



## Sesión 16: Almacenamiento persistente

---

---

---

---

---

---

---

---

### Índice



- Almacenes de registros
- Registros
- Consultas de registros
- Listener del registro
- Optimización de consultas
- Patrón de diseño adaptador

---

---

---

---

---

---

---

---

### Almacenamiento persistente



- Almacenes de registros
- Registros
- Consultas de registros
- Listener del registro
- Optimización de consultas
- Patrón de diseño adaptador

---

---

---

---

---

---

---

---

## RMS



- **RMS = *Record Management System***
  - Nos permite almacenar datos de forma persistente
  - Esta API se encuentra en `javax.microedition.rms`
- **No se especifica la forma en la que se guardan realmente los datos**
  - Deben guardarse en cualquier memoria no volátil
- **Los datos se guardan en almacenes de registros**
  - Un almacén de registros contiene varios registros
  - Cada registro contiene
    - Un identificador
    - Un *array* de *bytes* como datos

Java y Dispositivos Móviles

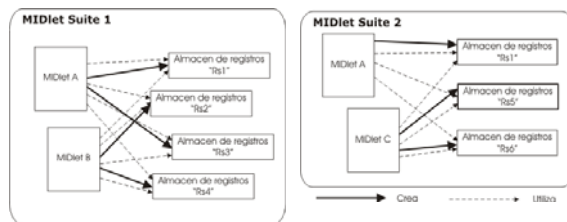
© 2007 Depto. Ciencia Computación e IA

Almacenamiento persistente-4

## Almacenes de registros



- **Un MIDlet puede crear y acceder a varios almacenes**
- **Los almacenes son privados de cada *suite***



Java y Dispositivos Móviles

© 2007 Depto. Ciencia Computación e IA

Almacenamiento persistente-5

## Operaciones con los almacenes



- **Abrir/crear un almacén**

```
RecordStore rs =  
    RecordStore.openRecordStore(nombre, true);
```

- **Cerrar un almacén**

```
rs.closeRecordStore();
```

- **Listar los almacenes disponibles**

```
String [] nombres = RecordStore.listRecordStores();
```


- **Eliminar un almacén**

```
RecordStore.deleteRecordStore(nombre);
```

Java y Dispositivos Móviles

© 2007 Depto. Ciencia Computación e IA

Almacenamiento persistente-6

Propiedades de los almacenes


- Nombre**

```
String nombre = rs.getName();
```
- Fecha de modificación**

```
long timestamp = rs.getLastModified();
```
- Versión**

```
int version = rs.getVersion();
```
- Tamaño**

```
int tam = rs.getSize();
```
- Tamaño disponible**

```
int libre = rs.getSizeAvailable();
```

Java y Dispositivos Móviles
© 2007 Depto. Ciencia Computación e IA
Almacenamiento persistente-7

---

---

---


---

---

---

---

---

Almacenamiento persistente


- Almacenes de registros
- Registros**
- Consultas de registros
- Listener del registro
- Optimización de consultas
- Patrón de diseño adaptador

Java y Dispositivos Móviles
© 2007 Depto. Ciencia Computación e IA
Almacenamiento persistente-8

---

---

---


---

---

---

---

---

Conjunto de registros


- Cada almacén contendrá un conjunto de registros
- Cada registro tiene

Identificador	Datos
1	A4 5D 12 09 ...
2	32 3E 1A 98 ...
3	FE 26 3B 45 ...

  - Identificador
    - Valor entero
  - Datos
    - Array de bytes
- El identificador se autoincrementará con cada inserción
- Deberemos codificar los datos en binario para añadirlos en un registro
  - Utilizar objetos `DataInputStream` y `DataOutputStream`
  - Podemos utilizar los métodos de serialización de los objetos

Java y Dispositivos Móviles
© 2007 Depto. Ciencia Computación e IA
Almacenamiento persistente-9

---

---

---

---

---

---

---

---

## Añadir datos



### ▪ Codificar los datos en binario

```
ByteArrayOutputStream baos =  
    new ByteArrayOutputStream();  
DataOutputStream dos =  
    new DataOutputStream(baos);  
dos.writeUTF(nombre);  
dos.writeInt(edad);  
byte [] datos = baos.toByteArray();
```

### ▪ Añadir los datos como registro al almacén

```
int id = rs.addRecord(datos, 0, datos.length);  
0  
rs.setRecord(id, datos, 0, datos.length);
```

Java y Dispositivos Móviles

© 2007 Depto. Ciencia Computación e IA

Almacenamiento persistente-10

---

---

---

---

---

---

---

---

## Consultar y borrar datos



### ▪ Leemos el registro del almacén

```
byte [] datos = rs.getRecord(id);
```

### ▪ Descodificamos los datos

```
ByteArrayInputStream bais =  
    new ByteArrayInputStream(datos);  
DataInputStream dis = new DataInputStream(bais);  
String nombre = dis.readUTF();  
String edad = dis.readInt();
```

### ▪ Eliminar un registro

```
rs.deleteRecord(id);
```

Java y Dispositivos Móviles

© 2007 Depto. Ciencia Computación e IA

Almacenamiento persistente-11

---

---

---

---

---

---

---

---

## Almacenamiento persistente



- Almacenes de registros
- Registros
- Consultas de registros
- Listener del registro
- Optimización de consultas
- Patrón de diseño adaptador

Java y Dispositivos Móviles

© 2007 Depto. Ciencia Computación e IA

Almacenamiento persistente-12

---

---

---

---

---

---

---

---

## Enumeración de registros



- Normalmente no conoceremos el identificador del registro buscado *a priori*
  - Podremos recorrer el conjunto de registros para buscarlo
  - Utilizaremos un objeto `RecordEnumeration`

```
RecordEnumeration re =  
    rs.enumerateRecords(null, null, false);
```

- Recorremos la enumeración

```
while(re.hasNextElement()) {  
    int id = re.nextRecordId();  
    byte [] datos = rs.getRecord(id);  
    // Procesar datos obtenidos  
    ...  
}
```

Java y Dispositivos Móviles

© 2007 Depto. Ciencia Computación e IA

Almacenamiento persistente-13

---

---

---

---

---

---

---

---

## Ordenación de registros



- Creamos un comparador

```
public class MiComparador implements RecordComparator {  
    public int compare(byte [] reg1, byte [] reg2) {  
        if( /* reg1 es anterior a reg2 */ ) {  
            return RecordComparator.PRECEDES;  
        } else if( /* reg1 es posterior a reg2 */ ) {  
            return RecordComparator.FOLLOWS;  
        } else if( /* reg1 es igual a reg2 */ ) {  
            return RecordComparator.EQUIVALENT;  
        }  
    }  
}
```

- Obtenemos la enumeración

```
RecordEnumeration re =  
    rs.enumerateRecords(new MiComparador(), null, false);
```

Java y Dispositivos Móviles

© 2007 Depto. Ciencia Computación e IA

Almacenamiento persistente-14

---

---

---

---

---

---

---

---

## Filtrado de registros



- Creamos un filtro

```
public class MiFiltro implements RecordFilter {  
    public boolean matches(byte [] reg) {  
        if( /* reg nos interesa */ ) {  
            return true;  
        } else {  
            return false;  
        }  
    }  
}
```

- Obtenemos la enumeración

```
RecordEnumeration re =  
    rs.enumerateRecords(null, new MiFiltro(), false);
```

Java y Dispositivos Móviles

© 2007 Depto. Ciencia Computación e IA

Almacenamiento persistente-15

---

---

---

---

---

---

---

---

## Almacenamiento persistente



- Almacenes de registros
- Registros
- Consultas de registros
- Listener del registro
- Optimización de consultas
- Patrón de diseño adaptador

Java y Dispositivos Móviles

© 2007 Depto. Ciencia Computación e IA

Almacenamiento persistente-16

---

---

---

---

---

---

---

---

## Listener



- Nos permite “escuchar” cambios en el registro

```
public class MiListener implements RecordListener {
    public void recordAdded(RecordStore rs, int id) {
        // Añadido un registro con identificador id a rs
    }
    public void recordChanged(RecordStore rs, int id) {
        // Modificado el registro con identificador id en rs
    }
    public void recordDeleted(RecordStore rs, int id) {
        // Eliminado el registro con identificador id de rs
    }
}
```

- Registrar el listener

```
rs.addRecordListener(new MiListener());
```

Java y Dispositivos Móviles

© 2007 Depto. Ciencia Computación e IA

Almacenamiento persistente-17

---

---

---

---

---

---

---

---

## Almacenamiento persistente



- Almacenes de registros
- Registros
- Consultas de registros
- Listener del registro
- Optimización de consultas
- Patrón de diseño adaptador

Java y Dispositivos Móviles

© 2007 Depto. Ciencia Computación e IA

Almacenamiento persistente-18

---

---

---

---

---

---

---

---

## Consultas



- Necesitamos realizar consultas en el almacén
  - Buscar registros que cumplan ciertos criterios
- Podemos utilizar una enumeración con un `RecordFilter`
- Esto nos forzará a recorrer todos los registros del almacén
  - Deserializar cada registro
  - Comprobar si los datos cumplen los criterios de la búsqueda
- Si tenemos almacenado un gran volumen de datos, hará que las consultas sean lentas

Java y Dispositivos Móviles

© 2007 Depto. Ciencia Computación e IA

Almacenamiento persistente-19

---

---

---

---

---

---

---

---

## Índices



- Podemos optimizar las consultas creando un almacén de índices
  - Tendremos un índice por cada registro almacenado
  - Los índices contendrán sólo los datos por los que se suele realizar la búsqueda
  - Además contendrán una referencia al registro donde se encuentra almacenado el dato al que corresponde

```
public class Cita {  
    Date fecha;  
    String asunto;  
    String descripcion;  
    String lugar;  
    String contacto;  
    boolean alarma;  
}
```

```
public class IndiceCita {  
    int id;  
    Date fecha;  
    boolean alarma;  
}
```

Java y Dispositivos Móviles

© 2007 Depto. Ciencia Computación e IA

Almacenamiento persistente-20

---

---

---

---

---

---

---

---

## Almacenamiento persistente



- Almacenes de registros
- Registros
- Consultas de registros
- Listener del registro
- Optimización de consultas
- Patrón de diseño adaptador

Java y Dispositivos Móviles

© 2007 Depto. Ciencia Computación e IA

Almacenamiento persistente-21

---

---

---

---

---

---

---

---

## Adaptador



- Para implementar el acceso a RMS conviene utilizar el patrón de diseño adaptador
  - Interfaz adaptada al dominio de nuestra aplicación, que encapsula una API genérica y nos aísla de ella
- Por ejemplo, para nuestra aplicación de citas
  - En RMS tenemos un método `getRecord`
  - En nuestro adaptador tenemos un método `getCita`
- Desde nuestra aplicación siempre accederemos al registro a través del adaptador

Java y Dispositivos Móviles

© 2007 Depto. Ciencia Computación e IA

Almacenamiento persistente-22

---

---

---

---

---

---

---

---

## Ejemplo de adaptador



```
public class AdaptadorRMS {

    public final static String RS_DATOS = "rs_datos";
    RecordStore rsDatos;

    public AdaptadorRMS() throws RecordStoreException {
        rsDatos = RecordStore.openRecordStore(RS_DATOS, true);
    }

    public int addCita(Cita cita)
        throws IOException, RecordStoreException {
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        DataOutputStream dos = new DataOutputStream(baos);
        cita.serialize(dos);
        byte[] datos = baos.toByteArray();
        int id = rsDatos.addRecord(datos, 0, datos.length);
        return id;
    }
}
```

Java y Dispositivos Móviles

© 2007 Depto. Ciencia Computación e IA

Almacenamiento persistente-23

---

---

---

---

---

---

---

---

## Clave primaria



- Necesitamos una clave primaria para poder referenciar cada registro
  - Podemos utilizar el identificador del registro en RMS
  - Deberemos guardarnos una referencia a este ID al leer los datos para posteriormente poderlo modificar o eliminar

```
public Cita getCita(int id)
    throws RecordStoreException, IOException {
    byte[] datos = rsDatos.getRecord(id);
    ByteArrayInputStream bais =
        new ByteArrayInputStream(datos);
    DataInputStream dis = new DataInputStream(bais);
    Cita cita = Cita.deserialize(dis);
    cita.setRmsID(id);
    return cita;
}
```

Java y Dispositivos Móviles

© 2007 Depto. Ciencia Computación e IA

Almacenamiento persistente-24

---

---

---

---

---

---

---

---