Games Programming

Philip Hanna 110CSC207

Code Repository

CSC207 Code Repository (including NetBeans and Java 1.6 setup)

The code repository is, well, a repository of code. Title aside, the repository provides a number of example programs and code snippets that you can study, use, adapt, extend, etc. – basically you are free to employ the code in whatever way you want. The idea behind the repository is to provide a number of example classes that illustrate some of the game programming concepts you will explore during the course. You can also employ the code within your project (if desired), e.g. using one of the example programs as a springboard from which to develop your own game.

The purpose of this document is twofold. Firstly, the steps needed to install and setup NetBeans and Java will be defined. Once setup, you will be able to easily view the repository and create new projects of your own (I do recommend that you develop your game using NetBeans; it is a very nice and powerful integrated development environment - IDE). Secondly, the structure and core classes within the repository will be defined.

I appreciate that this might be the first time you've got your 'hands dirty' installing and configuring a Java environment. It will be a useful experience (and hopefully not too traumatic). If you do run into problems that you can't resolve please do contact me (contact details can be found at the back of the Module Handbook).

Software Requirements

You will need the following:

- Latest version of Java SE bundled with NetBeans (about 150MB in size) available from
  http://java.sun.com/javase/downloads/netbeans.html

  At the time of writing this is 'JDK 6.0 Update 2 with NetBeans 5.5.1'

- The CSC207 Code Repository available from the CSC207 course page on Queen's On-line

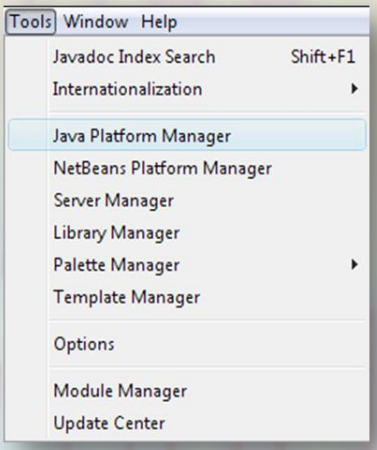
Java and NetBeans Installation / Setup

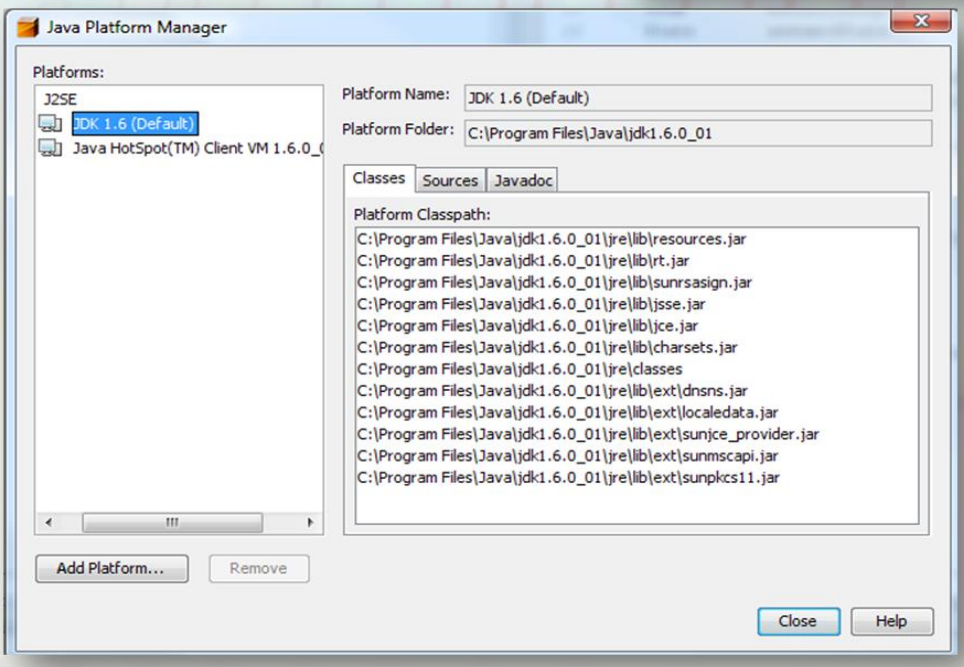
Please note: If you are using the machines available within the First Floor BCB Lab then you don't need to worry about installing either NetBeans or Java 1.6 (it is already installed) and can skip forward to the next section. In other words, steps 1 and 2 are really only intended for you to install the software on your own machine.

Step 1: Install JDK6.0 with NetBeans bundle (the default installation options will likely suffice)

Step 2 (Optional): Check that NetBeans is using JDK6.0

- Start NetBeans
- Once NetBeans has loaded, select the 'Tools' menu and then pick the 'Java Platform Manager' option. A dialog box will appear. Confirm that JDK1.6 is the default platform that will be used by NetBeans (we'll get onto how we can change this shortly).
- If for some reason JDK1.6 is not listed, you can add it by clicking on the 'Add Platform' button and navigating to the directory where JDK 6.0 was installed. Click on the JDK 6.0 folder and then hit 'Next' (Aside: on my computer this was C:\Program Files\Java\jdk1.6.0)
- Click 'Finish'



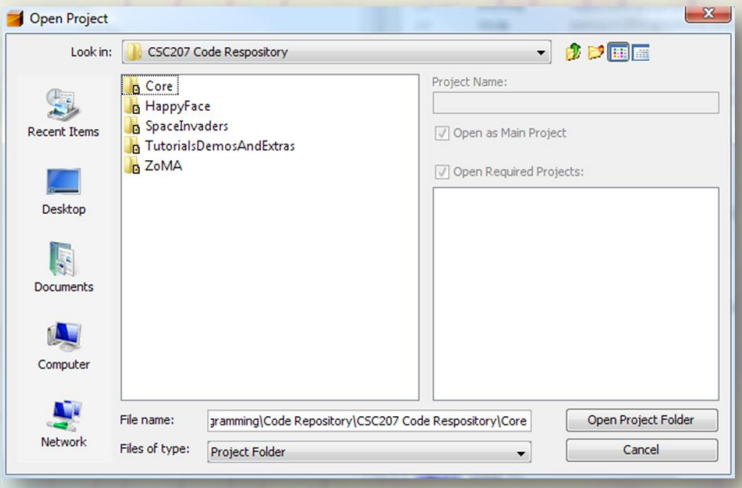


CSC207 Code Repository Installation

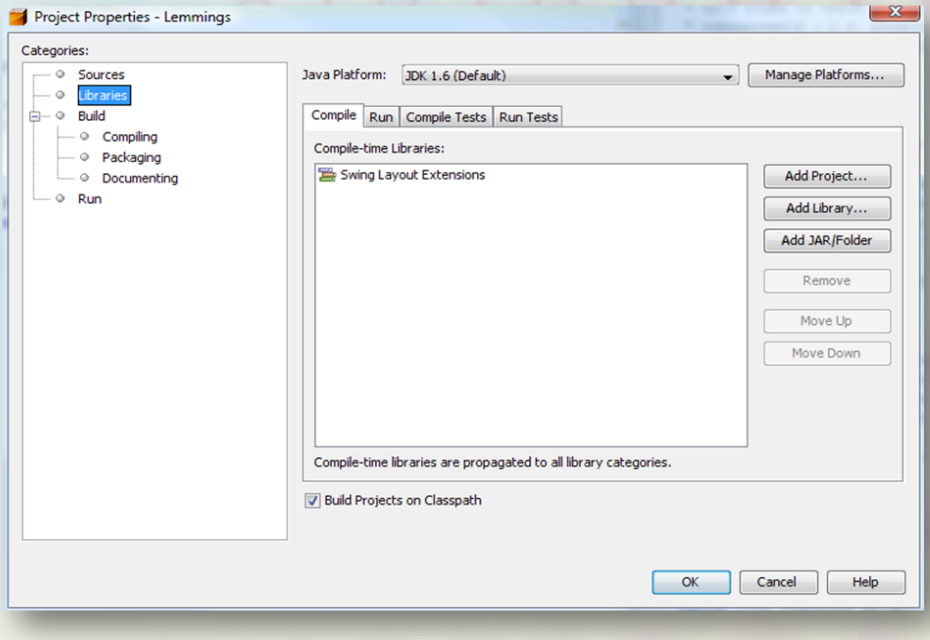
Step 1: Download and unzip the code repository from Queen's Online (for example onto your own computer or onto a memory key if you are going to use the computers in the lab).

Step 2: Start NetBeans. From the 'File' menu, select 'Open Project'.

Step 3: Navigate to where you unzipped the code repository. Go inside the '110CSC207 Code Repository' folder. Within this you should see a total of five projects that you can open. To open a project, click on its name (you don't need to go inside the project folder) and once the folder is highlighted, click on the 'Open Project Folder' button. You should open all five projects (i.e. when one project has been opened, repeat this step for the next project – although, have a read of Step 4 first!)

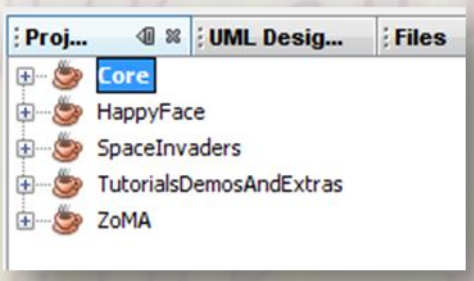


Step 4: **Important:** The first time you open the project you may get an error message about conflicts in the project. If you do get an error, it simply means that the JDK6.0 installed name and directory needs to be updated (i.e. the project comes along with *my* JDK6.0 locations, which may not be the same on the machine you are using). To correct this, right click on the project name in the Projects window (e.g. 'Core', 'ZoMA', etc.). At the bottom of the pop-up menu, click on 'Properties'. This will open a dialog. Click on the 'Libraries' category and then select Java 1.6 JDK to be the platform for this project. Problem solved.



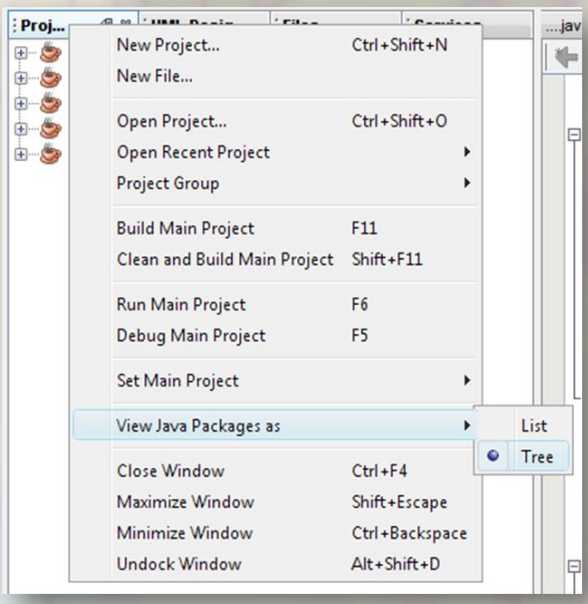
Step 5: Before you can run any programs within the repository you will need to build the project (i.e. compile the Java files, copy the image and sound assets into the correct build directories, etc.). In order to build the project, select the 'Clean and Build Main Project' option from the 'Build' menu. If all goes well, you should see something similar to the following message 'BUILD SUCCESSFUL (total time: 4 seconds)'.

What's within the Code Repository?



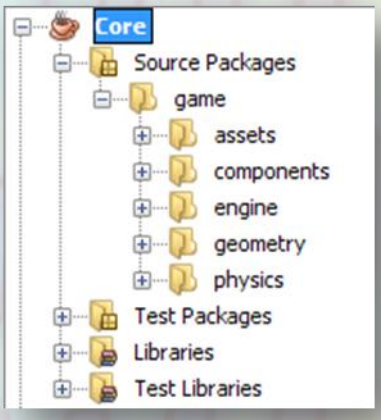
When all the CSC207 projects have been opened within NetBeans you should be able to see a total of five different projects. Each project can be expanded (click on the '+' icon) to show the Source Packages which, in turn, can be expanded to show the packages within the project (this is where our code and graphics, etc. will reside).

Below, I've provided a brief overview of the contents of each project – over the course of the module you will hopefully become familiar with a lot of the code that is contained within the repository, although please don't worry if all of this appears somewhat daunting at the moment – I purposefully wish to provide you with exposure to larger, more complex programs that you will likely have encountered during Level one modules.



Aside: In the following screenshots I've setup NetBeans to view packages in a 'tree-like' manner (which helps when the project is large). If you wish to use this view, right click on the 'Projects' tab and then select 'View Java Packages as:' and finally select Tree.

The 'Core' Project



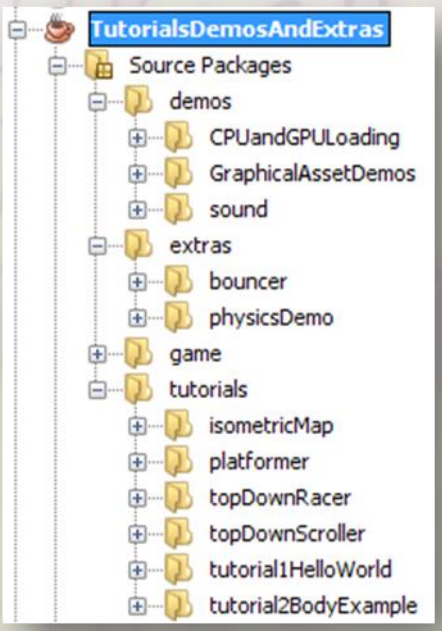
The 'Core' project does not contain any runnable programs, instead it provides a number of packages that hold the 'core' classes which the example games build upon. As such, it is better regarded as a library of classes and methods that can be used to help build a game.

Lectures will explore different aspects of the core classes from both algorithmic and functional perspectives.

An overview of the different packages can be found below:

- **game.assets** – this package contains all the various graphical and sound asset classes, including an asset loader and asset manager
- **game.engine** – this package contains the game engine and other key classes, e.g. game layer, game object, etc.
- **game.physics** – this package contains an impulse-based physics simulation that can enable game objects to collide and respond to collisions in a 'believable' manner.
- **game.geometry** - this package contains a small number of classes that can be used to describe the geometry of a game object (the geometry is used to test for overlap between two game objects and is also used within the physics simulator).
- **game.components** - this package contains a number of useful extended game objects that can be used for providing buttons, textual elements, etc.

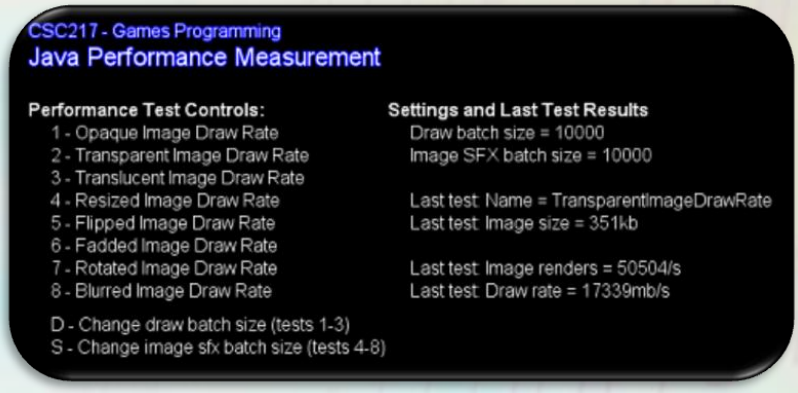
The ‘TutorialsDemosAndExtras’ Project



As you will doubtlessly pick-up from the unsubtle name of this project, the project contains a number of tutorials, demos and extra bits and bobs. In particular:

demos.CPUandGPULoading

This package provides a number of simple programs intended to measure timing and CPU/GPU performance.



demos.GraphicalAssetDemos / demos.sound

This package contains a number of programs that illustrate some of the graphical and sound asset classes contained within the core.assets package.

extras.physicsDemo

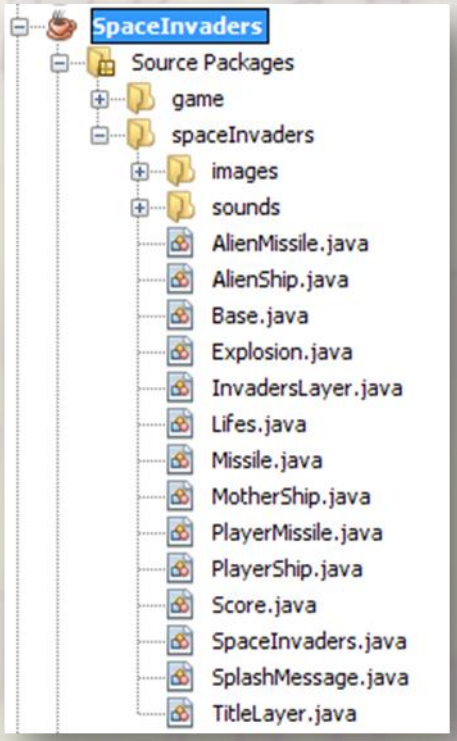
This package provides a number of physics based demos of the classes contained within the core.physics package. The physics code is based on that developed by Erin Catto (http://www.gphysics.com/downloads)

tutorials.isometricMap / tutorials.platformer / tutorials.topDownRacer / tutorials.topDownScroller

This package holds a number of tutorials that include four simple and common 2D game types: isometric tiled games, top-down racing car game, top-down scrolling shooter and side-on platform game. The 'games' are really only intended to provide an illustrative starting point towards constructing the game type (other faster approaches exist). The code contained within the tutorial packages should be read in conjunction with the associated tutorial document – which explores some of the issues involved in building the game.

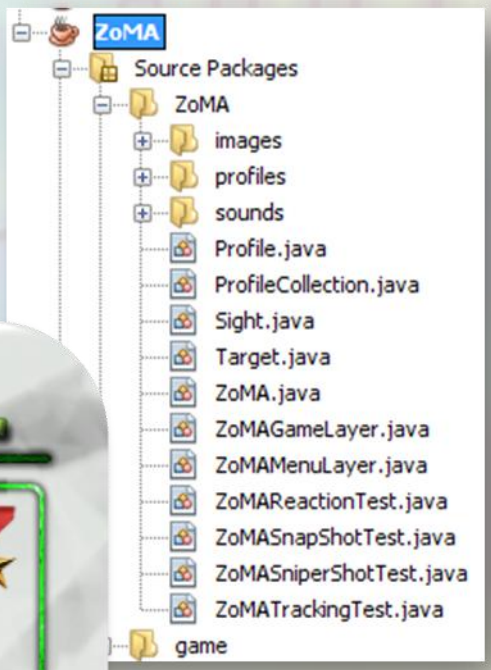
The ‘SpaceInvaders’ Project

Surely you don’t need to be told what’s within this project? Suffice to say, we will be exploring Space Invaders within the lectures from the perspective of how we might go about designing and implementing the game.



The ‘ZoMA’ Project

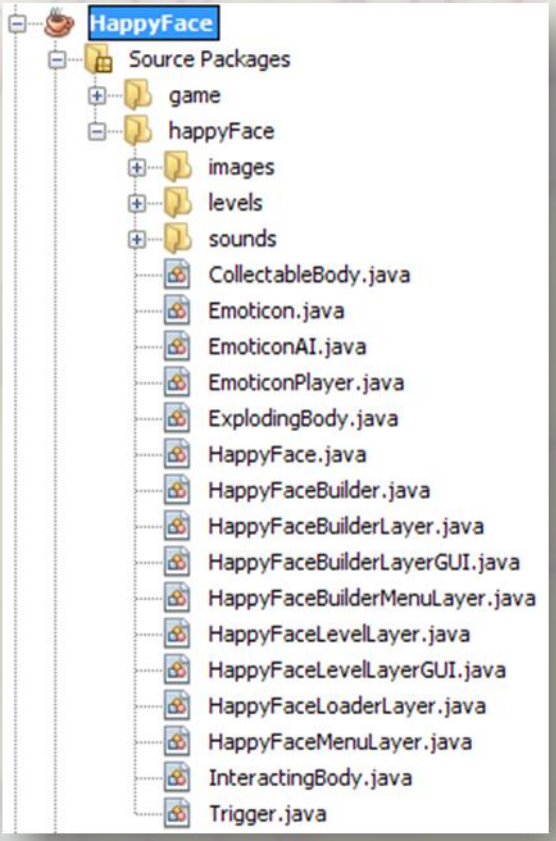
ZoMA is one of two larger example games provided within the code repository. The game, should you wish to classify it as a game, involves testing and measuring the player’s mouse reaction, aiming and tracking skills. From an educational point of view, the code hopefully provides some examples of GUI construction and a design that facilitates the introduction of new test types.

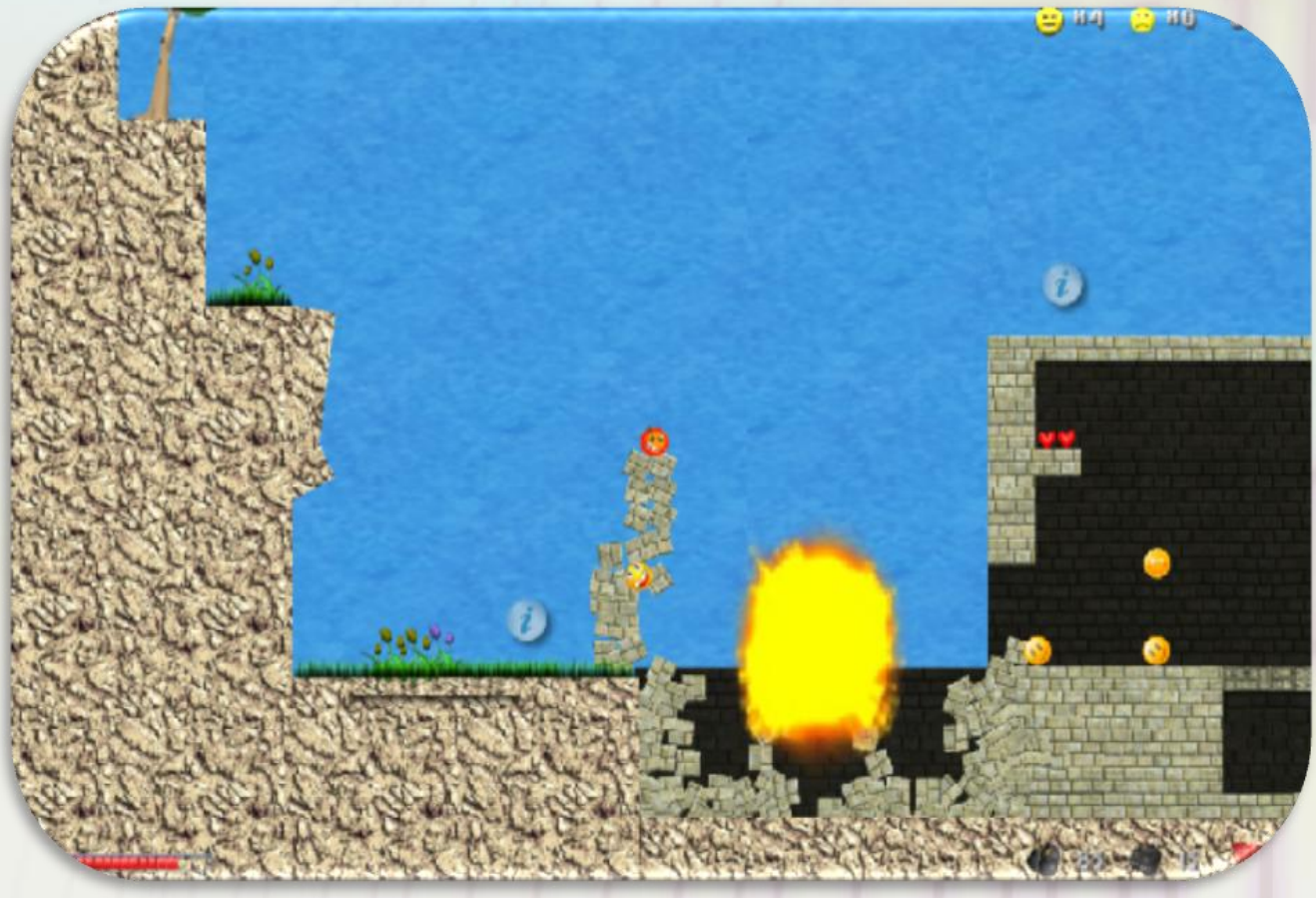


The ‘HappyFace’ Project

The HappyFace project is the largest project contained within the repository (by a good margin). The game is a platformer that makes extensive use of destroyable scenery. A level editor is also included that can be used to create new levels.

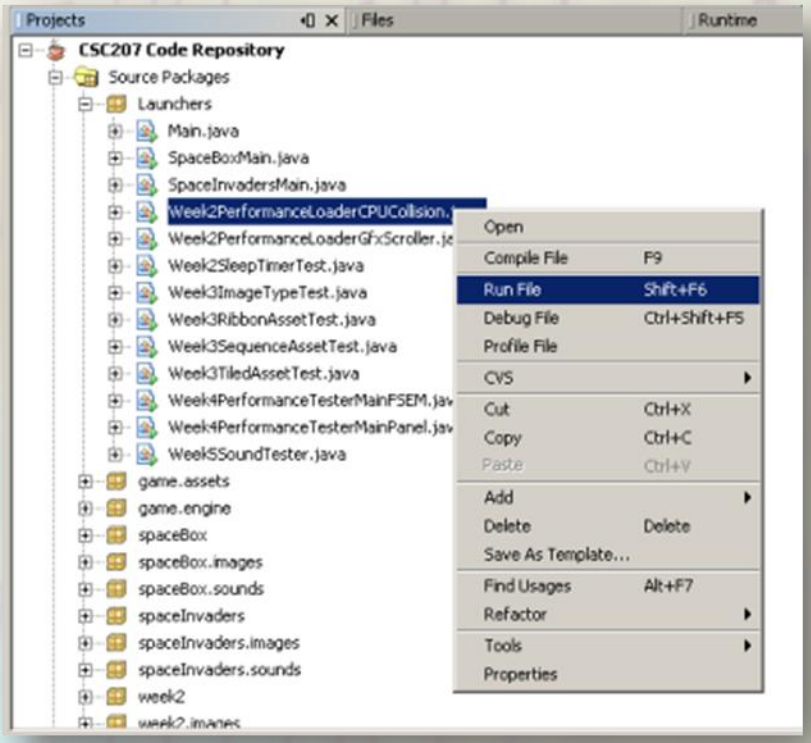
From an educational perspective the game provides an example of the physics classes in operation and a setup that involves loading in a number of different game levels.





Running Programs from the Code Repository

A NetBeans project can contain any number of different Java classes with main methods, i.e. runnable Java programs. NetBeans will indicate that the file is runnable by including a small green arrow on the file icon. In order to execute the file, right click on the filename and then select 'Run File' from the popup menu.



As an aside: Each project can also have one main class (which will be automatically executed if you select the 'Run Main Project' menu option from the Run menu). You can change the main class by right clicking on the project name, selecting Properties and then selecting the Main Class from within the Run tab. You will need to do this if you wish to setup your game so it can be automatically run from a JAR.

Getting to grips with NetBeans

My best advice is to start playing with NetBeans – it's a 'proper' IDE which means that it's both very powerful and will have an associated learning curve. However, once you do get to grips with it you will find that Java development becomes a lot more straightforward using the built in tools that NetBeans provides.

You can view a Quick Start Guide by clicking on the 'Help' menu. Additionally, a more comprehensive user guide is available (also accessible from http://www.netbeans.org/kb/55/using-netbeans/index.html)

Based on the feedback from last year, I've included some hints and advice below that should hopefully resolve some of the common problems:

Build/'Clean and Build' : Whenever you clean and build a NetBeans project a build folder is created (in the same directory that contains the src folder). All Java files contained with the source folder are compiled and their class files copies across into a corresponding folder in the build directory. Any images, sounds, text files, etc. contained in the source folder are also copied across into the build folder. Hence, whenever you add new images, etc. into one of your source folders you should do a 'Clean and Build' to make sure that the new image, etc. gets copied across into the build folder. If you don't, then whenever your code goes looking for the image, etc., it won't be able to find it!

JVM Parameters: The Java runtime (i.e. the thing that actually runs your Java programs) has a number of configuration options that control how the JVM uses memory, hardware, etc. You might hope that the default configuration options will suffice – however, this may not be the case. All of the projects contained within the code repository set a number of JVM options. I've provided details of some key options below:

- -Xms32m - Set the initial heap size to 32MB
- -Xmx1024m - Set the maximum heap size to 1GB
- -Dsun.java2d.opengl=true - Force Java to use OpenGL rendering
- -Dsun.java2d.d3d=true - Force Java to use DirectDraw rendering

You can view/set the JVM parameters of a project within NetBeans by right clicking on the project, then selecting Run and updating the VM Options field.

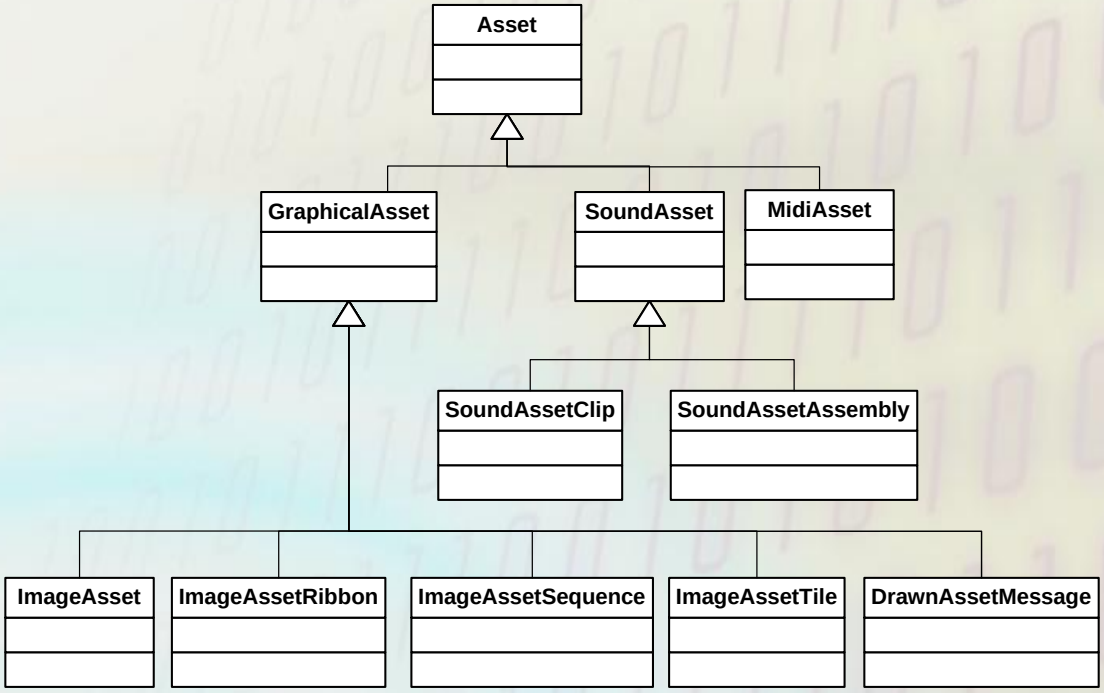
Important: You should try setting both the OpenGL and DirectDraw options (although only one at a time!) to see which provides better performance. This is particularly true if you are using Vista where driver compatibility and the more stringent Vista driver model may entail that performance differs considerably between the two options.

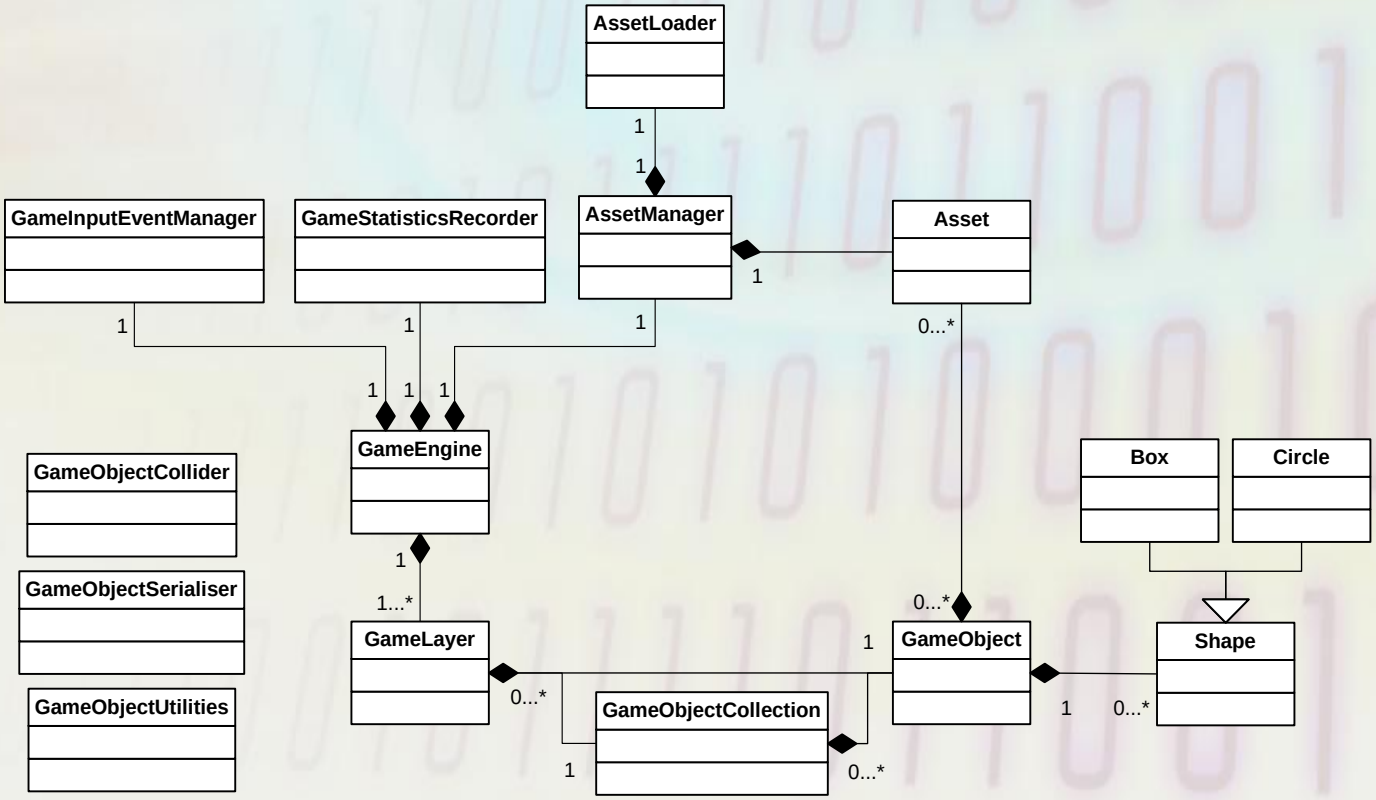
Hint: Whenever you create a new project, copy across the settings used in another project (e.g. from one within the code respository).

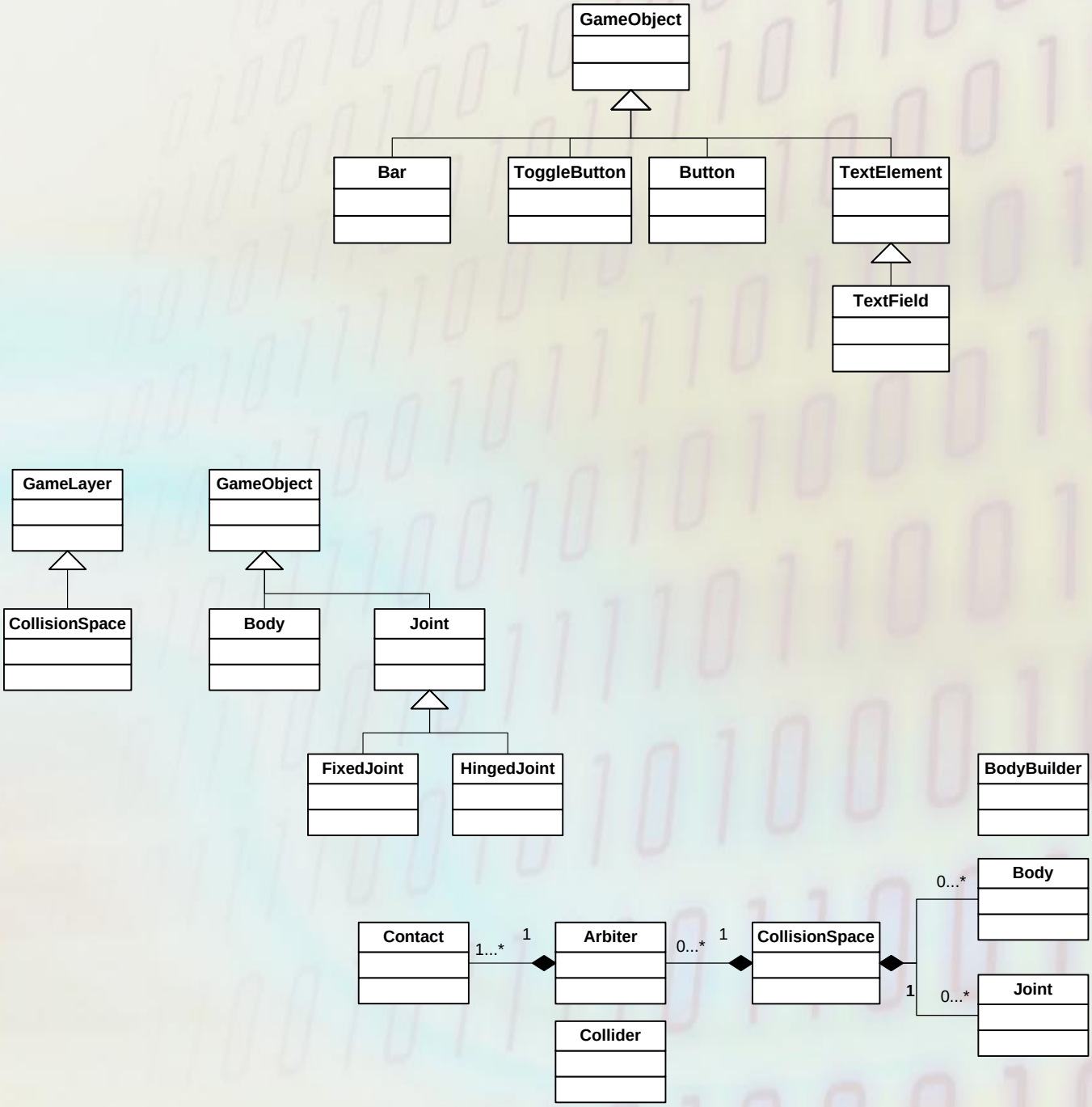
Libraries: I've been a bit naughty with the projects contained within the code repository. As indicated in the text, the 'Core' project should really be regarded as a library that can be 'imported' and used within other projects. It's actually very easy to do this within NetBeans, however, I've opted to include a full copy of the contents in each project, i.e. each project has it's own copy of the core game packages. Why? Well, it makes it easier to pursue the code as you see the complete codebase.

Core Game Class Structures

I've included UML class and composition diagrams below for the core packages Hopefully it will help direct any investigation of the classes.






GameObject
Bar
ToggleButton
Button
TextElement
TextField
GameLayer
GameObject
CollisionSpace
Body
Joint
FixedJoint
HingedJoint
BodyBuilder
Body
Contact
1...*
1
Arbiter
0...*
1
CollisionSpace
0...*
1
0...*
Joint
Collider