Report of CS 522 Assignment 3

Name: Yunfeng Wei
CWID: 10394963
E-mail: ywei14@stevens.edu
Video: In the Zip archive file,
BookStore App: Yunfeng_Wei_assignment_3_BookStore.MOV
BasicChat App: Yunfeng_Wei_assignment_3_BasicChat.MOV

Part 1: Book Store

1. To return a book entity object back to the main activity, create a BookContract class and a AuthorContract class in the sub package contracts, as Figure 1.1 and Figure 1.2. Plus, in the BookContract class, define a SEPARATE_CHAR which is used by the Pattern SEPARATOR to separate each authors' names by "|".

```
public class BookContract {
    public static final String ID = "_id";
    public static final String TITLE = "title";
    public static final String ISBN = "isbn";
    public static final String PRICE = "price";
    public static final String AUTHORS = "authors";
    public static final String SEPARATE_CHAR = "|";
    private static final Pattern SEPARATOR = Pattern.compile(Character.toString(SEPARATE_CHAR.charAt(0)), Pattern.LITERAL);

    public static String getTitle(Cursor cursor) {
        return cursor.getString(cursor.getColumnIndexOrThrow(TITLE));
    }

    public static void putTitle(ContentValues values, String title) { values.put(TITLE, title); }

    public static long getId(Cursor cursor) {
        return cursor.getLong(cursor.getColumnIndexOrThrow(ID));
    }

    public static void putId(ContentValues values, long id) { values.put(ID, id); }

    public static String getIsbn(Cursor cursor) {
        return cursor.getString(cursor.getColumnIndexOrThrow(ISBN));
    }

    public static Author[] getAuthors(Cursor cursor) {
        String temp = cursor.getString(cursor.getColumnIndexOrThrow(AUTHORS));
        String[] authors = SEPARATOR.split(temp);
        int len = authors.length;
        Author[] author = new Author[len];
        for (int i = 0; i < len; i++) {
            String[] names = authors[i].split(Author.SEPARATE_KEY);
            if (names.length == 2) {
                author[i] = new Author(names[0], names[1]);
            }
            else if (names.length == 3) {
                author[i] = new Author(names[0], names[1], names[2]);
            }
        }
        return author;
    }

    public static void putIsbn (ContentValues values, String isbn) { values.put(ISBN, isbn); }
```

Figure 1.1

```java
package edu.stevens.cs522.bookstore.contracts;

import ...

/**
 * Created by wyf920621 on 2/8/15.
 */
public class AuthorContract {
    public static final String ID = "_id";
    public static final String NAME = "name";
    public static final String BOOK_FOREIGN_KEY = "book_fk";

    public static long getId(Cursor cursor) {
        return cursor.getLong(cursor.getColumnIndexOrThrow(ID));
    }
    public static void putId(ContentValues values, long id) { values.put(ID, id); }

    public static String getName(Cursor cursor) {
        return cursor.getString(cursor.getColumnIndexOrThrow(NAME));
    }

    public static void putName(ContentValues values, String name) { values.put(NAME, name); }

    public static long getBookForeignKey(Cursor cursor) {
        return cursor.getLong(cursor.getColumnIndexOrThrow(BOOK_FOREIGN_KEY));
    }

    public static void putBookForeignKey(ContentValues values, long book_fk) {
        values.put(BOOK_FOREIGN_KEY, book_fk);
    }
}
```

Figure 1.2

2. Extend the Book entity class and Author entity class to initialize the fields from an input cursor and a method that writes the fields of the entity to a ContentValue object, as Figure 2.1 and Figure 2.2

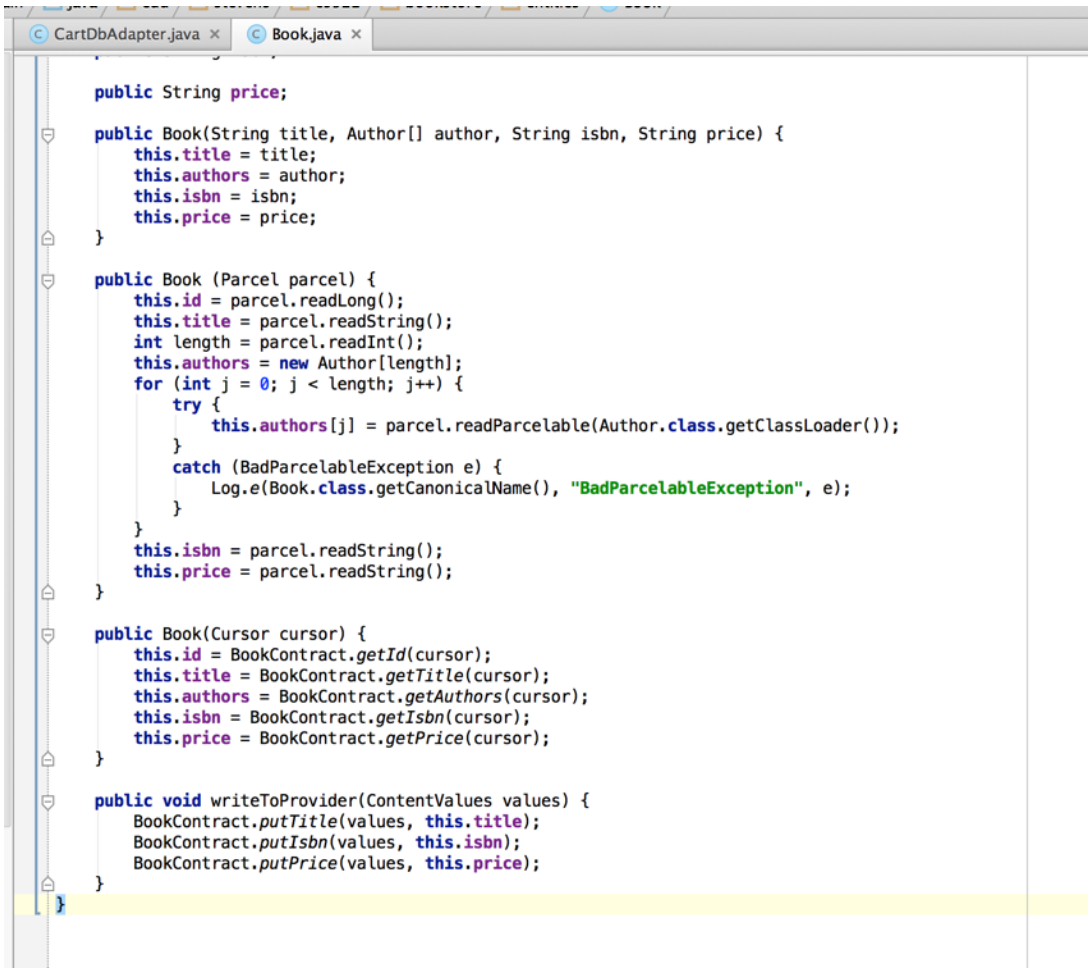


Figure 2.1

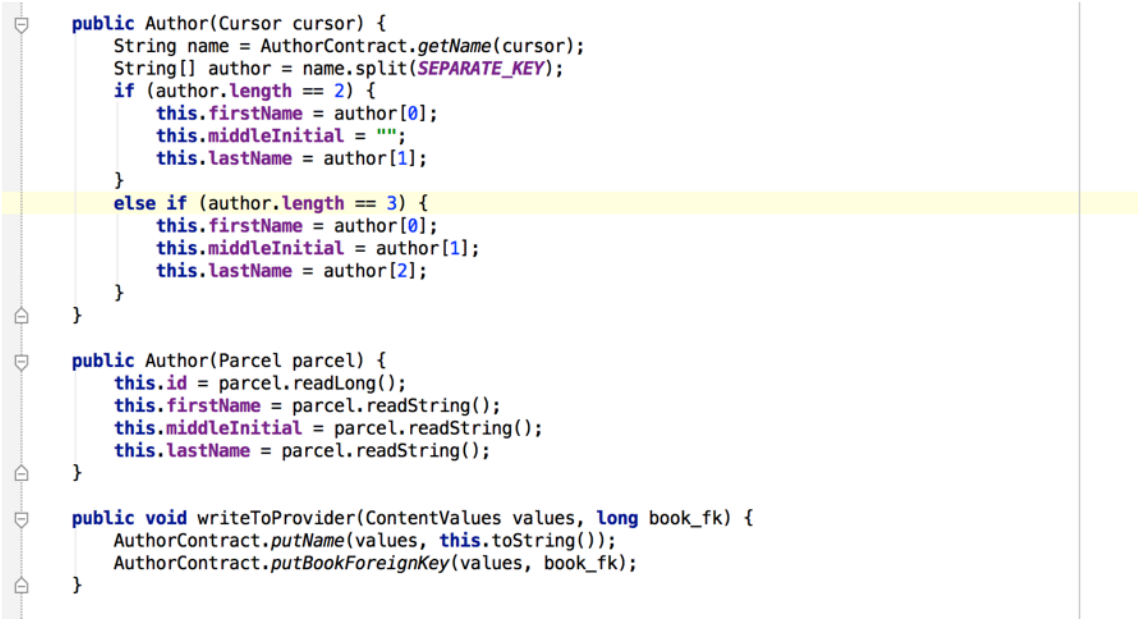


Figure 2.2

3. Define a sub package called databases. In the sub package, define an adapter for database called CartDbAdapter. In the class, first define useful string literals such as DATABASE_NAME, BOOK_TABLE, AUTHOR_TABLE, DATABASE_VERSION, BOOK_TABLE_CREATE, AUTHOR_TABLE_CREATE, CREATE_INDEX and DATABASE_CREATE as Figure 3.1. Then, define a static inner class called DatabaseHelper which extends SQLiteOpenHelper as Figure 3.2. Finally, define each useful application-specific operations for accessing the database, such as open(), close(), fetchAllBooks(), fetchBook(), persist(), delete(), deleteAll() and so on as Figure 3.3 and Figure 3.4

```
*/
public class CartDbAdapter {
    private SQLiteDatabase database;
    private DatabaseHelper databaseHelper;
    private static final String DATABASE_NAME = "cart.db";
    private static final String BOOK_TABLE = "Books";
    private static final String AUTHOR_TABLE = "Authors";
    private static final int DATABASE_VERSION = 1;
    private static final String BOOK_TABLE_CREATE =
            "CREATE TABLE " + BOOK_TABLE + " (" + BookContract.ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " + BookContract.TITLE
            + " TEXT NOT NULL, " + BookContract.ISBN + " TEXT NOT NULL, " + BookContract.PRICE + " TEXT NOT NULL " + ");";
    private static final String AUTHOR_TABLE_CREATE =
            "CREATE TABLE " + AUTHOR_TABLE + " (" + AuthorContract.ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " + AuthorContract.NAME
            + " TEXT NOT NULL, " + AuthorContract.BOOK_FOREIGN_KEY + " INTEGER, " + "FOREIGN KEY (" + AuthorContract.BOOK_FOREIGN_KEY +
            + BookContract.ID + ") ON DELETE CASCADE" + ");";
    private static final String CREATE_INDEX =
            "CREATE INDEX AuthorsBookIndex ON " + AUTHOR_TABLE + "(" + AuthorContract.BOOK_FOREIGN_KEY + ");";
    private static final String DATABASE_CREATE = BOOK_TABLE_CREATE + AUTHOR_TABLE_CREATE + CREATE_INDEX;

    // Constructor
    public CartDbAdapter(Context context) {
        databaseHelper = new DatabaseHelper(context, DATABASE_NAME, null, DATABASE_VERSION);
    }
```

Figure 3.1

```
    private static class DatabaseHelper extends SQLiteOpenHelper {
        public DatabaseHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version) {
            super(context, name, factory, version);
        }

        @Override
        public void onCreate(SQLiteDatabase sqLiteDatabase) {
            sqLiteDatabase.execSQL("PRAGMA foreign_keys = ON");
            sqLiteDatabase.execSQL(BOOK_TABLE_CREATE);
            sqLiteDatabase.execSQL(AUTHOR_TABLE_CREATE);
            sqLiteDatabase.execSQL(CREATE_INDEX);
        }

        @Override
        public void onUpgrade(SQLiteDatabase sqLiteDatabase, int i, int i2) {
            // Log the version upgrade
            Log.w("DatabaseHelper", "Upgrading from version" + i + " to " + i2);
            sqLiteDatabase.execSQL("DROP TABLE IF EXISTS " + BOOK_TABLE);
            sqLiteDatabase.execSQL("DROP TABLE IF EXISTS" + AUTHOR_TABLE);
            onCreate(sqLiteDatabase);
        }
    }
```

Figure 3.2

```
public CartDbAdapter open() throws SQLException {
    database = databaseHelper.getWritableDatabase();
    database.execSQL("PRAGMA foreign_keys = ON");
    return this;
}

public Cursor fetchAllBooks() {
    String query = "SELECT " + BOOK_TABLE + "." + BookContract.ID + ", " + BookContract.TITLE + ", " + BookContract.PRICE
            + ", " + BookContract.ISBN + ", GROUP_CONCAT(" + AuthorContract.NAME + ",\'" + BookContract.SEPARATE_CHAR + "\') as " +
            BOOK_TABLE + " LEFT OUTER JOIN " + AUTHOR_TABLE + " ON " + BOOK_TABLE + "." + BookContract.ID + " = " + AUTHOR_TABLE +
            AuthorContract.BOOK_FOREIGN_KEY + " GROUP BY " + BOOK_TABLE + "." + BookContract.ID + ", " + BookContract.TITLE + ", "
            + BookContract.PRICE + ", " + BookContract.ISBN + ";";
    return database.rawQuery(query, null);
}

public Book fetchBook(long rowId) {
    String query = "SELECT " + BOOK_TABLE + "." + BookContract.ID + " AS " + BookContract.ID + ", " + BOOK_TABLE + "." + BookContra
            + ", " + BOOK_TABLE + "." + BookContract.ISBN + " AS " + BookContract.ISBN + ", GROUP_CONCAT(" + AUTHOR_TABLE + "." + A
            BOOK_TABLE + " LEFT OUTER JOIN " + AUTHOR_TABLE + " ON " + BOOK_TABLE + "." + BookContract.ID + " = " + AUTHOR_TABLE +
            AuthorContract.BOOK_FOREIGN_KEY + " WHERE " + BOOK_TABLE + "." + BookContract.ID + " = ?" + ";";
    Cursor cursor = database.rawQuery(query, new String[] {String.valueOf(rowId)});
    cursor.moveToNext();
    return new Book(cursor);
}
```

Figure 3.3

```
public void persist(Book book, Author[] authors) throws SQLException {
    ContentValues values = new ContentValues();
    book.writeToProvider(values);
    Cursor tmp_cursor = database.query(BOOK_TABLE, new String[] {BookContract.ID, BookContract.TITLE, BookContract.ISBN}, BookContr
    long book_fk;
    if (!tmp_cursor.moveToFirst()) {
        book_fk = database.insertOrThrow(BOOK_TABLE, null, values);
        book.id = book_fk;
    }
    else {
        book_fk = BookContract.getId(tmp_cursor);
        book.id = book_fk;
    }
    tmp_cursor.close();
    values.clear();
    for (Author author: authors) {
        author.writeToProvider(values, book_fk);
        Cursor cursor = database.query(AUTHOR_TABLE, new String[] {AuthorContract.ID, AuthorContract.NAME, AuthorContract.BOOK_FORE
        long aid;
        if (!cursor.moveToFirst()) {
            aid = database.insertOrThrow(AUTHOR_TABLE, null, values);
            author.id = aid;
        }
        else {
            aid = AuthorContract.getId(cursor);
            author.id = aid;
        }
        cursor.close();
        values.clear();
    }
}

public boolean delete(Book book) {
    int res = database.delete(BOOK_TABLE, BookContract.ISBN + " = ?", new String[] {book.isbn});
    return res != 0;
}

public boolean deleteAll() {
    int res = database.delete(BOOK_TABLE, null, null);
    return res != 0;
}
```

Figure 3.4

4. In the BookStoreActivity, use the SimpleCursorAdapter to connect the cursor resulting from a query to the list view, use the startManagingCursor to manage the cursor as Figure 4.1. To set the Contextual Action Bar to show the detail of a book or delete it, define an Actionmode.Callback abstract class and override the onCreateActionMode() method, onPrepareActionMode() method, onActionItemSelected() method and onDestroyActionMode method as Figure 4.2, set the onItemLongClickListener to start the Contextual Action Bar as Figure 4.3. Finally, when get the activity result, requery the cursor and swap the cursor in order

to refresh the adapter, and when destroying the activity, close the cursor and database as Figure 4.4

```
private SimpleCursorAdapter bookCursorAdapter = null;
private CartDbAdapter adapter = null;
private Cursor cursor = null;
private ActionMode mActionMode = null;
public static f                                          _KEY";
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // TODO Set the layout (use cart.xml layout)
    setContentView(R.layout.cart);
    // TODO use an array adapter to display the cart contents.

    adapter = new CartDbAdapter(this);
    adapter.open();
    String[] from = new String[] { BookContract.TITLE, BookContract.AUTHORS};
    int[] to = new int[] {R.id.cart_row_title, R.id.cart_row_author};
    bookCursorAdapter = new SimpleCursorAdapter(this, R.layout.cart_row, cursor, from, to);
    cursor = adapter.fetchAllBooks();
    startManagingCursor(cursor);
    setListAdapter(bookCursorAdapter);
    bookCursorAdapter.changeCursor(cursor);
```

[< Android API 17 Platform >] android.os
public final class Bundle extends Object
implements Parcelable, Cloneable

Figure 4.1

```
private ActionMode.Callback mActionModeCallback = new ActionMode.Callback() {
    @Override
    public boolean onCreateActionMode(ActionMode actionMode, Menu menu) {
        actionMode.getMenuInflater().inflate(R.menu.bookstore_cab_menu, menu);
        return true;
    }


    @Override
    public boolean onPrepareActionMode(ActionMode actionMode, Menu menu) { return false; }

    @Override
    public boolean onActionItemClicked(ActionMode actionMode, MenuItem menuItem) {
        long rowId = (Long)mActionMode.getTag();
        Book book = adapter.fetchBook(rowId);
        switch (menuItem.getItemId()) {
            case R.id.detail:
                Intent intent = new Intent(BookStoreActivity.this, BookDetailActivity.class);
                intent.putExtra(BOOK_STORE_KEY, book);
                startActivity(intent);
                break;
            case R.id.delete:
                adapter.delete(book);
                cursor = adapter.fetchAllBooks();
                bookCursorAdapter.swapCursor(cursor);
                //bookCursorAdapter.notifyDataSetChanged();
                break;
        }
        return true;
    }

    @Override
    public void onDestroyActionMode(ActionMode actionMode) {
        mActionMode = null;
    }
};
```

Figure 4.2

```
        getListView().setChoiceMode(ListView.CHOICE_MODE_SINGLE);

        getListView().setOnItemLongClickListener(new AdapterView.OnItemLongClickListener() {
            @Override
            public boolean onItemLongClick(AdapterView<?> adapterView, View view, int position, long id) {
                view.setSelected(true);
                cursor.moveToPosition(position);
                Long rowId = BookContract.getId(cursor);

                if (mActionMode != null) {
                    return false;
                }
                mActionMode = BookStoreActivity.this.startActionMode(mActionModeCallback);
                mActionMode.setTag(rowId);
                return true;
            }

        });
```

Figure 4.3

```
    @Override
    protected void onActivityResult(int requestCode, int resultCode,
            Intent intent) {
        super.onActivityResult(requestCode, resultCode, intent);
        // TODO Handle results from the Search and Checkout activities.

        // Use SEARCH_REQUEST and CHECKOUT_REQUEST codes to distinguish the cases.

        // SEARCH: add the book that is returned to the shopping cart.

        // CHECKOUT: empty the shopping cart.
        if (requestCode == ADD_REQUEST) {
            if (resultCode == RESULT_OK) {
                Log.v("ADD_REQUEST", "RESULT OK");
                cursor = adapter.fetchAllBooks();
                bookCursorAdapter.swapCursor(cursor);
                //bookCursorAdapter.notifyDataSetChanged();
            }
        }
        else if (requestCode == CHECKOUT_REQUEST) {
            if (resultCode == RESULT_OK) {
                adapter.deleteAll();
                cursor = adapter.fetchAllBooks();
                bookCursorAdapter.swapCursor(cursor);
                //bookCursorAdapter.notifyDataSetChanged();
            }
        }
    }

    @Override
    protected void onDestroy() {
        stopManagingCursor(cursor);
        cursor.close();
        adapter.close();
        super.onDestroy();
```

Figure 4.4

5. In the AddBookActivity class, after search a book, use the adapter to persist the book and authors into the database and return to the BookStoreActivity as Figure 5.1. In the BookStoreActivity, if the CheckOutActivity results to OK, then delete all books in the database as Figure 5.2

```
        // CANCEL: cancel the search request
        switch (item.getItemId()) {
            case R.id.search:
                Book newBook = searchBook();
                adapter.persist(newBook, newBook.authors);
                setResult(RESULT_OK, intent);
                finish();
                break;
            case R.id.search_cancel:
                setResult(RESULT_CANCELED);
                finish();
                break;
        }
        return true;
    }
```

Figure 5.1

```
    @Override
    protected void onActivityResult(int requestCode, int resultCode,
            Intent intent) {
        super.onActivityResult(requestCode, resultCode, intent);
        // TODO Handle results from the Search and Checkout activities.

        // Use SEARCH_REQUEST and CHECKOUT_REQUEST codes to distinguish the cases.

        // SEARCH: add the book that is returned to the shopping cart.

        // CHECKOUT: empty the shopping cart.
        if (requestCode == ADD_REQUEST) {
            if (resultCode == RESULT_OK) {
                Log.v("ADD_REQUEST", "RESULT OK");
                cursor = adapter.fetchAllBooks();
                bookCursorAdapter.swapCursor(cursor);
                //bookCursorAdapter.notifyDataSetChanged();
            }
        }
        else if (requestCode == CHECKOUT_REQUEST) {
            if (resultCode == RESULT_OK) {
                adapter.deleteAll();
                cursor = adapter.fetchAllBooks();
                bookCursorAdapter.swapCursor(cursor);
                //bookCursorAdapter.notifyDataSetChanged();
            }
        }
    }
```

Figure 5.2

6. Finally run and test the App as the video shows.

Part 2: Basic Chat App

1. Create a new sub package called entities, in the package, define a Message class and a Peer class as Figure 1.1 and Figure 1.2, each class implements Parcelable interface to write to or read from a Parcel, each class has a writeToProvider method to write entity's fields into the ContentValue object, each class also has a constructor which initializes the fields from an input cursor.

Peer.java × Message.java ×

```java
/**
 * Created by wyf920621 on 2/10/15.
 */
public class Message implements Parcelable {
    public long id;
    public String messageText;
    public String sender;


    public static final Creator<Message> CREATOR = new Creator<Message>() {
        public Message createFromParcel(Parcel parcel) { return new Message(parcel); }

        public Message[] newArray(int i) { return new Message[i]; }
    };
    public int describeContents() { return 0; }

    public void writeToParcel(Parcel parcel, int i) {
        parcel.writeLong(id);
        parcel.writeString(messageText);
        parcel.writeString(sender);
    }

    public Message(String messageText, String sender) {
        this.id = 0;
        this.messageText = messageText;
        this.sender = sender;
    }

    public Message(Parcel parcel) {
        this.id = parcel.readLong();
        this.messageText = parcel.readString();
        this.sender = parcel.readString();
    }
    public void writeToProvider(ContentValues values, long peer_fk) {
        MessageContract.putMessageText(values, this.messageText);
        MessageContract.putSender(values, this.sender);
        MessageContract.putPeerFk(values, peer_fk);
    }
}
```

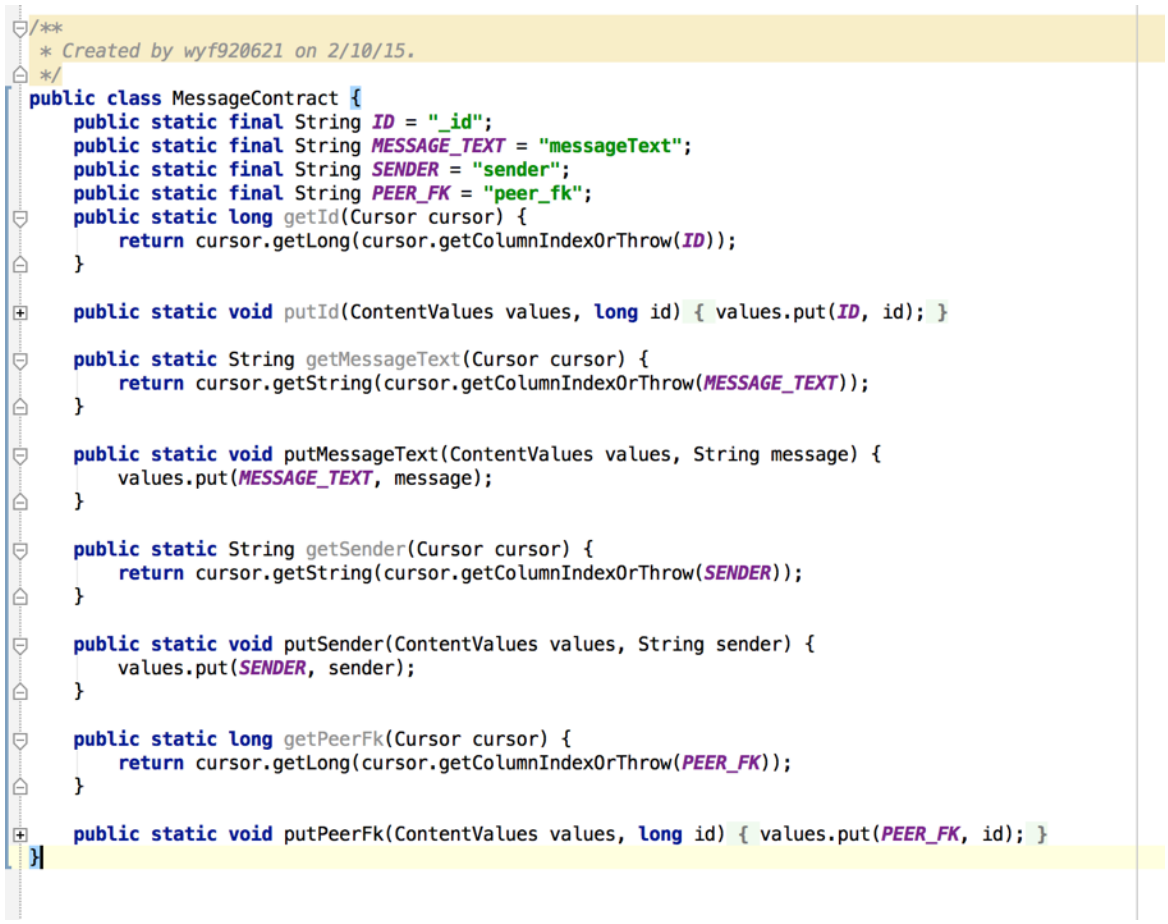
Figure 1.1

Peer.java × Message.java ×

```
public class Peer implements Parcelable {
    public long id;
    public String name;
    public InetAddress address;
    public int port;

    public static final Creator<Peer> CREATOR = new Creator<Peer>() {
        public Peer createFromParcel(Parcel parcel) { return new Peer(parcel); }

        public Peer[] newArray(int i) { return new Peer[i]; }
    };
    public int describeContents() { return 0; }

    public void writeToParcel(Parcel parcel, int i) {
        parcel.writeLong(id);
        parcel.writeString(name);
        parcel.writeString(address.getHostName());
        parcel.writeInt(port);
    }

    public Peer(String name, InetAddress address, int port) {
        this.id = 0;
        this.name = name;
        this.address = address;
        this.port = port;
    }
    public Peer(Parcel parcel) {
        this.id = parcel.readLong();
        this.name = parcel.readString();
        String hostName = parcel.readString();
        try {
            address = InetAddress.getByName(hostName);
        }
        catch (UnknownHostException e) {
            Log.e("Peer", e.getMessage());
        }

        this.port = parcel.readInt();
    }
    public void writeToProvider(ContentValues values) {
        PeerContract.putName(values, this.name);
        PeerContract.putAddress(values, this.address);
        PeerContract.putPort(values, this.port);
    }
}
```

Figure 1.2

2. Create a sub package called contracts and define a MessageContract class and a PeerContract class in the package to be used to query the database, as Figure 2.1 and Figure 2.2



```java
/**
 * Created by wyf920621 on 2/10/15.
 */
public class MessageContract {
    public static final String ID = "_id";
    public static final String MESSAGE_TEXT = "messageText";
    public static final String SENDER = "sender";
    public static final String PEER_FK = "peer_fk";
    public static long getId(Cursor cursor) {
        return cursor.getLong(cursor.getColumnIndexOrThrow(ID));
    }

    public static void putId(ContentValues values, long id) { values.put(ID, id); }

    public static String getMessageText(Cursor cursor) {
        return cursor.getString(cursor.getColumnIndexOrThrow(MESSAGE_TEXT));
    }

    public static void putMessageText(ContentValues values, String message) {
        values.put(MESSAGE_TEXT, message);
    }

    public static String getSender(Cursor cursor) {
        return cursor.getString(cursor.getColumnIndexOrThrow(SENDER));
    }

    public static void putSender(ContentValues values, String sender) {
        values.put(SENDER, sender);
    }

    public static long getPeerFk(Cursor cursor) {
        return cursor.getLong(cursor.getColumnIndexOrThrow(PEER_FK));
    }

    public static void putPeerFk(ContentValues values, long id) { values.put(PEER_FK, id); }
}
```

Figure 2.1

```
public class PeerContract {
    public static final String ID = "_id";
    public static final String NAME = "name";
    public static final String ADDRESS = "address";
    public static final String PORT = "port";

    public static long getId(Cursor cursor) {
        return cursor.getLong(cursor.getColumnIndexOrThrow(ID));
    }

    public static void putId(ContentValues values, long id) { values.put(ID, id); }

    public static String getName(Cursor cursor) {
        return cursor.getString(cursor.getColumnIndexOrThrow(NAME));
    }

    public static void putName(ContentValues values, String name) { values.put(NAME, name); }

    public static InetAddress getAddress(Cursor cursor) {
        String address = cursor.getString(cursor.getColumnIndexOrThrow(ADDRESS));
        try {
            return InetAddress.getByName(address);
        }
        catch (UnknownHostException e) {
            Log.e("PeerContract.getAddress()", e.getMessage());
            return null;
        }
    }

    public static void putAddress(ContentValues values, InetAddress address) {
        values.put(ADDRESS, address.getHostName());
    }

    public static int getPort(Cursor cursor) {
        return cursor.getInt(cursor.getColumnIndexOrThrow(PORT));
    }

    public static void putPort(ContentValues values, int port) { values.put(PORT, port); }
}
```

Figure 2.2

3. Create a sub package called databases. In the package, define a class called ChatDbAdapter, in the class, define several useful string literals such as DATABASE_NAME, MESSAGE_TABLE, PEER_TABLE, DATABASE_VERSION, PEER_TABLE_CREATE, MESSAGE_TABLE_CREATE, CREATE_INDEX and CREATE_TABLE as Figure 3.1. Also, define a inner class DatabaseHelper which extends SQLiteOpenHelper to help access the database as Figure 3.2. Finally, define several useful methods like open(), openReadable(), close(), fetchAllMessages(), insert(), fetchMessages() and so on as Figure 3.3 and Figure 3.4

```
public class ChatDbAdapter {
    private SQLiteDatabase database;
    private DatabaseHelper databaseHelper;
    private static final String DATABASE_NAME = "chat.db";
    private static final String MESSAGE_TABLE = "Messages";
    private static final String PEER_TABLE = "Peers";
    private static final int DATABASE_VERSION = 1;
    private static final String PEER_TABLE_CREATE =
            "CREATE TABLE " + PEER_TABLE + " ( " + PeerContract.ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
                    PeerContract.NAME + " TEXT NOT NULL, " + PeerContract.ADDRESS + " TEXT NOT NULL, " +
                    PeerContract.PORT + " INTEGER NOT NULL);";
    private static final String MESSAGE_TABLE_CREATE =
            "CREATE TABLE " + MESSAGE_TABLE + " ( " + MessageContract.ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
                    MessageContract.MESSAGE_TEXT + " TEXT NOT NULL, " + MessageContract.SENDER + " TEXT NOT NULL, " + MessageContra
                    "FOREIGN KEY (" + MessageContract.PEER_FK + ") REFERENCES " + PEER_TABLE + "(" + PeerContract.ID + ")" +
                    " ON DELETE CASCADE);";
    private static final String CREATE_INDEX =
            "CREATE INDEX MessagesPeerIndex ON " + MESSAGE_TABLE + "(" + MessageContract.PEER_FK + ");";
    private static final String CREATE_TABLE = PEER_TABLE_CREATE + MESSAGE_TABLE_CREATE + CREATE_INDEX;

    public ChatDbAdapter(Context context) {
        databaseHelper = new DatabaseHelper(context, DATABASE_NAME, null, DATABASE_VERSION);

    }
```

Figure 3.1

```
    public ChatDbAdapter open() throws SQLException
    {
        database = databaseHelper.getWritableDatabase();
        database.execSQL("PRAGMA foreign_keys = ON");
        return this;
    }

    public ChatDbAdapter openReadable() throws SQLException
    {
        database = databaseHelper.getReadableDatabase();
        database.execSQL("PRAGMA foreign_keys = ON");
        return this;
    }

    public void close() {
        database.close();
    }

    // TODO: Query to get Message from Peer: name + messageText
    public Cursor fetchAllMessages() {
        String query = "SELECT " + MESSAGE_TABLE + "." + MessageContract.ID + " AS " + MessageContract.ID + ", " + PEER_TABLE + "."
                MESSAGE_TABLE + "." + MessageContract.MESSAGE_TEXT + " AS " + MessageContract.MESSAGE_TEXT + " FROM " + PEER_TABLE
                "." + MessageContract.PEER_FK + ";";
        return database.rawQuery(query, null);
    }
```

Figure 3.2

```
    private static class DatabaseHelper extends SQLiteOpenHelper {
        public DatabaseHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version) {
            super(context, name, factory, version);
        }

        @Override
        public void onCreate(SQLiteDatabase sqLiteDatabase) {
            sqLiteDatabase.execSQL("PRAGMA foreign_keys = ON");
            sqLiteDatabase.execSQL(PEER_TABLE_CREATE);
            sqLiteDatabase.execSQL(MESSAGE_TABLE_CREATE);
            sqLiteDatabase.execSQL(CREATE_INDEX);
        }

        @Override
        public void onUpgrade(SQLiteDatabase sqLiteDatabase, int i, int i2) {
            Log.v("DatabaseHelper", "Upgrading from " + i + " to " + i2);
            sqLiteDatabase.execSQL("DROP TABLE IF EXISTS " + MESSAGE_TABLE);
            sqLiteDatabase.execSQL("DROP TABLE IF EXISTS " + PEER_TABLE);
            onCreate(sqLiteDatabase);
        }
    }
```

Figure 3.3

```
public void insert(Peer peer, Message message) throws SQLException{
    ContentValues values = new ContentValues();
    peer.writeToProvider(values);
    Cursor tmp_cursor = database.query(PEER_TABLE, new String[] {PeerContract.ID, PeerContract.NAME, PeerContract.ADDRESS, Peer
    long peer_fk;
    if (!tmp_cursor.moveToFirst()) {
        peer_fk = database.insertOrThrow(PEER_TABLE, null, values);
        peer.id = peer_fk;
    }
    else {
        peer_fk = PeerContract.getId(tmp_cursor);
        peer.id = peer_fk;
    }
    tmp_cursor.close();
    values.clear();
    message.writeToProvider(values, peer_fk);
    long mid = database.insertOrThrow(MESSAGE_TABLE, null, values);
    message.id = mid;
}

public Cursor fetchMessages(Peer peer) {
    long id = peer.id;
    return database.query(MESSAGE_TABLE, new String[] {MessageContract.ID, MessageContract.MESSAGE_TEXT}, MessageContract.PEER_
}
}
```

Figure 3.4

4. In the ChatServerActivity class, use the SimpleCursorAdapter class to connect the cursor resulting from a query to the list view. Use the startManagingCursor to manage the cursor from a query, as Figure 4.1, when Destroy the Activity, stop managing the cursor ,close the cursor and the database as Figure 4.2. Each time the database changes, requery the cursor and refresh the adapter as Figure 4.3

```
/*
 * TODO: Initialize the UI.
 */
ListView messageList = (ListView)findViewById(R.id.msgList);
next = (Button)findViewById(R.id.next);

dbAdapter = new ChatDbAdapter(this);
try {
    dbAdapter.open();
}
catch (SQLException e) {
    Log.e("open ChatDbAdapter", e.getMessage());
}
String[] from = new String[] {PeerContract.NAME, MessageContract.MESSAGE_TEXT};
int[] to = new int[] {R.id.peer_row, R.id.message_row};
cursorAdapter = new SimpleCursorAdapter(this, R.layout.peer_row, cursor, from, to);
cursor = dbAdapter.fetchAllMessages();
startManagingCursor(cursor);
messageList.setAdapter(cursorAdapter);
cursorAdapter.swapCursor(cursor);
/*
 * End Todo
 */
```

Figure 4.1

```
@Override
protected void onDestroy() {
    stopManagingCursor(cursor);
    cursor.close();
    dbAdapter.close();
    if (socketIsOK()) {
        closeSocket();
    }
    super.onDestroy();
}
```

Figure 4.2

```
public void onClick(View v) {

    byte[] receiveData = new byte[1024];

    DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);

    try {

        serverSocket.receive(receivePacket);
        Log.i(TAG, "Received a packet");

        InetAddress sourceIPAddress = receivePacket.getAddress();
        Log.i(TAG, "Source IP Address: " + sourceIPAddress);

        /*
         * TODO: Extract sender and receiver from message and display.
         */
        receiveData = receivePacket.getData();
        String temp = new String(receiveData, 0, receivePacket.getLength());
        if (!temp.isEmpty()) {
            String[] nameAndContent = SEPARATOR.split(temp);
            Peer peer = new Peer(nameAndContent[0], sourceIPAddress, serverSocket.getLocalPort());
            Message message = new Message(nameAndContent[1], nameAndContent[0]);
            dbAdapter.insert(peer, message);
            cursor = dbAdapter.fetchAllMessages();
            //cursorAdapter.notifyDataSetChanged();
            cursorAdapter.swapCursor(cursor);
        }
        /*
         * End Todo
         */

    } catch (Exception e) {
        Log.e(TAG, e.getMessage());
        socketOK = false;
    }
}

@Override
```

Figure 4.3

5. Create an Activity called PeerActivity to list each peers in the list view as Figure 5.1, set the OnItemClickListener to the list view to show the detail of a peer in PeerDetailActivity when the peer is clicked on the view.

```java
public class PeerActivity extends ListActivity {
    public static final String PEER_ACTIVITY_KEY = "edu.stevens.cs522.chat.oneway.server.PeerActivity";
    SimpleCursorAdapter cursorAdapter;
    Cursor cursor = null;
    ChatDbAdapter dbAdapter;
    ListView listView;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Intent intent = getIntent();
        setContentView(R.layout.activity_peer);
        dbAdapter = new ChatDbAdapter(this);
        try {
            dbAdapter.openReadable();
            cursor = dbAdapter.getPeers();
        }
        catch (SQLException e) {
            Log.v("dbAdapter.getPeers", e.getMessage());
        }
        listView = (ListView)findViewById(android.R.id.list);
        startManagingCursor(cursor);
        String[] from = new String[] {PeerContract.NAME};
        int[] to = new int[] {R.id.peer_list};
        cursorAdapter = new SimpleCursorAdapter(this, R.layout.peer_list, cursor, from, to);
        setListAdapter(cursorAdapter);
        getListView().setChoiceMode(ListView.CHOICE_MODE_SINGLE);
        getListView().setOnItemClickListener((adapterView, view, position, rowId) -> {
                cursor.moveToPosition(position);
                Peer peer = new Peer(PeerContract.getName(cursor), PeerContract.getAddress(cursor), PeerContract.getPort(cursor));

                peer.id = PeerContract.getId(cursor);

                Intent mIntent = new Intent(PeerActivity.this, PeerDetailActivity.class);
                mIntent.putExtra(PEER_ACTIVITY_KEY, peer);
                startActivity(mIntent);
        });
    }
```

Figure 5.1

6. Create a new Activity called PeerDetailActivity, show the detail of the peer including IP address, port and each message sent by the peer on the view as Figure 6.1.

```
public class PeerDetailActivity extends Activity {
    Cursor cursor;
    ChatDbAdapter dbAdapter;
    SimpleCursorAdapter cursorAdapter;
    ListView listView;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_peer_detail);
        Intent intent = getIntent();
        Peer peer = intent.getParcelableExtra(PeerActivity.PEER_ACTIVITY_KEY);
        TextView addressView = (TextView)findViewById(R.id.peer_ip);
        TextView portView = (TextView)findViewById(R.id.peer_port);
        addressView.setText("IP Address: " + peer.address.getHostAddress());
        portView.setText("Port: " + peer.port);
        dbAdapter = new ChatDbAdapter(this);
        try {
            dbAdapter.openReadable();
        }
        catch (SQLException e) {
            e.printStackTrace();
        }
        cursor = dbAdapter.fetchMessages(peer);
        startManagingCursor(cursor);
        String[] from = new String[] {MessageContract.MESSAGE_TEXT};
        int[] to = new int[] {R.id.peer_message};
        cursorAdapter = new SimpleCursorAdapter(this, R.layout.peer_message, cursor, from, to);
        listView = (ListView)findViewById(R.id.peer_messages);
        listView.setAdapter(cursorAdapter);
    }

    @Override
    protected void onDestroy() {
        stopManagingCursor(cursor);
        cursor.close();
        dbAdapter.close();
        super.onDestroy();
    }
}
```

Figure 6.1

7. Finally run and test the App as the video shows.