Super Asteroids Development - Task 1: Design

# CS240 Advanced Programming Concepts: Project 1

## Introduction

The first step in developing a program like *Super Asteroids* is to spend some time understanding the problem you are trying to solve. Studying the information presented in the *Super Asteroids* specification document, and attending class lectures and help sessions will you aide you in coming to understand the *Super Asteroids* program as it currently exists – and help you to grasp the project requirements that you are being tasked with. Once you understand the problem, you can begin the process of designing your solution – this will likely be an iterative process. Begin by identifying the pieces of the program that are missing/required (your responsibilities) and the constraints the existing design and implementation place upon possible solutions. Think through as much of your design as you can without writing any code – as a rule, design (and documentation of it) should always proceed implementation. This being said, on large and complex systems implementation is often incremental, meaning that after features are identified at a high level of abstraction (i.e. high level design) their low-level design and implementation are performed incrementally and according to a logical plan.

As implementation of a software system progresses it is common (especially for meaningful projects) for errors or omissions in the design to become apparent. This is to be expected, because some insights only come from actually “building things”. When this happens to you (and it WILL happen) make the necessary changes and corrections to your design and then incorporate them into your code. Such habits will ensure that your design always reflects your product, and greatly assist you in understanding and improving your work. This will become especially relevant at some point in the future when others must work with and build upon what you produce.

## Task 1

As a first step in completing the Super Asteroids program you will need to implement the data import capability (see the Super Android specification for more information about this feature). Numerous design decisions must be made in order to complete this task. In addition to the data importer class itself, a database schema and object model must be developed to facilitate the transfer of information from game configuration files into internal Java objects that can then be used to populate the database (in accordance with the aforementioned schema). These objects, henceforth referred to as model objects (Yup – I just said “henceforth”; welcome to the renaissance…), will also lend themselves to extracting data from the database during gameplay. Though they are mainly data containers, these objects will also contain important algorithms needed to run the game.

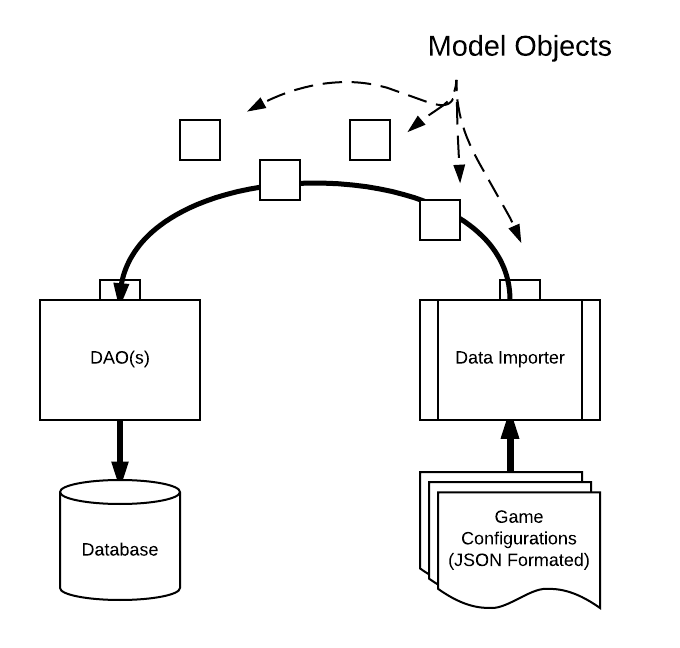


Figure 1

As shown in the diagram above, these objects do not “magically” insert themselves into the database; rather the information they contain must be inserted by one or more Data Access Objects (DAOs). These objects encapsulate the database access logic for the program, including querying, inserting, updating and deleting data in the database. In addition to their role in assisting the Data Importer to populate the database, the DAO(s) will also be used to load configuration data FROM the database during gameplay (see Figure 2 below). As determined by your approach, you will need to design one or more DAOs for this project.

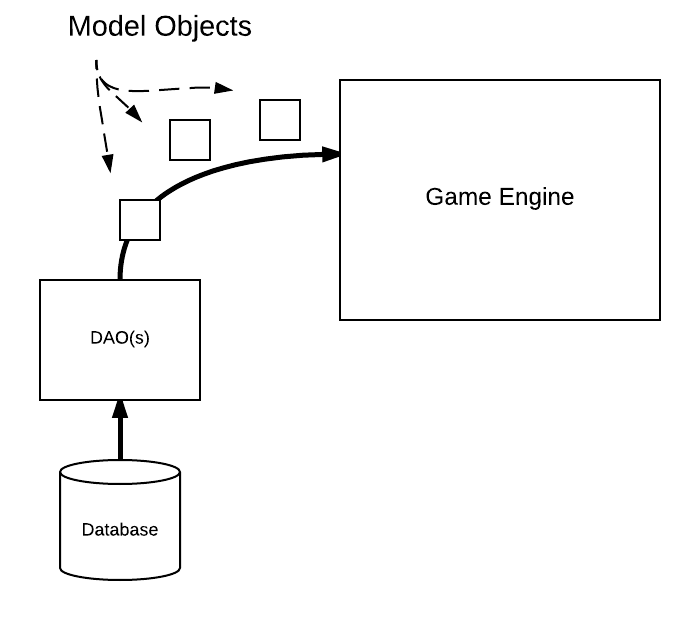
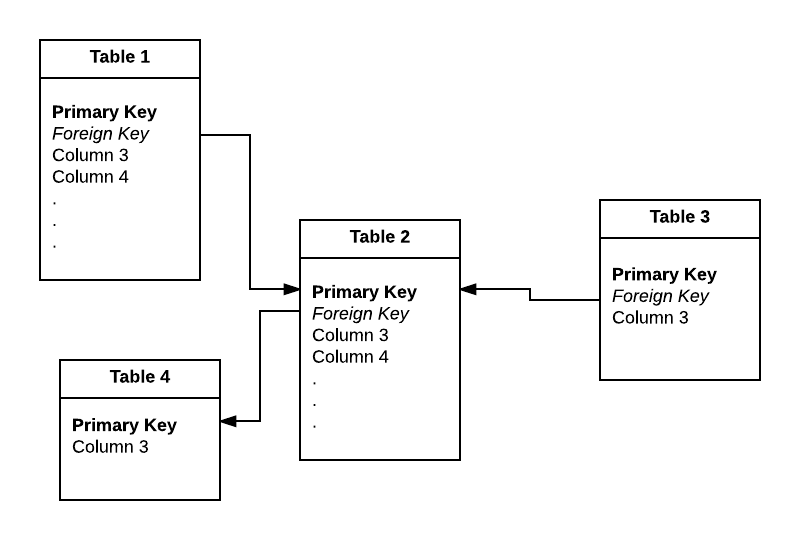


Figure 2

**Note:** These classes will be implemented using Android’s SQLite framework.

### SQLite Schema

Using the game configuration information found in the Super Asteroids specification as a reference (HINT: pay particular attention to the details of the JSON format we are using), design an SQLite database for the game. The database should be made up of relational tables designed to hold the data related to the various elements of the game. As noted in the specification, the database should not be designed to store image or sound files – though references to these files will need to be represented. Capture your database design using a relational schema diagram similar to the one shown here…



Such diagrams may be created using Lucidchart, Visio, Omnigraffle, or a drawing package of your choice.

Once you have settled on a schema for your database, create a text file containing a “drop table” statement for each table in your schema. Following the “drop table” statements, you must also list “create table” statements for each of the tables you have designed. Both the diagram and text document must be included in your assignment submission.

**Note:** Firefox’s “SQLite Manager” plugin has been identified as an easily accessible tool for interactively designing, creating, and working with a simple database. This tool might also assist you in forming the necessary queries if you are unfamiliar/unpracticed with SQL.

### Model Classes

In addition to the database design, you must design a set of model classes that captures the core information that is manipulated by the game at runtime. Naturally, these classes must encapsulate the information stored in the database you have just designed, and will likely include classes such as Cannon, Asteroid, Level, ExtraPart etc. Such classes will serve as receptacles of the information found in their respective database tables. Identify all the types of objects that you believe will be necessary, as well as any sensible inheritance relationships that such objects would reasonably have. The JSON format outlined in the specification should give you some ideas here. As a hint/help, you might observe that a significant portion of the game’s behavior involves visible objects – things that must be drawn on the screen, and updated as “stuff happens”. This suggests the creation of a VisibleObject class or interface that other classes or interfaces can inherit from. The following observations should give you a good head-start here:

* VisibleObjects all Update() and Draw()…
* Visible objects can be partitioned into objects that must have a position defined (PositionedObject) or those that don’t (Space (i.e. the star field)).
* PositionedObjects can be divided into objects that move (MovingObject), objects that sit on the background (BgObject), or stationary objects that are not part of the background but are periodically updated (MiniMap).
* MovingObjects include weapon projectiles (Projectile), asteroids (Asteroid), and the player’s ship (Ship).
* It will be useful to have a container for storing all model objects (AsteroidsGame)

These observations do not address all aspects necessary in your object model. For instance, you will need to account for the fact that a ship is made up of ship parts, some of which must be attached to other ship parts (i.e. the main body of the craft). How will you capture these details in your model class design?

Create your design using a UML class diagram, examples of such diagrams can be found in the project specification. Include all of the *model classes* and Data Access Object (DAO) classes you are proposing. Your design document should include the members and methods that you are currently aware of for each class that you identify. Once again Lucidchart, Visio, etc. are good tools for creating such diagrams.

**Note:** You can access UML “shapes” within Lucidchart by clicking the “more shapes” button in the lower left of your Lucidchart browser window, clicking the UML option and then hitting “save”. Several sets of shapes useful in generating UML style diagrams will be added to your list of shapes on the left side of the window.

Once you are satisfied with your design implement it in the Super Asteroids project. Obviously these class definitions are going to be very sparse at this point – that is fine. Create a new package or packages for your classes and implement them within it. Give your package a reasonable name (“model” perhaps). Make sure to document your code with meaningful comments describing at a minimum the purpose and function of classes and methods. Use Javadoc formatted comments so that they can be extracted by the Javadoc tool.

Once your classes have been implemented, run the Javadoc tool to extract/generate class and method documentation for the new files you have created. You will submit the resulting documents, along with your UML diagram as part of this assignment.

## Submission

Submissions for this assignment must be provided electronically. You will do this by first creating a website containing the four items mentioned above. As a reminder, these are:

1. A relational schema diagram depicting your database design.
2. A text document containing the Drop and Create statements for the tables of your design.
3. UML diagram(s) of the necessary model and data access classes you have identified to implement (Model classes, DAOs, dataImporter etc.).
4. The Javadoc generated documentation of your project (generated after you have added the model classes just mentioned).

Files placed in the public\_html directory of your CS home directory are automatically published to the web. This directory allows you to easily create your own website. For example, the URL for the personal website of a user with the BYU CS login “mary” would be <http://students.cs.byu.edu/~mary>. If Mary placed a file named database.txt in her public\_html directory, the file would be accessible on the web at <http://students.cs.byu.edu/~mary/database.txt> . You may use your CS web site, or some other website, to submit your design document.

**Note:** Please be sure to recursively give permissions to all files in the public\_html folder. This can be done by "chmod -R 755 /public\_html" or (if you are working locally on a lab machine) by checking the box at the bottom of the permissions tabs in the properties page.

Once your website/files are available via a website, finish the electronic submission of this assignment by sending the TA’s an email containing:

1. Your name
2. The URL of your relational schema diagram
3. The URL of your text file containing your DROP and CREATE statements
4. The URL(s) of your UML diagram document(s)
5. The URL of your Javadoc’s index.html file

**Note:** The TA email address is: cs240ta@cs.byu.edu