Obtenemos el producto para revisar si ya esta en el carrito
6      if (p == null) { // Si no existe entonces hay que tomarlo de las existencias
7          p = existencias.getProducto(IdProducto); // Lo buscamos en las existencias
8          if (!p == null) { // Si lo encontro entonces lo clonamos
9              p = (Producto) p.clone(); // Lo clonamos
10             p.setCantidad(0); // Ponemos la cantidad de 0 para que lo cambiemos despues
11         }
12     }
13     if (!p == null) { // Si el producto es diferente de null
14         copiaProducto = (Producto) p.clone(); // Lo clonamos
15         System.out.print("Ingresa la cantidad del producto: ");
16         copiaProducto.setCantidad(Integer.parseInt(br.readLine())); // Ponemos la cantidad desde el
17                                     // teclado
18         oos.writeInt(2); // Le decimos que queremos comprar un objeto
19         oos.flush();
20         oos.reset(); // Hago un reset para que no tenga cache de objetos
21         copiaProducto.setCantidad(p.getCantidad() + copiaProducto.getCantidad()); // Lo agregamos a
22                                     // la copia
23         oos.writeObject(copiaProducto); // Lo enviamos
24         oos.flush();
25         oos.reset(); // Hago un reset para que no tenga cache de objetos
26         if (ois.readBoolean()) { // Significa que si hay existencias suficientes
27             copiaProducto.setCantidad(copiaProducto.getCantidad() - p.getCantidad());
28             carr.addProducto(copiaProducto); // Lo agregamos al carrito
29             System.out.println("Articulo agregado al carrito");
30
31         } else { // No hay existencias
32             System.out.println("No hay suficientes existencias");
33         }
34     } else {
35         System.out.println("El producto no existe");
36     }
37     break;
38 case 5:// Modificar carrito
39     boolean sePuedeSalir2 = false;
40     while (!sePuedeSalir2) { // Entramos en el ciclo para modificar el carrito
41         System.out.println("" +
42             "Ingresa la operacion que quieras realizar:\n" +
43             "1)Mostrar carrito\n" +
44             "2)Modificar cantidad de un producto\n" +
45             "3)Eliminar producto\n" +
46             "4)Salir\n" +
47             "");
48
49         int opcion2 = Integer.parseInt(br.readLine());
50         switch (opcion2) {
51             case 1:// Mostrar carrito
52                 System.out.println(carr);
53
54                 break;
55             case 2:// Modificar cantidad de un producto
56                 System.out.print("Ingresa el ID del producto: ");
57
58                 IdProducto = Integer.parseInt(br.readLine());
59                 p = carr.getProducto(IdProducto); // Buscamos el producto que queremos modificar
60
61                 if (!p == null) { // Si el objeto es diferente de nulo
62                     p = (Producto) p.clone();
63                     System.out.print("Ingresa la cantidad del producto: ");
64

```

Código 12 Segmento de código que gestiona el cliente desde el cliente, parte 2

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73

```

```

        p.setCantidad(Integer.parseInt(br.readLine()));
        oos.writeInt(2);
        oos.flush();
        oos.reset(); // Hago un reset para que no tenga cache de objetos
        oos.writeObject(p);
        oos.flush();
        oos.reset(); // Hago un reset para que no tenga cache de objetos
        if (ois.readBoolean()) { // Si hay existencias
            carr.removeProduct(p.getIdProducto(), false); // Lo borramos
            carr.addProduct(p); // Y lo agregamos
            System.out.println("Carrito modificado correctamente");
        } else {
            System.out.println("No hay suficientes existencias");
        }
    } else {
        System.out.println("El producto no esta en el carrito");
    }
    break;
case 3: // Elimar producto
    System.out.print("Ingresa el ID del producto: ");

    IdProducto = Integer.parseInt(br.readLine());
    p = carr.getProducto(IdProducto); // Lo buscamos en el carrito
    if (p != null) {
        carr.removeProduct(IdProducto, false); // Lo quitamos
        System.out.println("Producto eliminado del carrito");
    } else {
        System.out.println("El producto no esta en el carrito");
    }
    break;
case 4: // Salir
    sePuedeSalir2 = true;
    break;
default:
    System.out.println("Opcion invalida");
    break;
    }
    break;
case 6: // Finalizar compra
    oos.writeInt(3);
    oos.flush();
    oos.reset(); // Hago un reset para que no tenga cache de objetos
    oos.writeObject(carr);
    oos.flush();
    oos.reset(); // Hago un reset para que no tenga cache de objetos
    if (ois.readBoolean()) { // Si paso la compra
        carr.generateTicket(); // Generamos el ticket
        carr = new CarritoDeCompra(); // Creamos un nuevo carrito de compra
        System.out.println("Compra exitosa");
    } else {
        System.out.println("Hubo un error con la compra");
    }
    break;
case 7:
    // Finalizar compra
    sePuedeSalir = true;
    oos.writeInt(5);
    oos.flush();
    oos.reset(); // Hago un reset para que no tenga cache de objetos
    break;
default:
    System.out.println("Opcion invalida");
    break;
    }
    oos.reset(); // Hago un reset para que no tenga cache de objetos
    }
} else {
    System.out.println("Pues no puedes comprar gg");
}
}

```

Código 13 Segmento de código que gestiona el cliente desde el cliente, parte 3

```
1 public void gestionaAbastecedorCliente(ObjectInputStream ois, ObjectOutputStream oos, BufferedReader br)
2     throws IOException, ClassNotFoundException { // Metodo para gestionar el abastecedor desde el cliente
3     System.out.print("Ingresa la contraseña: ");
4     String pass = br.readLine().trim();
5     oos.writeUTF(pass); // Enviamos la contraseña
6     oos.flush();
7     oos.reset(); // Hago un reset para que no tenga cache de objetos
8     if (ois.readBoolean()) { // Si lo autorizamos
9         boolean sePuedeSalir = false;
10        Producto p;
11        CarritoDeCompra carr = (CarritoDeCompra) ois.readObject(); // Obtenemos las existencias
12        boolean deboLeer = false;
13        while (!sePuedeSalir) {
14            System.out.println("Ingresa la operacion que quieres realizar:\n" +
15                "1)Agregar productos\n" +
16                "2)Actualiza un producto\n" +
17                "3)Quita un producto\n" +
18                "4)Mostrar productos\n" +
19                "5)Salir\n" +
20                "");
21            int operacion = Integer.parseInt(br.readLine().trim());
22            switch (operacion) {
23                case 1: // Registrar producto
24                    oos.writeInt(1);
25                    oos.flush();
26                    oos.reset(); // Hago un reset para que no tenga cache de objetos
27                    p = new Producto();
28                    System.out.print("Ingresa el nombre del producto: ");
29                    p.setNombre(br.readLine().trim().replaceAll("\\u001B\\[[\\d;]*[^\d;]", ""));
30                    System.out.print("Ingresa la descripcion del producto: ");
31                    p.setDescripcion(br.readLine().trim().replaceAll("\\u001B\\[[\\d;]*[^\d;]", ""));
32                    System.out.print("Ingresa la cantidad de articulos que habra: ");
33                    p.setCantidad(Integer.parseInt(br.readLine().trim()));
34                    System.out.print("Ingresa el precio: ");
35                    p.setPrecio(Double.parseDouble(br.readLine().trim()));
36                    oos.writeUTF(p.getNombre());
37                    oos.flush();
38                    oos.reset(); // Hago un reset para que no tenga cache de objetos
39                    oos.writeInt(p.getCantidad());
40                    oos.flush();
41                    oos.reset(); // Hago un reset para que no tenga cache de objetos
42                    oos.writeDouble(p.getPrecio());
43                    oos.flush();
44                    oos.reset(); // Hago un reset para que no tenga cache de objetos
45                    oos.writeUTF(p.getDescripcion());
46                    oos.flush();
47                    oos.reset(); // Hago un reset para que no tenga cache de objetos
48                    System.out.println("Producto registrado");
49                    deboLeer = true;
50                    break;
```

Código 14 Segmento de código que gestiona el abastecedor desde el cliente, parte 1

```

1  case 2: // Modificar producto
2      System.out.print("Ingresa el ID del producto: ");
3      p = carr.getProducto(Integer.parseInt(br.readLine().trim())); // Buscamos el producto en las
4      // existencias
5      if (p != null) { // Si el producto es diferente de nulo
6          oos.writeInt(2);
7          oos.flush();
8          oos.reset(); // Hago un reset para que no tenga cache de objetos
9          System.out.print("Ingresa el nombre del producto: ");
10         p.setNombre(br.readLine().trim().replaceAll("\\u001B\\[[\\d;]*[\\^\\d;]", ""));
11         System.out.print("Ingresa la descripción del producto: ");
12         p.setDescripcion(br.readLine().trim().replaceAll("\\u001B\\[[\\d;]*[\\^\\d;]", ""));
13         System.out.print("Ingresa la cantidad de artículos que habrá: ");
14         p.setCantidad(Integer.parseInt(br.readLine().trim()));
15         System.out.print("Ingresa el precio: ");
16         p.setPrecio(Double.parseDouble(br.readLine().trim()));
17         oos.writeInt(p.getIdProducto());
18         oos.flush();
19         oos.reset(); // Hago un reset para que no tenga cache de objetos
20         oos.writeUTF(p.getNombre());
21         oos.flush();
22         oos.reset(); // Hago un reset para que no tenga cache de objetos
23         oos.writeInt(p.getCantidad());
24         oos.flush();
25         oos.reset(); // Hago un reset para que no tenga cache de objetos
26         oos.writeDouble(p.getPrecio());
27         oos.flush();
28         oos.reset(); // Hago un reset para que no tenga cache de objetos
29         oos.writeUTF(p.getDescripcion());
30         oos.flush();
31         oos.reset(); // Hago un reset para que no tenga cache de objetos
32         System.out.println("Producto actualizado");
33     } else {
34         System.out.println("ID inválido");
35     }
36     deboLeer = true;
37     break;
38 case 3: // Borrar producto
39     System.out.println("Ingresa el ID del producto");
40     p = carr.getProducto(Integer.parseInt(br.readLine().trim())); // Buscamos el producto
41     if (p != null) { // Si el producto es diferente de nulo
42         oos.writeInt(3);
43         oos.writeInt(p.getIdProducto());
44         oos.flush();
45         oos.reset(); // Hago un reset para que no tenga cache de objetos
46         System.out.println("Producto eliminado");
47     } else {
48         System.out.println("ID inválido");
49     }
50     deboLeer = true;
51     break;
52 case 4: // Mostramos las existencias
53     System.out.println(carr);
54     deboLeer = false;
55     break;
56 case 5: // Salir del ciclo
57     sePuedeSalir = true;
58     oos.writeInt(4);
59     oos.flush();
60     oos.reset(); // Hago un reset para que no tenga cache de objetos
61     deboLeer = true;
62     break;
63 default: // Opción inválida
64     deboLeer = false;
65     System.out.println("Opción inválida");
66     break;
67 }
68 if (deboLeer) {
69     carr = (CarritoDeCompra) ois.readObject();
70 }
71 oos.reset(); // Hago un reset para que no tenga cache de objetos
72 }
73 }
74 } else {
75     System.out.println("Contraseña incorrecta");
76 }
77 }

```

Código 15 Segmento de código que gestiona el abastecedor desde el cliente, parte 2


```

1 public void gestionaClienteServidor(ObjectInputStream ois, ObjectOutputStream oos)
2     throws IOException, ClassNotFoundException, CloneNotSupportedException {
3     CarritoDeCompra carr = getCarritoCompra();
4     Producto p;
5     int idProducto;
6     oos.writeObject(carr);
7     oos.flush();
8     oos.reset(); // Hago un reset para que no tenga cache de objetos
9     boolean sePuedeSalir = false;
10    while (!sePuedeSalir) {
11        int operacion = ois.readInt();
12        switch (operacion) {
13            case 1: // Mostrar detalles del producto
14                idProducto = ois.readInt();
15                p = carr.getProducto(idProducto);
16                oos.writeObject(p);
17                oos.flush();
18                oos.reset(); // Hago un reset para que no tenga cache de objetos
19                ArrayList<File> archivos = p.obtenerImagenes();
20                oos.writeInt(archivos.size());
21                oos.flush();
22                oos.reset(); // Hago un reset para que no tenga cache de objetos
23                byte[] b = new byte[1024];
24                for (File file : archivos) {
25                    int n;
26                    long enviados = 0, tam = file.length();
27                    oos.writeLong(tam);
28                    oos.flush();
29                    oos.reset(); // Hago un reset para que no tenga cache de objetos
30                    try (DataInputStream dis = new DataInputStream(new FileInputStream(file))) {
31                        while (enviados < tam) {
32                            n = dis.read(b);
33                            oos.write(b, 0, n);
34                            oos.flush();
35                            oos.reset(); // Hago un reset para que no tenga cache de objetos
36                            enviados += (long) n;
37                        }
38                    }
39                }
40                break;
41            case 2: // Comprobar existencias
42                p = (Producto) ois.readObject();
43                if (carr.getProducto(p.getIdProducto()).getCantidad() >= p.getCantidad()) {
44                    oos.writeBoolean(true);
45                    oos.flush();
46                    oos.reset(); // Hago un reset para que no tenga cache de objetos
47                } else {
48                    oos.writeBoolean(false);
49                    oos.flush();
50                    oos.reset(); // Hago un reset para que no tenga cache de objetos
51                }
52                break;
53            case 3: // Comprar nuevo carrito
54                CarritoDeCompra nuevo = (CarritoDeCompra) ois.readObject();
55                CarritoDeCompra propuesta = (CarritoDeCompra) carr.clone();
56                boolean siCompro = true;
57                for (Iterator<Producto> iterator = nuevo.getProductos().iterator(); iterator.hasNext();) {
58                    p = iterator.next();
59                    if (carr.getProducto(p.getIdProducto()).getCantidad() < p.getCantidad()) {
60                        oos.writeBoolean(false);
61                        oos.flush();
62                        oos.reset(); // Hago un reset para que no tenga cache de objetos
63                        siCompro = false;
64                        break;
65                    } else {
66                        Producto actualiza = (Producto) carr.getProducto(p.getIdProducto()).clone();
67                        actualiza.setCantidad(-p.getCantidad());
68                        propuesta.addProduct(actualiza);
69                    }
70                }
71                if (siCompro) {
72                    carr.setProductos(propuesta.getProductos());
73                    carr.setTotal(0.0);
74
75                    oos.writeBoolean(true);
76                    oos.flush();
77                    oos.reset(); // Hago un reset para que no tenga cache de objetos
78                } else {
79                    oos.writeBoolean(false);
80                    oos.flush();
81                    oos.reset(); // Hago un reset para que no tenga cache de objetos
82                }
83                break;
84            case 4: // Devuelve productos
85                oos.writeObject(carr);
86                oos.flush();
87                oos.reset(); // Hago un reset para que no tenga cache de objetos
88                break;
89            case 5:
90                sePuedeSalir = true;
91                break;
92        }
93        oos.reset(); // Hago un reset para que no tenga cache de objetos
94    }
95
96    saveCarritoCompra(carr);
97 }

```

Código 16 Segmento de código que gestiona el cliente desde el servidor

```
1 public void gestionaAbastecedorServidor(ObjectInputStream ois, ObjectOutputStream oos)
2     throws IOException { // Manejamos el abastecedor desde el servidor
3     CarritoDeCompra carr = getCarritoCompra();
4     Producto p;
5     oos.writeObject(carr);
6     oos.flush();
7     oos.reset(); // Hago un reset para que no tenga cache de objetos
8     boolean sePuedeSalir = false;
9     while (!sePuedeSalir) { // Ciclamos mientras no se deba salir
10        int operacion = ois.readInt();
11        switch (operacion) {
12            case 1: // Agrega producto
13                p = new Producto(); // Recibimos las propiedades del nuevo producto
14                p.setIdProducto(carr.getProductos().size() + 1);
15                p.setNombre(ois.readUTF());
16                p.setCantidad(ois.readInt());
17                p.setPrecio(ois.readDouble());
18                p.setDescripcion(ois.readUTF());
19                carr.addProduct(p); // Lo agregamos
20                carr.setTotal(0.0); // Cambiamos el total a 0
21                break;
22            case 2: // Actualiza producto
23                p = new Producto();
24                p.setIdProducto(ois.readInt());
25                p.setNombre(ois.readUTF());
26                p.setCantidad(ois.readInt());
27                p.setPrecio(ois.readDouble());
28                p.setDescripcion(ois.readUTF());
29                carr.removeProduct(p.getIdProducto(), true); // Lo borramos y reindexamos
30                carr.addProduct(p); // Agregamos producto
31                carr.setTotal(0.0); // Cambiamos el total a 0
32                break;
33            case 3: // Quita producto
34                carr.removeProduct(ois.readInt(), true); // Lo quitamos de la lista
35                carr.setTotal(0.0);
36                break;
37            case 4: // Salir
38                sePuedeSalir = true;
39                break;
40        }
41        saveCarritoCompra(carr);
42
43        oos.writeObject(carr);
44        oos.flush(); // Enviamos el carrito actualizado
45        oos.reset(); // Hago un reset para que no tenga cache de objetos
46    }
47
48 }
```

Código 17 Segmento de código que gestiona el abastecedor desde el servidor

Y por último creamos un objeto para el servidor y para el cliente, que se encargue de iniciar los procesos para cada uno.

```

1 public class Cliente {
2
3     public static void main(String[] args) {
4         try {
5             BufferedReader br = new BufferedReader(
6                 new InputStreamReader(System.in)); // Creamos un buffer para la lectura del teclado
7             System.out.print("Ingresa la direccion IP: ");
8             String host = br.readLine();
9             System.out.print("Ingresa el puerto: ");
10            int puerto = Integer.parseInt(br.readLine());
11            Socket cl = new Socket(host, puerto); // Creamos un socket para conectarnos al servidor con el host y el
12                                                    // puerto
13            System.out.println("Ingresa el tipo de usuario:\n1)Cliente\n2)Admin");
14            int tipo = Integer.parseInt(br.readLine().trim());
15            Gestor ges = new Gestor(); // Generamos un objeto para manejar el tipo de usuario
16            ObjectOutputStream oos = new ObjectOutputStream(cl.getOutputStream());
17            oos.flush(); // Hacemos un flush porque parece que hay un bug y se debe hacer inmediatamente
18                        // al crear el ObjectOutputStream
19            oos.reset(); // Hago un reset para que no tenga cache de objetos
20            ObjectInputStream ois = new ObjectInputStream(cl.getInputStream());
21            oos.writeInt(tipo); // Enviamos el tipo de usuario
22            oos.flush();
23            oos.reset(); // Hago un reset para que no tenga cache de objetos
24            switch (tipo) {
25                case 1: // Cliente
26                    ges.gestionaClienteCliente(ois, oos, br); // Gestionamos si se refiere a un cliente, desde el
27                                                                // cliente
28                    break;
29                case 2: // Abastecedor
30                    ges.gestionaAbastecedorCliente(ois, oos, br); // Gestionamos a un Abastecedor, desde el cliente
31                    break;
32                default:
33                    System.out.println("Opcion invalida");
34                    break;
35            }
36            br.close();
37            ois.close();
38            oos.close();
39            cl.close();
40        } catch (Exception e) {
41            e.printStackTrace();
42        }
43    }
44 }
45 }

```

Código 18 Segmento de código del cliente


```

1 public class Servidor {
2
3     public static void main(String[] args) {
4         while (true) {
5             try {
6                 System.out.println("Esperando conexión en el puerto 3070");
7                 Gestor ges = new Gestor(); // Creamos el objeto gestor de usuarios
8                 ServerSocket s = new ServerSocket(3070); // Generamos un Servidor en el puerto 3070
9                 Socket cl = s.accept(); // Esperamos conexión
10                System.out.println("Conexión aceptada");
11                System.out.println("Conexión establecida desde"
12                    + cl.getInetAddress() + ":" + cl.getPort()); // Mostramos la información de la conexión
13                ObjectOutputStream oos = new ObjectOutputStream(cl.getOutputStream());
14                oos.flush(); // Hacemos flush por una cosa de Java
15                oos.reset(); // Hago un reset para que no tenga cache de objetos
16                ObjectInputStream ois = new ObjectInputStream(cl.getInputStream());
17                int tipo = ois.readInt(); // Recibimos el tipo de usuario del cliente
18                switch (tipo) {
19                    case 1: // Cliente
20                        oos.writeBoolean(true); // Si autorizado
21                        ges.gestionaClienteServidor(ois, oos); // Gestiona Cliente desde el servidor
22                        oos.flush();
23                        oos.reset();
24                        break;
25                    case 2: // Abastecedor
26                        String pass = ois.readUTF(); // Recibimos la contraseña
27                        if (!(pass == null) && pass.equals("12345")) { // Solo la comparamos con 12345
28                            oos.writeBoolean(true); // Le devolvemos que si paso
29                            oos.flush();
30                            oos.reset(); // Hago un reset para que no tenga cache de objetos
31                            ges.gestionaAbastecedorServidor(ois, oos); // Gestiona Abastecedor desde el Servidor
32                        } else {
33                            oos.writeBoolean(false); // No autorizado
34                            oos.flush();
35                            oos.reset(); // Hago un reset para que no tenga cache de objetos
36                        }
37                        break;
38                    default:
39                        oos.writeBoolean(false); // No autorizado
40                        oos.flush();
41                        oos.reset();
42                        break;
43                }
44                ois.close();
45                oos.close();
46                cl.close();
47                s.close();
48                TimeUnit.SECONDS.sleep(5); // Sleep de 5 segundos
49            } catch (Exception e) {
50                e.printStackTrace();
51                System.out.println("Pues se murio");
52                try {
53                    TimeUnit.SECONDS.sleep(10); // Sleep de 10 segundos, porque puede ser que el puerto siga ocupado
54                } catch (Exception e2) {
55                    e2.printStackTrace();
56                }
57            }
58        }
59    }
60 }

```

Código 19 Segmento de código del Servidor

Y finalmente solo las pruebas

```
C:\Program Files\PowerShell\7\pwsh.exe

^c1
Esperando conexión en el puerto 3070
Conexión aceptada
Conexión establecida desde/127.0.0.1:51782
[]

^c2
Ingresa la dirección IP: 127.0.0.1
Ingresa el puerto: 3070
Ingresa el tipo de usuario:
1) Cliente
2) Admin
1
Ingresa la operación que quieras realizar:
1) Revisar carrito de compras
2) Listar productos en tienda
3) Mostrar detalles del producto
4) Comprar producto
5) Modificar carrito
6) Finalizar compra
7) Salir
2
ID: 1 Nombre: Discraft Ultra Star Disco Deportivo de 175 g Existencias: 29
ID: 2 Nombre: Nintendo Joy-Con Pro Controller para Nintendo Switch - Standard Edition Existencias: 48
ID: 3 Nombre: Teclado Ultra Delgado - K223 Teclas silenciosas y rpidas con iluminación din mica RGB, Resistente a derrames, iluminación din mica RGB, Resistente a derrames, 104 Teclas Ergonómico USB Cable Teclados para PC/Mac con Windows 7, 8, 10 o Mac OS (Negro) Existencias: 97
ID: 4 Nombre: Oculus Quest 2 - Advanced All-In-One Virtual Reality Headset - 128 GB Existencias: 1500
ID: 5 Nombre: Tomie: Complete Deluxe Edition Existencias: 20
Ingresa la operación que quieras realizar:
1) Revisar carrito de compras
2) Listar productos en tienda
3) Mostrar detalles del producto
4) Comprar producto
5) Modificar carrito
6) Finalizar compra
7) Salir
```

Código 20 Ventana de pruebas, parte 1

```
C:\Program Files\PowerShell\7\pwsh.exe

^c1
Esperando conexión en el puerto 3070
Conexión aceptada
Conexión establecida desde/127.0.0.1:51782
[]

^c2
3
Ingresa el ID del producto: 4
Producto{
  idProducto=4,
  nombre=Oculus Quest 2 - Advanced All-In-One Virtual Reality Headset - 128 GB,
  cantidad=1500,
  precio=7408.18,
  descripcion=Next-level Hardware - Make every move count with a blazing-fast processor and our highest-resolution display. All-In-One Gaming - With backward compatibility, you can explore new titles and old favorites in the expansive Quest content library. Immersive Entertainment - Get the best seat in the house to live concerts, groundbreaking films, exclusive events and more. Easy Setup - Just open the box, set up with the smartphone app and jump into VR. No PC or console needed. Requires wireless internet access and the Oculus app (free download) to set up device. Premium Display - Catch every detail with a stunning display that features 56% more pixels than the original Quest;Ultimate Control - Redesigned Oculus Touch controllers transport your movements directly into VR with intuitive controls;PC VR Compatible - Step into incredible Oculus Rift titles by connecting an Oculus Link cable to a compatible gaming PC. Oculus Link Cable sold separately;3D Cinematic Sound - Hear in all directions with built-in speakers that deliver cinematic 3D positional audio
}

Ingresa la operación que quieras realizar:
1) Revisar carrito de compras
2) Listar productos en tienda
3) Mostrar detalles del producto
4) Comprar producto
5) Modificar carrito
6) Finalizar compra
7) Salir
```

Código 21 Ventana de pruebas, parte 2

Cuestionario

1. ¿Qué es serialización?
 - a. La serialización es el proceso de codificación de un objeto para sus diferentes usos
2. ¿En qué difiere del marshalling?
 - a. Se refiere al proceso de codificación para el transporte de datos
3. ¿Por qué es importante la serialización?
 - a. Porque tiene diferentes usos
 - i. Un método de persistencia de objetos

- ii. Un método para la distribución de objetos
 - iii. Un método para detectar cambios en variables en el tiempo.
- 4. Cuatro ejemplos de tecnologías en las que se haga uso de la serialización.
 - a. Java
 - b. Python
 - c. PHP
 - d. Perl
- 5. ¿Qué limita el tamaño máximo del archivo a enviar?
 - a. El límite del paquete del protocolo

Conclusiones

Referencias

- [1] Oracle, «Object Serialization: Frequently Asked Questions,» Oracle, [En línea]. Available: <https://www.oracle.com/java/technologies/javase/serializationfaq-jsp.html>. [Último acceso: 4 Junio 2022].
- [2] Apache PDFBox, «Apache PDFBox | A Java PDF Library,» Apache PDFBox, 5 Mayo 2022. [En línea]. Available: <https://pdfbox.apache.org/>. [Último acceso: 4 Junio 2022].
- [3] Oracle, «Desktop (Java Platform SE 8),» Oracle, [En línea]. Available: [https://docs.oracle.com/javase/8/docs/api/java/awt/Desktop.html#open\(java.io.File\)](https://docs.oracle.com/javase/8/docs/api/java/awt/Desktop.html#open(java.io.File)). [Último acceso: 4 Junio 2022].
- [4] «Getting Started | Building Java Projects with Maven,» Spring, [En línea]. Available: <https://spring.io/guides/gs/maven/#scratch>. [Último acceso: 4 Junio 2022].