Rodríguez Herrera Isaí

Juarez Sandoval Fabian

Universidad Politécnica de Tecámac

Ingeniería en Software

1922IS

Vespertino

PROGRAMACION PARA MOVILES II

UNIVERSIDAD POLITÉCNICA DE TECÁMAC

INDICE

Modelado de acceso a datos.

El modelado de datos es el proceso de analizar y definir todos los diferentes datos que su empresa recopila y produce, así como las relaciones entre esos bits de datos. El proceso de modelado de sus datos crea una representación visual de datos a medida que se utilizan en su negocio, y el propio proceso es un ejercicio de conocimiento y aclaración de sus requisitos de datos.

¿Por qué es importante el modelado?

Al modelar sus datos, documentará los datos que tiene, cómo los usa y cuáles son sus requisitos relacionados con el uso, la protección y la gobernanza. Mediante el modelado de datos, su organización:

- Crea una estructura para la colaboración entre su equipo de TI y sus equipos comerciales.
- Expone oportunidades para mejorar los procesos comerciales, al definir las necesidades y los usos de los datos.
- Ahorra tiempo y dinero en TI y en inversiones en procesos, mediante una planificación adecuada por adelantado.
- Reduce los errores (y la entrada de datos redundantes propensa a errores), al tiempo que mejora la integridad de los datos.
- Aumenta la velocidad y el rendimiento de la recuperación y el análisis de datos, al planificar la capacidad y el crecimiento.

Por lo tanto, no se trata solo de lo que obtiene con el modelado de datos, sino también de cómo lo obtiene. El propio proceso ofrece importantes ventajas.

¿Qué es la manipulación de datos en dispositivos móviles?

Un Objeto de Acceso a Datos o Data Access Object (DAO) son una serie de objetos que le permiten tener acceso y manipular datos mediante programación en bases de datos locales o remotos. Puede utilizar DAO para administrar bases de datos, así como sus objetos y su estructura.

Es un componente de software que suministra una interfaz común entre la aplicación y uno o más dispositivos de almacenamiento de datos, tales como una Base de datos o un archivo. El término se aplica frecuentemente al Patrón de diseño Object.

¿Como funciona?

Los Objetos de Acceso a Datos son un Patrón de Diseño Core J2EE y considerados una buena práctica. La ventaja de usar objetos de acceso a datos es que cualquier objeto de negocio (aquel que contiene detalles específicos de operación o aplicación) no requiere conocimiento directo del destino final de la información que manipula.

Los Objetos de Acceso a Datos pueden usarse en Java para aislar a una aplicación de la tecnología de persistencia Java subyacente(API de Persistencia Java), la cual podría ser JDBC, JDO, EJB,CMP(Persistencia controlada por el Contenedor), TopLink, Hibernate, iBATIS, o cualquier otra tecnología de persistencia. Usando Objetos de Acceso de Datos significa que la tecnología subyacente puede ser actualizada o cambiada sin cambiar otras partes de la aplicación.

¿Qué es la persistencia de datos en los dispositivos moviles?

La persistencia en Android consiste en tres tipos de almacenamientos con un propósito muy específico. Con Shared Preferences podemos almacenar y recuperar en el formato clave-valor información como texto, booleanos y números; lo que lo convierte en potencial para almacenar configuraciones del usuario como: estilos, preferencias, etc.

¿En que consiste?

Este tipo de persistencia es uno de los más conocidos debido a que son soportados por la mayoría de los lenguajes de programación aparte de Java; consiste en guardar y recuperar la información en archivos; Android permite escribir y leer archivos que se encuentren ubicados en la propia Memoria Interna del dispositivo; al igual que con Shared Preferences.

2.1 Almacenar archivos en Memoria Interna

Hay que tener en cuenta que estos archivos son guardados en la Memoria Interna del dispositivo la cual puede ser limitada en el dispositivo y puede ralentizar el mismo.

Consideraciones

No es necesario pedir permisos especiales en el Manifest.

Sobre su uso: Apertura y Escritura de archivos en Android

Android proporciona el método OutputStreamWriter para escribir archivos en Android a través de la apertura del mismo con el método openFileOutput; el mismo recibe como parámetro:

El nombre del archivo.

El modo de acceso:

Es posible configurar los archivos para que solo puedan ser gestionados por la aplicación y por nadie ni nada más; ni siquiera el usuario; los modos de acceso son los siguientes:

- MODE_PRIVATE: Solo es asesible por la aplicación y por nadie ni nada más (crea el archivo o lo sobrescribe si ya existe).
- MODE_APPEND: Añade contenido a un archivo existente en el dispositivo.
- MODE_WORLD_READABLE: Permite que otras aplicaciones puedan leer el archivo (deprecated desde el API 17).
- MODE_WORLD_WRITEABLE: Permite que otras aplicaciones puedan escribir el archivo (deprecated desde el API 17).

¿Cuáles son los mecanismos de tolerancia a fallos?

Un diseño tolerante a fallos es un sistema que está capacitado para continuar su funcionamiento cuando algún componente del sistema falla., posiblemente a un nivel más reducido, lo que es mejor a que el sistema falle completamente.

El hardware con tolerancia a fallos requiere a veces que las piezas rotas pueden ser extraídas y reemplazadas por nuevas piezas mientras el sistema sigue funcionando (en informática conocido como sustitución en caliente). Un sistema de este tipo con una sola copia de seguridad se conoce como único punto tolerante y representa la gran mayoría de sistemas tolerantes de fallos. En este tipo de sistemas el tiempo medio entre fallos debe ser lo suficientemente largo como para que los operadores puedan arreglar los dispositivos rotos (tiempo medio de reparación) antes de que la copia de seguridad también falle. Se

recomienda que el tiempo entre fallos sea lo más largo posible, pero no es estrictamente necesario en un sistema tolerante de fallos.

La tolerancia a fallos funciona muy bien en las aplicaciones informáticas. La primera computadora con tolerancia a fallos fue SAPO en la República Checa. La empresa Tandem Computers ha basado todo su negocio en este tipo de equipos, que utilizan un único punto de tolerancia para crear sus sistemas NonStop, cuyos periodos de funcionamiento pueden medirse en años.

Los programas informáticos también pueden utilizar arquitecturas libres de fallos, por ejemplo, en la replicación de procesos.

Los formatos de datos también pueden ser diseñados para degradarse correctamente. El lenguaje HTML por ejemplo, está diseñado para ser compatible, lo cual permite a los navegadores Web ignorar las nuevas entidades html que no entienden sin provocar que el documento sea inutilizable.

Codificación

MainActivity

```
public class MainActivity extends AppCompatActivity {

    vistaPong vistaPong;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Display display = getWindowManager().getDefaultDisplay();
        Point tamanio = new Point();
        display.getSize(tamanio);
        vistaPong = new vistaPong(this, tamanio.x, tamanio.y);
        setContentView(vistaPong);
    }
    @Override
    protected void onResume() {
        super.onResume();
        vistaPong.resume();
    }
    @Override
    protected void onPause() {
        super.onPause();
        vistaPong.pause();
    }
}
```

Pelota

```
public class Pelota {

    private RectF mCoordenadasRec;
    private float mVelcidadX;
    private float mVelicidadY;
    private float pelotaAncho;
    private float pelotaAlto;

    public Pelota(int screenX, int screenY){
        pelotaAncho = screenX / 100;
        pelotaAlto = pelotaAncho;
        mVelicidadY = screenY / 4;
        mVelcidadX = mVelicidadY;
        mCoordenadasRec = new RectF();
    }
    public RectF getRect(){
        return mCoordenadasRec;
    }
    public void actualizar(long fps){
        mCoordenadasRec.left = mCoordenadasRec.left + (mVelcidadX / fps);
        mCoordenadasRec.top = mCoordenadasRec.top + (mVelicidadY / fps);
        mCoordenadasRec.right = mCoordenadasRec.left + pelotaAncho;
        mCoordenadasRec.bottom = mCoordenadasRec.top - pelotaAlto;
    }
    public void invertirVelocidadX(){
        mVelicidadY = -mVelicidadY;
    }

    public void invertirVelocidadY(){
        mVelcidadX = -mVelcidadX;
    }
```

```
public void setRandomVelocidad(){
    Random generator = new Random();
    int answer = generator.nextInt(2);
    if(answer == 0){
        invertirVelocidadX();
    }
}
public void aumentarVelocidad(){
    mVelcidadX = mVelcidadX + mVelcidadX / 10;
    mVelicidadY = mVelicidadY + mVelicidadY / 10;
}
public void quitarObsY(float y){
    mCoordenadasRec.bottom = y;
    mCoordenadasRec.top = y - pelotaAlto;
}

public void quitarObsX(float x){
    mCoordenadasRec.left = x;
    mCoordenadasRec.right = x + pelotaAncho;
}

public void reset(int x, int y){
    mCoordenadasRec.left = x / 2;
    mCoordenadasRec.top = y - 20;
    mCoordenadasRec.right = x / 2 + pelotaAncho;
    mCoordenadasRec.bottom = y - 20 - pelotaAlto;
}
```

VistaPong

```
class vistaPong extends SurfaceView implements Runnable{
    Thread mGameThread = null;
    SurfaceHolder contenedor;
    volatile boolean estaJugando;
    boolean estaPausa = true;

    Canvas mCanvas;
    Paint pincel;
    long mFPS;

    int mTamPanX;
    int mTamPanY;

    Barra barra;
    Pelota pelota;

    int puntuacion = 0;
    int vidas = 3;

    public vistaPong(Context context, int x, int y) {
        super(context);
        mTamPanX = x;
        mTamPanY = y;
        contenedor = getHolder();
        pincel = new Paint();
        barra = new Barra(mTamPanX, mTamPanY);
        pelota = new Pelota(mTamPanX, mTamPanY);
        setupAndRestart();
    }
```

```
public void setupAndRestart(){
    pelota.reset(mTamPanX, mTamPanY);
    if(vidas == 0) {
        puntuacion = 0;
        vidas = 3;
    }
}
@Override
public void run() {
    while (estaJugando) {
        long inicioFrame = System.currentTimeMillis();
        if(!estaPausa){
            actualizar();
        }
        dibujar();
        long frameActual = System.currentTimeMillis() - inicioFrame;
        if (frameActual >= 1) {
            mFPS = 1000 / frameActual;
        }

    }

}
public void dibujar() {
```

```
public void dibujar() {

    if (contenedor.getSurface().isValid()) {
        mCanvas = contenedor.lockCanvas();
        mCanvas.drawColor(Color.argb(0, 0, 0, 0));
        pincel.setColor(Color.argb(255, 255, 255, 255));
        mCanvas.drawRect(barra.getRect(), pincel);
        mCanvas.drawRect(pelota.getRect(), pincel);
        pincel.setColor(Color.argb(255, 255, 255, 255));
        pincel.setTextSize(40);
        mCanvas.drawText("Puntuacion: " + puntuacion + "   Lives: " + vidas, 10, 50, pincel);
        contenedor.unlockCanvasAndPost(mCanvas);
    }

}
```

```
public void actualizar() {
    barra.update(mFPS);
    pelota.actualizar(mFPS);
    if(RectF.intersects(barra.getRect(), pelota.getRect())) {
        pelota.setRandomVelocidad();
        pelota.invertirVelocidadY();
        pelota.quitarObsY(barra.getRect().top - 2);

        puntuacion++;
        pelota.aumentarVelocidad();
    }
    if(pelota.getRect().bottom > mTamPanY){
        pelota.invertirVelocidadY();
        pelota.quitarObsY(mTamPanY - 2);
        vidas--;
        if(vidas == 0){
            estaPausa = true;
            setupAndRestart();
        }
    }
    if(pelota.getRect().top < 0){
        pelota.invertirVelocidadY();
        pelota.quitarObsY(24);
    }

    if(pelota.getRect().left < 0){
        pelota.invertirVelocidadX();
        pelota.quitarObsX(2);
    }
```

```
        if(pelota.getRect().right > mTamPanX){
            pelota.invertirVelocidadX();
            pelota.quitarObsX(mTamPanX - 22);
        }
    }

    public void pause() {
        estaJugando = false;
        try {
            mGameThread.join();
        } catch (InterruptedException e) {
            Log.e("Error:", "...");
        }

    }
    public void resume() {
        estaJugando = true;
        mGameThread = new Thread(this);
        mGameThread.start();
    }
```

```
@Override
public boolean onTouchEvent(MotionEvent motionEvent) {

    switch (motionEvent.getAction() & MotionEvent.ACTION_MASK) {
        case MotionEvent.ACTION_DOWN:
            estaPausa = false;
            if(motionEvent.getX() > mTamPanX / 2){
                barra.setMovementState(barra.RIGHT);
            }
            else{
                barra.setMovementState(barra.LEFT);
            }
            break;
        case MotionEvent.ACTION_UP:
            barra.setMovementState(barra.STOPPED);
            break;
    }
    return true;
}
```

Barra

```
public class Barra {

    private RectF mRect;

    private float mLength;
    private float mHeight;


    private float mXCoord;


    private float mYCoord;



    private float mBatSpeed;

    public final int STOPPED = 0;
    public final int LEFT = 1;
    public final int RIGHT = 2;


    private int mBatMoving = STOPPED;


    private int mScreenX;
    private int mScreenY;
```

```
public Barra(int x, int y){

    mScreenX = x;
    mScreenY = y;


    mLength = mScreenX / 8;


    mHeight = mScreenY / 25;


    mXCoord = mScreenX / 2;
    mYCoord = mScreenY - 20;

    mRect = new RectF(mXCoord, mYCoord, mXCoord + mLength, mYCoord + mHeight);


    mBatSpeed = mScreenX;

}


public RectF getRect(){
    return mRect;
}

public void setMovementState(int state){
    mBatMoving = state;
}
```

```
    public void update(long fps){

        if(mBatMoving == LEFT){
            mXCoord = mXCoord - mBatSpeed / fps;
        }

        if(mBatMoving == RIGHT){
            mXCoord = mXCoord + mBatSpeed / fps;
        }


        if(mRect.left < 0){ mXCoord = 0; } if(mRect.right > mScreenX){
            mXCoord = mScreenX -

                    (mRect.right - mRect.left);
        }

        mRect.left = mXCoord;
        mRect.right = mXCoord + mLength;
    }
```