UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Facultad de Ciencias

Introducción a las Ciencias de la Computación

Práctica 14: Persistencia de datos.

Profesora: Amparo López Gaona
Ayudante: Ramsés Antonio López Soto
Ayudantes de Laboratorio:
Adrián Aguilera Moreno
Kevin Jair Torres Valencia

Objetivos

El objetivo de esta práctica es que el alumno refuerce sus conocimientos acerca de cómo lograr que los objetos creados durante la ejecución de un programa persistan más allá de la ejecución del mismo mediante su almacenamiento en archivos en disco, así como acerca del procedimiento para recuperar objetos de archivos. Este proceso se conoce en `Java` como serialización de objetos.

Introducción

La serialización de objetos permite escribir objetos a archivos con una sola instrucción, con lo cual quedan grabados hasta que se decida eliminarlos o modificarlos. También permite recuperar los objetos grabados en archivos.

Para que objetos de una clase puedan serializarse es necesario que dicha clase implemente la interfaz Serializable, que se encuentra definida en el paquete java.io. La interfaz es una interfaz de marcado que tiene el siguiente código:

```
public interface Serializable { }
```

Para que un objeto sea serializable todas las variables de la estructura deben ser serializables. Todos los tipos primitivos, los objetos de la clase `String` y algunos de otras clases de `Java` son serializables.

Para serializar objetos, además de especificar que serán serializables se requiere trabajar con un archivo, en el cual se almacenen o bien se recuperen de ahí cuando se requiera. Al trabajar con archivos es necesario considerar las posibles excepciones que pueden dispararse, éstas son subclases de `IOException` definida en el paquete `java.io`. En este paquete está también definida una serie de subclases de esta excepción, entre ellas `EOFException` y `FileNotFoundException`.

Para grabar objetos en un archivo, es decir, para serializar, se requiere crear un objeto de la clase `ObjectOutputStream` del paquete `java.io` utilizando la siguiente instrucción:

```
ObjectOutputStream obj = new ObjectOutputStream(new
    FileOutputStream(nombreArch));
```

Una vez creado el objeto de la clase `ObjectOutputStream` se puede utilizar el método `writeObject(objeto)` para grabar el objeto que toma como parámetro. Si el objeto que se intenta grabar no es de una clase que implemente la interfaz `Serializable` se dispara la excepción `NotSerializableException`.

Al terminar de serializar todos los objetos se debe llamar al método close para asegurar que no se pierdan los objetos grabados en el archivo especificado. Independientemente de que haya habido un error o no es necesario cerrar el archivo, por esto es recomendable incluirla

en la clausula `finally`.

La operación complementaria a grabar objetos es la de recuperarlos. Para recuperar objetos de un archivo se requiere crear un objeto de la clase `ObjectInputStream` como sigue:

```
ObjectInputStream objeto = new ObjectInputStream(new FileInputStream(nombreArch));
```

Para leer los objetos serializados se utiliza el método `readObject()`, el cual construye un objeto de la clase indicada pero lo regresa como una referencia de tipo `Object`, por lo que es necesario hacer una conversión explícita.

Desarrollo

1. En esta práctica tu misión es implementar el siguiente pseudocódigo:

Algorithm 1 Algoritmo secreto

```
ALGORITMO cubiertaPoligonal
ENTRADA: puntos - un arreglo de objetos Point con las coordenadas (x, y)
SALIDA: Un arreglo de puntos que forman la cubierta poligonal del conjunto de entrada.
n ← longitud de puntos
tempCubierta ← arreglo de tamaño n para almacenar los puntos extremos
cubierta ← 0 {Contador para los puntos en la cubierta}
for cada pointIndex desde 0 hasta n − 1 do
    isInside ← False
    for cada i desde 0 hasta n − 1 do
        if i == pointIndex then
            continuar
        for cada j desde i + 1 hasta n − 1 do
            if j == pointIndex then
                continuar
            for cada k desde j + 1 hasta n − 1 do
                if k == pointIndex then
                    continuar
                p ← puntos[i]
                q ← puntos[j]
                r ← puntos[k]
                current ← puntos[pointIndex]
                if isPointInsideTriangle(p, q, r, current) then
                    isInside ← True
                    break
            if isInside then
                break
        if isInside then
            break
    if NOT isInside then
        tempCubierta[cubierta] ← puntos[pointIndex]
        cubierta ← cubierta + 1
poligono ← arreglo de tamaño cubierta
for cada i desde 0 hasta cubierta − 1 do
    poligono[i] ← tempCubierta[i]
return poligono
```

La salida `poligono` en el pseudocódigo anterior debe ser persistente, y tu programa debe permitir visualizar el resultado por el pseudocódigo, es decir, mostrar solo los identificadores de los puntos, (imprimen su resultado en la terminal y tambien conservarlos en un archivo "solucion.txt").

Formato de Entrega

1. Las prácticas se entregarán en parejas.
2. Cada práctica (sus archivos y directorios) deberá estar contenida en un directorio llamado `EquipoX_pY`, donde:
   (a) X es el número de equipo correspondiente.
   (b) Y es el número de la práctica.

   Por ejemplo: `Equipo09_p01`
3. NO incluir los archivos .class dentro de la carpeta.
4. Los archivos de código fuente deben estar documentados.
5. Se pueden discutir y resolver dudas entre los integrantes del grupo. Pero cualquier práctica plagiada total o parcialmente será penalizada con cero para los involucrados.
6. La práctica se debe subir al Github Classroom correspondiente.
7. La entrega en classroom debe contener el link HTTPS y SSH de su repositorio y es lo único que se debe entregar.
8. El horario y día de entrega se acordará en la clase de laboratorio y no deberá sobrepasar 2 clases de laboratorio.