

Build a 2D Platformer

Using Java

03/22/2014

nyguerrillagirl@brainycode.com

lafigueroa@brainycode.com

Note: This is a working/draft version

Copyright © 2014 brainycode.com All rights reserved.

Permission is **not** granted to copy and distribute an electronic version of this document. Please direct links to this website or request permission.

Permission is NOT granted for commercial use.

Contents

Chapter 1 – Getting Started/Introduction.....	6
Purpose.....	6
What do I bring to this endeavor?	7
What do you bring to this endeavor?	7
Why build your own game engine?	8
Does it make sense to develop games using Java?	9
Best Practices - revisited.....	33
Quick History of 2D Games and Platformers	10
How can I play older games (no longer available for sale) or in the arcades?.....	24
Writing a Design Document	31
Game Design Document Outline	31
Using Best Practices	33
What is JUnit?.....	33
What is Ant and Maven?.....	33
What's the plan for learning how to use these tools?.....	34
Game Engines	34
The Plan	34
Overview of the Plan	34
Exercises	35
Chapter 2 – Graphics	37
Applications vs. Applets	37
Organizing the code	37
Lab 2-1: Creating a program that displays a window and runs as either an application or applet.....	38
Drawing	53
Color	53
drawstring().....	54
Centering the Text.....	55
Lab 2-1: continued	56
Drawing Lines.....	57
Fonts.....	58
Lab 2-2: Drawing a Star Field.....	59
Random Number Generation	61

Drawing Ellipses and Circles.....	62
Drawing Rectangles	62
Additional draw methods	63
drawArc.....	63
Lab 2-3: Draw a Happy Face	65
Drawing Polygons.....	66
Drawing Images	68
Lab 2-4: Drawing an Image	69
Clipping.....	71
Graphics2D	72
Making changes to the objects we draw	72
Alpha.....	72
Anti-aliasing.....	72
Scaling Images	72
Chapter 3 - Introducing the Game Loop.....	73
Process Input.....	73
Simulate Game World.....	74
Render.....	74
What are Threads?	74
Why do we need Threads in our game program?	75
Creating a Thread.....	77
Lab 3-1: A Quick Introduction to Using Ant	79
Lab 3-2: Bouncing Ball.....	84
Lab 3-3: Displaying an animated GIF.....	102
Keyboard Processing	122
Creating Pong.....	122
Creating Breakout.....	122
Creating MindSweeper	122
Bibliography	124
Appendix A: Web References.....	125
Appendix B – Using Eclipse	126
Appendix C – Using JUnit (test-driven development).....	127
Directory Setup	127

Imports	129
Creating a test case.....	130
Appendix D – Ant.....	150
Appendix E – Setting up on a Linux Machine	150
Appendix F – Setting up on a Macintosh Machine.....	150
Appendix G – MAME	151
Obtain and Install MAME	151
Obtain Game ROMS.....	151
Configure your system.....	152
Keyboard Commands to start and control a game	153
Appendix H – Using Ant	154
Appendix X – Pong, Breakout and MindSweeper.....	165
Pong	165
Breakout.....	165
MindSweeper	166
Appendix X – The Top 25 Platformers you should check out.....	169

Chapter 1 – Getting Started/Introduction



Figure 1 - <http://arcadesushi.com/top-25-best-2d-platformers/>

Purpose

I¹ was motivated to write these notes after completing a set of youtube videos detailing the creation of a one-level simple 2D platformer from beginning to end in Java. At the end of the videos I had an interesting start of a game with delightful graphics and interesting mechanics but I felt unhappy with the fact that the code and techniques were not explained in sufficient detail to provide me a sense of deep satisfaction or accomplishment. For most of the time I followed the videos I felt more like a code monkey² than a developer and I am sure I had a lot of company.

I think the key piece missing for me was a big picture view of what we were programming towards. I know the video presenter knew the direction he was working towards but when you are trying to bring along random viewers of various backgrounds and experience it pays to draw a big picture or help to develop a conceptual model of what is going on and why. In addition, it also helps to occasionally take a more in depth look at the methods being used and how the algorithms work so that the developer is comfortable extending them or knowing where to look when something does not work.

The purpose of these notes is to present and build in a logical progression the skills required to build a 2D game and specifically a platform game in Java. I want you the reader to walk away with a deeper understanding of what it takes to build a 2D platform game as we delve into the details and build up knowledge in a logical progression. The programs will go from simple single-screen games to specific 2D platformers and finally into a more general/generic development tool to support any modern 2D platform game you wish to build.

By the end of reading these notes you will have completed several classic 2D games and well on our way to our own generic game engine.

¹ Nyguerrillagirl

² Code monkey is a derogatory term for an unskilled programmer.

Who are we?

Nyguerrillagirl – L. Figueroa

I have been programming with Java for quite some time and have been trying to incorporate the use of best practice techniques in all the programming I do – recreational and professionally. Some of these techniques involve the use of dependency injection in order to make the code flexible and testable. The use of build tools like Ant, Maven or Gradle to manage dependency of external libraries as projects get bigger and have many moving pieces and projects. In addition, the use of JUnit to test as many of the key functions and algorithms as you build the program. I will be using as many techniques that seem practical given the difficulty of the task at hand. That is, I will not use tools for simple projects that do not require it but as our Java Projects grow I will start to include various best practice techniques.

I am a professional programmer and have always pursued a strong interest in the techniques involved in building games.

lafigueroa – Q. T. Rodriguez

TBD

Who are you?

The only prerequisites to getting started with these notes are the following:

- Knowledge of Java that would be obtained in a one semester college course
- The use of Eclipse for software development (see Appendix B if a quick tutorial is required.)

If you need to learn or refresh yourself on Java then I recommend the following free online sources:

- ✚ Java Language and Virtual Machine Specifications at <http://docs.oracle.com/javase/specs/>
 - This is not intended to be used by those who want to learn Java but more as a reference or to lookup areas you are not familiar with once you have gone through a course or one of the books suggested in this list.
- ✚ Thinking in Java - <http://www.mindviewinc.com/Books/downloads.html>
 - This too is less of a “learn Java” book but is a great book to read while learning Java to get into the Java frame of mind
- ✚ Introduction to Programming Using Java, 6th Edition, <http://math.hws.edu/javanotes/>
 - This is an excellent free e-book that could be used in any college course

You should also know your way around the Eclipse IDE. I highly recommend the following tutorials for more information:

- ✚ Eclipse IDE Tutorial: <http://www.vogella.com/articles/Eclipse/article.html>
- ✚ Video tutorials: <http://eclipsutorial.sourceforge.net/totalbeginner.html>

I will always provide detailed instructions for how to set up or use an Eclipse feature for the first time but going forward from that point I will not repeat the detailed steps. In addition, all my screen shots and explanations will come from using Microsoft Windows versions of all products. There will be an appendix for how to setup and run on a Macintosh or Linux programming platform.

Learn more about building games – use commercial game engine?

You are reading these notes to learn how to build a game and a generic game engine but you should also invest some time using one of more of the freely available game engines you can obtain online.

If you were only interested in getting a game prototype up and running then tools such as GameMaker, Construct, or Multimedia Fusion 2 will do a very good job. But they also prove to be useful tools to learn the mechanics and concerns around building your own game and game engine. In order to gain an appreciation for what it takes to build your own game engine you should get acquainted with one or two free tools available on the Internet.

The point of these notes is to learn what it takes from a programming perspective to build your own 2D game engine in order to build various games. It will allow you to develop a deep understanding of what these engines are trying to accomplish and prepare you to move on to creating or using a 3D game engine.

A very good one that I highly recommend is GameMaker Studio or the previous version Game Maker 8.1. A game engine allows you to put together your game images (sprites), sounds, music, background, and build levels. You can also write code to create and manage the various objects that make up your game.

Here are some screenshots from the game I developed using the art and instructions in the highly recommended book “Game Maker’s Apprentice”³,



Figure 2 - Game Maker Game - Pyramid Panic!

This is a game where you an explorer trapped inside a large pyramid level. You try to collect treasures placed throughout the level as you dodge mummies, scorpions and other dangers. This game contains some classic game mechanics as in being able to collect a special item which enables you to temporarily immobilize the persistent mummies. It also provides a top-down view as shown in the next figure of the level.

³ This is the best book I have ever encountered with respect to professional looking game graphics and wonderful game ideas. I highly recommend the book.

The great benefit you get from a programming point of view in using one of these game development tools is an understanding of the complexity and the components that go into building and using a game engine toolkit.



Figure 3 - Playing Level 1 of Pyramid Panic!

There are many programs like Game Maker that provide the capability to create entertaining games. I highly recommend you try one or more in order to get a feel for what your own game engine will need in order to allow users to build a generic platform game.

Does it make sense to develop games using Java?

A very good question to pose: "Is Java the ideal language to create games?" At one time I would have certainly said – no way. Why? Java did have a reputation for being slow and rather awful at building applications with graphical components. In addition, the fact is the most popular and practical language used today to create video games is the programming language C++ not an interpreted language like Java. But the popularity of games such as Minecraft and the number of Java-based games written on Android mobile devices have shown that Java has matured to provide programmers the capability to create 2D or 3D games. Since the original release of Java there have many advancements with the development of Swing components, Java 2D API, just-in-time compilation and various support libraries to support the development of fast full-screen Java applications.



Figure 4 – Minecraft

The question really isn't why develop a game in Java – but why not! The fact is that you can install the Java development kit on Windows PC, Linux and Macintosh machines. The tools or libraries that you will be using are available on all these systems and are free. The concepts and ideas we will be developing are transferrable to other programming languages. It makes sense to use Java!

Quick History of 2D Games and Platformers

Material from: http://en.wikipedia.org/wiki/Platform_game and
http://en.wikipedia.org/wiki/Video_game_genres

What type of game?

There are many types of games one can build these fall under various game genres.

Action - Ball and Paddle

One of the earliest games ever developed was one similar to an electronic form of tennis names “Tennis for Two” in 1958. Our first planned game will be a simple version of Pong and a Pong variant – Breakout. There have been numerous variations of the ball and paddle game developed since its first inception.

An interesting issue when developing such a game is what to do for the single player – introduce a computer opponent of course. The area of creating computer opponents, enemies, or friends requires the development of a topic area we will not cover in this book – artificial intelligence. We will implement simple algorithms to simulate computer opponents.

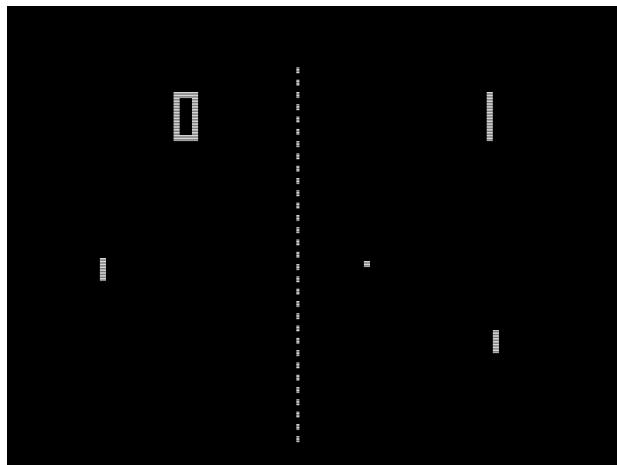


Figure 5 - First commercially successful video arcade game - Pong! (1972)



Figure 6 - Mario Tennis (2000) for the N64

Action - Beat'em up and hack and slash

This is a game where the player has many one-on-one battle encounters with many computer-controlled enemies. The beat'em up is usually hand-to-hand combat and in the hack and slash you yield weapons like swords. The genre was very popular in the 1980's (e.g. Kung-Fu Master) and still exists today but is combined with other game mechanics such as side-scrolling and puzzle mechanics. A very popular recent addition to the genre is Castle Crashers originally released on the Xbox Live Arcade but made available on other platforms.



Figure 7 - Beat 'em up Kung-Fu Master (1984)



Figure 8 - 2D beat 'em up (2008)

Steam recently (2012) released a version of the game. The game supports cooperative gameplay for up to four players.

I would probably place the game “God of War” in the hack and slash genre.



Figure 9 - From "God of War III" (2010)

We shall develop a network based beat-em up to demo how to implement a cooperative game for a 2D platformer.

Action - Fighting Game

This is the classic one-on-one combat. The user has various combo moves available against a computer or human opponent. The game Karate Champ (1984) was a video arcade game that had the martial arts elements we typically associate with this type of game and would be considered a candidate for any top-20 list of the genre.



Figure 10 - Karate Champ (1984)

This genre is not quite as popular as it once was and modern versions of the genre such as Dead or Alive still have a strong following.



Figure 11 - Dead or Alive 5 Fighting Game (2012)

Action - Maze Game

Maze games are games where the entire playing field is a maze. There is usually a timer, monsters obstructing the progress of the player. The most popular game in this genre is Pac-Man. One of the earliest (and most difficult) was Rogue.

There are not too many games made in this genre today.

One of the earliest versions of this type of game is Serpentine. In this game you control a snake that tries to earn points by eating the enemy snakes that enter the level with you.



Figure 12 - Serpentine Maze game (apple version 1982)



Figure 13 - Pac-Man Maze Game (1980)

The player controls Pac-Man through the maze trying to gobble up the dots and avoid the four enemies (Blinky, Pinky, Inky and Clyde).

We will implement an Arcade and Atari 2600 version of a maze game titled Berserker.

From: [http://en.wikipedia.org/wiki/Berzerk_\(arcade_game\)](http://en.wikipedia.org/wiki/Berzerk_(arcade_game))

The player controls a green stick man, representing a humanoid. Using a joystick (and a firing button to activate a laser-like weapon), the player navigates a simple maze filled with many robots, who fire lasers back at the player character. A player can be killed by being shot, by running into a robot or an exploding robot, coming into contact with the electrified walls of the maze itself, or by being touched by the player's nemesis, Evil Otto.

The game is notable for being one of the first video arcade games to use synthesized speech.



Figure 14 – Berserker (Atari 2600 version 1982)

Action - Pinball Game

These games simulate the look and feel of an arcade pinball machine. An early (1982) pinball simulation was developed for the Apple II called David's Midnight Magic.



Figure 15 - Pinball Game circa (1982)

A more modern pinball game is Pinball Arcade by FarSight Studios.



Figure 16 - The Pinball Arcade Funhouse

Many of the pinball games are modeled after real pinball machines.

We will not get into this genre at all but those that are interested in exploring it are encouraged to investigate the freeware video game engine “Visual Pinball⁴.“ The software consists of an editor and the simulator to run your game.

Action - Platform Game

A platform game is a computer/video game involving a game character (Mario, Jumpman, Sonic, Eathworm Jim, etc.) walking, running, jumping or bumping between suspended platforms. The game avatar or character has to go over obstacles, jump over enemies, or perform any number of the various ways designers have developed to make it challenging for the player to attain the levels goal and advance through the game.

One of the earliest games fitting the description of a true platformer was the game Space Panic. It was an arcade game released in 1980.

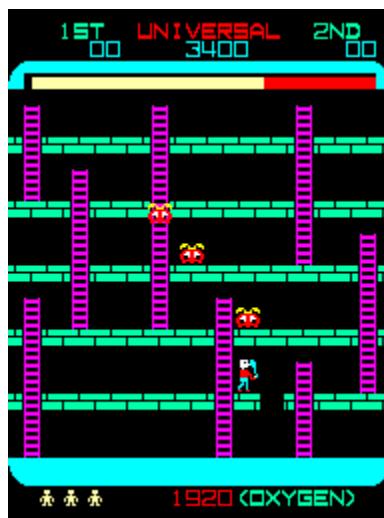


Figure 17 - Image of Space Panic (from emulator)

(see http://en.wikipedia.org/wiki/Space_Panic for more information)

In the game Space Panic the player does not jump but rather digs holes in the platforms at the same time he is trying to lure the alien enemies into them in order to trap them. The player then hits the alien trapped in a hole in order to remove them from the game. The challenge is to get the job done while the amount of oxygen is slowly being depleted. The game mechanics popular among many platform games is the use of ladders to maneuver between the platforms, a limited set of actions one can use to eliminate or escape the enemy which in this case is digging of a hole. A popular clone of the game was Apple Panic by Broderbund for the Apple II.

A similar and extremely popular game with the same premise was Lode Runner (1983). The game was also unique in that users (you and I) could create their own levels.

⁴ <http://sourceforge.net/projects/vpinball/>



Figure 18 - Lode Runner

In Lode Runner the avatar (seen all in white above) runs around trying to collect all the treasure placed strategically on the platforms. The player runs, suspends across hand-to-hand bars and climbs stairs all the while trying to avoid the guards. The player can create holes in the platform that temporarily traps the guards and allows the player to walk over trapped guards. This game was quite addictive and popular. It has been translated into many different platforms but I think the definitive version is the Apple II one.

The game Donkey Kong was the first platformer game to have all the elements we associate with a platformer – player jumps over obstacles and gaps. The game introduced two game characters that have stood the test of time – Mario (known as Jumpman in the game) our hero, and Donkey Kong, the brute who stole our damsel in distress.



Figure 19 - NES image of Donkey Kong

All the platformer games we have discussed so far were all single-screen games or what is more commonly referred to as games created with a static playing field. The evolution of platformers are in games that had levels that were longer than one screen but did not have the cool scrolling features yet.

One popular game demonstrating this effect was the game Pitfall! where the level had connected screens (not exactly a scroller in the true sense of the term.)



Figure 20 - Atari Pitfall!

The first game to have smooth horizontal scrolling⁵ was the game Jump Bug.



Figure 21 - Jump Bug

The game may at first appear unusual for those more accustomed to cartoonish avatars that are anthropomorphic (human-like in looks or behavior). In the game the player controls a car as it moves through various environments/levels jumping, falling, shooting, and collecting treasures.

The game that eventually became the archetype for the elements in a true 2D scrolling game was the game Super Mario Bros by NES.

⁵ http://en.wikipedia.org/wiki/Jump_Bug



Figure 22 - Super Mario Bros

The objective of the game was to of course save the damsel, Princess Toadstool, by rushing/running/spinning through the Mushroom Kingdom fighting the forces of evil (Browser and minions). Who has not played this game?

The popularity of platformers has declined since its heyday into the 16-bit era of the 1990's. That does not mean that game designers have not developed today innovative and intriguing platformers by introducing new game mechanics and twists. Once such game is VVVVVV (pronounced "vee" and/or six times pronounce "vee") that introduced a new game mechanic where the player does not jump but reverses the direction of gravity to go from platform to platform by falling either upwards or downwards.

From: <http://en.wikipedia.org/wiki/VVVVVV>

The player controls Captain Viridian, who at the outset of *VVVVVV* must evacuate the spaceship along with the captain's crew, when the ship becomes affected by "dimensional interference". The crew escapes through a teleporter on the ship; however, Captain Viridian becomes separated from the rest of the crew on the other end of the teleporter. Upon returning to the ship, the Captain learns that the ship is trapped in an alternate dimension (referred to as *Dimension VVVVVV*), and that the ship's crew has been scattered throughout this dimension. The player's goal, as Captain Viridian, is to rescue the missing crew members and find the cause of the dimensional interference.⁶

⁶⁶ <http://www.eurogamer.net/articles/vvvvvv-review>



Figure 23 - The puzzle-platformer VVVVVV (

The game has the look and sound of a Commodore 64 game but the smartness and comic sense of some the great game indie developers are creating for today's game audience.

Another game to check out is Braid a puzzle platformer that is now available on many different platforms (e.g. PC, Xbox, etc). It is has the classic game story – the fellow trying to rescue a princess from a monster. The game won the “Innovation in Game Design” award in 2006. The game avatar can run, jump and climb the many game levels that have a time-manipulation component (e.g. player can go back in time). The additional aspect of the game is how the story unfolds in text so we can see the motivation and reasons Tim (our hero) wants to save the Princess.



Figure 24 - Braid - Tim's search for the Princess

There is much to explore in this genre and the main goal is to develop several example games in this genre in addition to the beginning of a 2D game engine tool to build any 2D platform game.

Action - Shooter

A shooter game is combat with projectile weapons such as guns. The player usually has to acquire ammunition while playing the game. The goal is usually to get to the end of level or attain some goal without the main character dying.

Shoot 'em up

One sub-genre is shoot 'em up (shmup) game. Usually this involves the player in some spacecraft shooting at large incoming enemies. The most well-known (not necessarily the best) was Space Invaders.



Figure 25 - Space Invaders (1978)

Space Invaders is an example of a “fixed shooter” since the player only moves left and right and the enemies come down to attack. Another kind of shooter is a “rail shooter” where the player follows a specific route (as into the screen viewpoint). The player does not decide on the path but decides on what to shoot. As popular as Space Invaders was the shump that really expanded the genre was Galaxian (1979).



Figure 26 - Galaxian (1979)

Galaxian had great sound and color. The enemies came down in kamikaze waves. The games introduced several “firsts” for a video arcade game – different colored fonts, animated sprites and explosions, a scrolling star field. The game had a memorable look and feel and the game play stands up to the test of time.

The first shump to introduce side-scrolling was the game Defender. It was one of the first games to integrate the mini-map feature into the game in a significant way.



Figure 27 - Defender (1980)

We will cover how to include a mini-map feature into your game. This genre is not as popular as it once was but if you go back and play any of these games you will play for hours. Probably the best horizontal shump is R-Type (1987). You can still play today using MAME (see next section).



Figure 28 - R-Type (1987)

Racing Game

A genre I find interesting and challenging is the racing game. This genre has been around for a long time and can be re-created using 2D technology but requires a bit of math and physics to make it feel like a 3D game. A classic game in this genre is Pole Position (1982) released both as an Arcade and console machine by Atari.



Figure 29 - Pole Position (1982)

Role Playing Game

Puzzle Game

Real time Strategy Game

Simulation Game

Sports Game

Music Video Game

How can I play older games (no longer available for sale) or in the arcades?

We may have whetted your appetite for trying out some of the games from the past. You may be wondering if there is a way you can play some of these older games.



Figure 30 - AppleWin

Fortunately for us we will not need to buy an old arcade machine or acquire an Atari 2600, Apple IIe or NES machine to play any of the classic games or yore. You can use a software application called an emulator. “An emulator duplicates the functions of one system using a different system, so that the second system behaves like (and appears to be) the first system.” (Huh?!) What that translates to is that we will be obtaining software from the Internet to install on your PC that shall be software versions of the hardware machines of Atari 2600, Apple IIe, C64 and NES. For example, we will use the AppleWin program to emulate an Apple IIe.

All the tools and applications you will need are available free online. All you need is an Internet connection to obtain it. The website, www.brainycode.com will have all the tools you will need to complete all the exercises and programs in this book. I will even upload my copy of the emulators and ROMs I recommend you try in these notes.



Figure 31 - The game Hard Hat Mack (first EA game) running on AppleWin emulator

MAME – Arcade Game Emulator

The first emulator I recommend you get an install is called MAME which stands for Multiple Arcade Machine Emulator. You can download and install MAME from <http://mamedev.org/>. An emulator for consoles and computers that you can obtain from the same web site is MESS (Multi Emulator Super System). See Appendix E for instructions on how to install and use MAME. MAME provides you the capability to play arcade games such as Donkey Kong, Space Invaders, Galaga, etc.

AppleWin, Stella and VICE – Consoles and Computer Emulators

There are very good emulators for playing older Apple IIe games, Atari 2600 games and Commodore 64 games. The emulators are AppleWin, Stella and VICE respectively. The best thing you can do is to download the emulators and beg, borrow or (gulp) procure the best games from all these machines. It will be insightful experiencing gaming as the games were originally developed and to see the manner in which the best programmers squeezed every bit out of these machines to give us such delightful games.



Figure 32 - Castle Wolfenstein

You can also pick up emulators for older consoles – NES, SNES, Sega, Nintendo 64 using the emulators FCEUX, SNES9X, Kega Fusion, and Project 64, respectively.

Best Apple II games⁷

- **Lode Runner** – This is our classic platformer that was developed in the 1980's by Doug Smith.
- Ultima IV – released in 1985 this was the fourth version of the role-playing video game.
- Bolo – a 1982 game that amazingly generates a random rectangular maze containing enemy bases. You the player control a tank and must destroy the enemy bases in order to advance to the next level.
- Robotron – This wonderful game was developed for the Apple II from the arcade game Robotron: 2084. The game takes place on a single-screen where Humanoids and Robotrons are randomly placed. Your job is to save the Humanoids. The game is fast and intense as you dodge, run one way and shoot in another while all these Robotrons are focused on wiping you off the screen. This is similar to the game Space Invaders in that it was unwinnable but who cared – it mattered more that you put in the time and lasted as long as possible⁸.
- Choplifter – This 1982 game was developed for the Apple II. The player navigates a combat helicopter to save hostages while taking on hostile tanks and other enemies.
- Castle Wolfenstein – This is a stealth-based action adventure created by Muse Software for the Apple II. The goal is to find secret war plans and escape alive. The game is played from a top-down perspective even though the player was viewed from the side as in a scroller. This game influenced the game later developed by Id – Wolfenstein 3D.

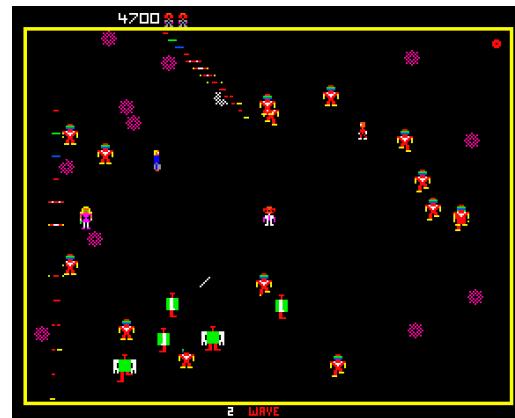


Figure 33 - Robotron

⁷ From <http://techland.time.com/2010/01/06/the-10-greatest-games-for-the-apple-ii/> and <http://techland.time.com/2010/11/29/lets-go-retro-best-computer-games-from-the-80s/>

⁸ I spent many hours on an Apple IIe playing this game. You can play the arcade version using MAME or the Apple version using AppleWin..



Figure 34 - Battlezone on Apple II

- Star Blazer – release in 1982 by Broderbund this game was a five level side-scrolling shump⁹. This is a great game to try to duplicate using the techniques we will cover in these notes.

- Wizardry – This is a role-playing video game. It was a great influence in the development of early console RPGs.

- Battlezone – This is port of the classic arcade game of the same name. The arcade

version was notable for using vector graphics and taking you the player inside a tank as you searched out for enemy tanks.

- Archon – this was a creative chess-like game, where a removal of a piece required a one-on-one battle.

One reason you should try out the top games from arcades, computers and consoles of the past is because the games varied in gameplay and creativity. Unlike today where games tend to fall into well-known and popular genres games of the past had more diversity and creative elements that is usually found today in indie low-budget games. The depth of diversity is generally applicable for any art form that makes its seminal debut. It was in the late 1970's and 80's that game genres such as platformers, car-racing, puzzle, adventures, role-playing, etc were first invented.

I highly recommend you try all the games out.

My personal favorite Apple IIe game is not on any top ten list – Crisis Mountain – a strategic arcade action game I played for hours!

⁹ Shump – shoot ‘em up



Figure 35 - My favorite Apple II platform game

From: http://www.atarimagazines.com/compute/issue40/review_crisis_mountain.php

The scenario of the game is that a group of terrorists was hiding out in the caverns of a dormant volcano in the Pacific Northwest. The volcano erupted unexpectedly, forcing the terrorists to abandon their hideout. As they fled, they left behind their loot and supplies - and several nuclear bombs. To save the West Coast from impending disaster, *you* must venture into Crisis Mountain, dig up and defuse the bombs while avoiding numerous hazards.

The Atari 2600 is responsible for bringing arcade magic and fun into households in the form of a game console. It was more limited in power and range than a computer but nonetheless provided hours of fun in converting arcade favorites or coming up with original game play and mechanics.

Best Atari Games

- **Pitfall!** – This was one of the top-selling Atari 2600 games. It was NOT made by Atari but by Activision a company that was one of the first third-party developers for a console. The company was started by disgruntled Atari developers who wanted to share in the success (read millions of dollars) their games were making for Atari. This game was one of the first maze-like jungle games where the player moved from screen to screen. It was a remarkable technical achievement given the limitations of the Atari 2600.



Figure 36 - Adventure

- Missle Command - This is a port of the arcade game with the same name. The game is considered one of the most notable games from the *Golden Age of Video Arcade Games*¹⁰.
- Super Breakout – This is a cool update to the game Breakout (see Appendix XXX).
- Adventure – This is considered the first action-adventure game and would be on anyone's top-100 games of all time. Playing the game today where the player is represented by a small square and the dragons trying to foil your adventure look more like a duck one can wonder how this game was responsible for many hours of lost time and fond memories. (See Figure 36)
- Combat - This was one of the launch title games for the Atari 2600. The game had many variations but it was basically two tanks on a playing field shooting at each other.
- Space Invaders – The arcade version of this game was responsible for a shortage of coins in Japan at the height of its popularity. This is a good port of the game to the 2600. Many gamers today may scratch their heads wondering “huh!? This was a hot game?” Yes it was!
- Dig Dug – This was a popular port of the arcade game with the same name. The objective is to eliminate underground-dwelling monsters by inflating them or dropping rocks on them. The player (Dig Dug) tries to avoid the game enemies.
- Pole Position – This is another good port of a Namco aracade game. It is a racing game where the player controls a Formula One race car.
- Frogger – This is a port of the popular arcade by the same name. The objective is to navigate frogs across a busy highway.
- Warlords – This is a port of an Atari arcade game that combined the elements of Breakout and Quadrapong.

Again, my personal Atari 2600 favorite does not make anyone's top ten (or twenty list) Night Driver.



Figure 37 - Atari Night Driver

I also enjoyed many hours playing Adventure, Combat and many other Atari 2600 games.

Note: I highlight the games I personally loved on some of the computers/consoles because I intend to re-create them as part of these notes.

¹⁰ http://en.wikipedia.org/wiki/Golden_Age_of_Video_Arcade_Games. This is the peak era for video games that warranted a Time magazine cover and concern from parents all over the world about how their kids were spending their time and money!!

Playing NES games

The Nintendo Entertainment System (NES) singlehandedly resurrected console game playing in 1985 with the introduction of many classic platform games and characters that still live on to this day. It was the best selling console for its time.

You can find very good NES emulators online. I highly recommend the website console classix at <http://www.consoleclassix.com/> to play any NES¹¹ games. Many of the top platformers of all time were done on the NES. You should try out the top ten NES platform games:

- Super Mario Bros 3 – This game has all the classic elements we associate with a platformer.
- Megaman II – This is action style platformer with entertaining value to this day
- Contra/Probotector/Gryzor – This is a classes NES game with action and great character control
- Bionic Commando –This is an action platformer game based on the 1987 arcade game by the same name.
- Kirby's Adventure – A platformer consisting of seven worlds each with a boss fight and a Warp door to travel to another world. Kirby was able to walk, run, jump and attack or dodge enemies. This was a late generation NES that featured and used all the power the NES had showing off the best graphical effects.



Figure 39 - Little Samson

characters assistance at various levels.

- Metroid – This is regarded as a action-adventure game where the player controls the character Samus Aran. The game was notable for revealing at the end that the protagonist was female.
- Journey of Silius – This is more of a side-scrolling run and gun game released in 1990.



Figure 38 - Bionic Commando

- Ninja Gaiden II – This game was released for the NES in 1989. The game combined elements of platforming, beat'em up, action and of course martial arts.
- Little Samson – This is a very good platformer that unfortunately is little known among NES game enthusiasts. It is an action platformer game with stellar graphics released in 1992.
- Castlevania III – A great story line that returns the franchise back to the original elements of the first game – action platform game. The game has multiple endings and

¹¹ They also have games for Sega Genesis, Game Gear, and many other classic systems

The best platformers were developed on the 16-bit consoles, Super Nintendo Entertainment System (SNES), Sega Genesis, etc. We will leave it as a exercise for you to explore and discover the best games/platformers from these consoles.

I highly encouraged that you try to play all the above NES games and actually try to complete as many as you can. It is well known that many NES games were rather difficult to play and unforgiving in comparison to today's games. Most of these games represent the best in the platforming genre.

Writing a Design Document

Before you start building and creating code for any game you should write a game design document (GDD). The document is used by everyone working on the game to obtain high-level concepts and details that describe the game to all those involved in creating art, programming and promoting the game.

Table 1 - Game Design Document Template

From: <http://wwwx.cs.unc.edu/courses/comp585-s11/585gamedesigndocumenttemplate.docx>

Game Design Document Outline

A game design document is the blueprint from which a game is to be built. As such, every single detail necessary to build the game should be addressed. The larger the team and the longer the design and development cycle, the more critical is the need. For your purpose, the intent is to capture as much as possible of your design. I want you to think big...bigger than what you are able to develop. I also want you to be clear about what the software delivers and what the design entails. My recommendation is that you define the ultimate game and then clarify what it is that you have developed. If you are finding it too difficult to do that, you may produce too documents.

1. Title Page
 - 1.1. Game Name – Perhaps also add a subtitle or high concept sentence.
2. Game Overview
 - 2.1. Game Concept
 - 2.2. Genre
 - 2.3. Target Audience
 - 2.4. Game Flow Summary – How does the player move through the game. Both through framing interface and the game itself.
 - 2.5. Look and Feel – What is the basic look and feel of the game? What is the visual style?
3. Gameplay and Mechanics
 - 3.1. Gameplay
 - 3.1.1. Game Progression
 - 3.1.2. Mission/challenge Structure
 - 3.1.3. Puzzle Structure
 - 3.1.4. Objectives – What are the objectives of the game?
 - 3.1.5. Play Flow – How does the game flow for the game player
 - 3.2. Mechanics – What are the rules to the game, both implicit and explicit. This is the model of the universe that the game works under. Think of it as a simulation of a world, how do all the pieces interact? This actually can be a very large section.
 - 3.2.1. Physics – How does the physical universe work?
 - 3.2.2. Movement in the game
 - 3.2.3. Objects – how to pick them up and move them
 - 3.2.4. Actions, including whatever switches and buttons are used, interacting with objects, and what means of communication are used

- 3.2.5. Combat – If there is combat or even conflict, how is this specifically modeled?
- 3.2.6. Economy – What is the economy of the game? How does it work?
- 3.2.7. Screen Flow -- A graphical description of how each screen is related to every other and a description of the purpose of each screen.
- 3.3. Game Options – What are the options and how do they affect game play and mechanics?
- 3.4. Replaying and Saving
- 3.5. Cheats and Easter Eggs
- 4. Story, Setting and Character
 - 4.1. Story and Narrative – Includes back story, plot elements, game progression, and cut scenes. Cut scenes descriptions include the actors, the setting, and the storyboard or script.
 - 4.2. Game World
 - 4.2.1. General look and feel of world
 - 4.2.2. Areas, including the general description and physical characteristics as well as how it relates to the rest of the world (what levels use it, how it connects to other areas)
 - 4.3. Characters. Each character should include the back story, personality, appearance, animations, abilities, relevance to the story and relationship to other characters
- 5. Levels
 - 5.1. Levels. Each level should include a synopsis, the required introductory material (and how it is provided), the objectives, and the details of what happens in the level. Depending on the game, this may include the physical description of the map, the critical path that the player needs to take, and what encounters are important or incidental.
 - 5.2. Training Level
- 6. Interface
 - 6.1. Visual System. If you have a HUD, what is on it? What menus are you displaying? What is the camera model?
 - 6.2. Control System – How does the game player control the game? What are the specific commands?
 - 6.3. Audio, music, sound effects
 - 6.4. Help System
- 7. Artificial Intelligence
 - 7.1. Opponent and Enemy AI – The active opponent that plays against the game player and therefore requires strategic decision making
 - 7.2. Non-combat and Friendly Characters
 - 7.3. Support AI -- Player and Collision Detection, Pathfinding
- 8. Technical
 - 8.1. Target Hardware
 - 8.2. Development hardware and software, including Game Engine
 - 8.3. Network requirements
- 9. Game Art – Key assets, how they are being developed. Intended style.
 - a.

We will try to come up with an abbreviated GDD for every game we create in order to get into the habit.

If you are serious about creating games than the first step should always be the GDD. “one of the stages of game design is communicating the design to others, ...Even if you’re developing a game all by yourself, it’s useful to write down the things you’ve decided on, to make notes and lists of features that you want to include in the game.

Check out the following links for actual game design documents:

<http://www.cs.tufts.edu/comp/150CIS/AnAntsLife/AnAntsLife-GameDesignDocument.pdf>

http://www.gamasutra.com/view/feature/130127/design_document_play_with_fire.php

Best Practices - revisited

A side goal we will have is to see how we can build our 2D game engine to have as many reusable and easily replaceable components. Another area we will try to exercise is the use of JUnit to test our algorithms.

I will not waste time trying to use JUnit on “Hello World” programs or Ant on Java projects intended to feature some programming feature (e.g. when exploring graphics methods for drawing). We will make greater use of best practice methods when we get into more complex game projects and start using more external libraries.

Using Best Practices

We will be using some coding best practices to create our games and game engines. A best practice is a rule that the software community has identified as leading to more readable and maintainable code. One such rule is to use JUnit to test our modules and classes. In order to make the building of the game test components easy to execute we will need a good build tool like Ant or Maven.

The key advantage to using best practices where testing is a guiding principle is that we get into the habit of creating our classes and modules so that they are independent and don’t have undue dependencies on each other. This allows us to swap out new classes to handle music, physics and graphics without having to make changes to the core classes.

The use of some of these techniques will seem like overkill for programs that are small and really just practice in demonstrating how to use some graphics routines. We will not do it for simple programs where the intention is to learn one or two graphic concepts but will use Maven and JUnit when we get into less than trivial programs.

What is JUnit?

JUnit is a simple to use framework for creating test cases and test suites. The goal of using a tool like JUnit is to create a suite (one or more testcases) that can be run as part of building your application. The key aspects of using a framework is to make the tests easily repeatable and integrated into how the application you are building gets built by the build tool. We will cover a “quick hands-on” introduction to JUnit and hopefully provide enough information and guidance so that you can appreciate its use and start to utilize it in your own programs if it happens to be a new tool for you.

What is Ant and Maven?

One thing you learn as the programs you build get larger, more complex and include more external libraries is the need for a build tool. There are several popular build tools for Java.

Ant is a Java library to assist in building Java applications. It provides the capability for developers to specify and control the files that are compiled, assembled and tested and executed.

Maven is a tool used for building and managing a Java-based project. In most cases we will not need to use Maven since our programs will not involve external libraries beyond what is available in the JDK. But once you start to use library X version A and library Y version B, etc, it is a good idea to use a build tool like Maven that will obtain those libraries from the Internet (or some locally accessible storage area) rather than have users fumble and try to figure out what and where you obtained your libraries from. I tend to favor Maven over Ant for my backend Java applications but it does impose a folder structure and most of the Java game libraries are not in Maven repositories. Therefore in keeping with the need for simplicity I will utilize Ant and confine myself to a minimal set of tasks so as stay focused on the purpose of learning 2D programming not how to use build tools.

What's the plan for learning how to use these tools?

We will present enough information and steps to make the instructions and use of these tools easy to follow. There are many websites available on the Internet that provide easy to follow tutorials if you would like more information.

Game Engines

The Plan

We will start by acquainting ourselves with the graphics capabilities available to us when programming in Java. Before we get into the action packed adventures in designing and creating our own 2D scrolling game we will examine how lines and rectangles are drawn and build rather simple one-screen games, such as Pong, Breakout and Minesweeper and progressively introduce more complex games with interesting mechanics. Together we will build many games but in order to actually test your understanding of the material you will be given challenging exercises to complete on your own. I highly encourage all readers truly vested in learning to do the exercises. We will then move on to learning how to use Slick2D (a java library that provides support for creating 2D games) and slowly work our way into a specific 2D platformer and finally build a generic 2D platform game engine. Along the way we will learn how to either create or borrow graphics and build tools (e.g map tile editor). The plan is to progress logically and thoroughly to master the programming aspects required to build a game.

Overview of the Plan

The chapters that follow will cover the following topics:

1. Creating Java projects that can be applications or Java applets
2. Basic 2D Graphics using Java APIs
3. Advanced 2D Graphics
4. The Game Loop and Threads
5. Keyboard Processing
6. Creating Pong
7. Collision Detection
8. Creating Breakout

9. Creating Mindsweeper
10. Building Combat Game
11. Using a joystick and other game devices
12. Learning Slick2D
13. Creating a TileMap
 - a. Building our own TileMap Editor
 - b. Using a common TileMap Editor
14. Building a 2D Vertical Scroller
15. Building a 2D Horizontal Scroller
16. Building a Generic 2D Platformer Game Engine
17. Creating Multiplayer Games



Before proceeding to the next chapters I highly recommend that you take the time to review the material in Appendix B, Appendix C and Appendix D that cover Eclipse, JUnit, and Ant, respectively. The material covered is by no means comprehensive or complete. The intention is to provide a familiarity with the tools and enough information to understand how and why they are being used within these notes. If you are familiar with the tools then by all means skip or better yet review and provide feedback.

Exercises

Exercise 1-1: Playing classic Games.

In order to get the feel for what inspired the development of the great 2D platformer games of yore we will set up our systems to allow us to play the games mentioned in this chapter.

STEP 1: Obtain an emulator (MAME or NES emulator).

STEP 2: Obtain game ROMs for the system.

STEP 3: Play some games, look up a description of the game on the Internet and original game reviews.

STEP 4: Play one of the games and develop a game design document for one popular 2D platformer created in the 1980's.

Exercise 1-2: Do research online to find and describe the top ten most popular platformer games for the SNES.

Exercise 1-3: Do research online to identify the top Sega Genesis platform games.

Exercise 1-4: Try to find and play one modern day platformer. Check out all the reviews. Play on your own and provide your own analysis of the game.

March 30, 2014

[BUILD A 2D PLATFORMER - JAVA]

Chapter 2 – Graphics

The objective of this chapter is to cover 2D graphics programming in Java. We will cover the following topics:

- Applications vs. Applets
- History of GUI and graphics using Java
- History of Java 2D
- The coordinate space
- Drawing Points, Lines and Shapes
- Color
- Painting and Rendering
- Text and Fonts
- Images and File Formats
- Image Processing
- Animations

In this chapter we will keep things simple – no games. In the chapters that follow we will utilize all the information in this chapter and build basic straightforward games.

Applications vs. Applets

A java program can be developed to run as either an application or an applet. The difference is that an applet runs within a container like a browser. You can actually develop your games to be invoked as either an application or an applet (to show off to friends on your website).

We will make the assumption that you have created many Java applications with minimal graphical elements.

Organizing the code

We will start to organize our code in a consistent manner that will fit the organization required for all our applications going forward.

We will use the following top-level package structure: **com.brainycode.platformer**.

We will place the code containing the main kick-off application under the additional packaging name **.main** (`com.brainycode.platformer.main`). If we create a Java Applet version of our code that will be under **.applet** package name, conforming to these conventions will allow us to immediately know where the start point is for all our applications and applets.

We will automatically create three additional folders *test*, *resources* and *libs*. The *test* folder will contain all our test cases and the *resources* directory will contain all our program images and sounds or any additional program elements (e.g data files). The *libs* folder will contain any external libraries our code depends on.

For the next few chapters we will create one workspace to hold the various Java projects we will be creating. You may be wondering – “Is there anyway to create my own Java project with Eclipse that contains all these folders?”

TBD: Discuss using the brainycode custom Java Project

The next example details the steps you can use in Eclipse to create a program that can run as either a Java Application or a Java Applet. Each program does the minimum of any graphics program – brings up a window and displays the text “Java 2D!”

Lab 2-1: Creating a program that displays a window and runs as either an application or applet

1. Create a new Eclipse workspace project (e.g. 2DPLATFORMER)
2. Create a new project SimpleGraphicsProgram
3. Create the new package com.brainycode.platformer.main
4. Create a new java class GamePanel that extends JPanel

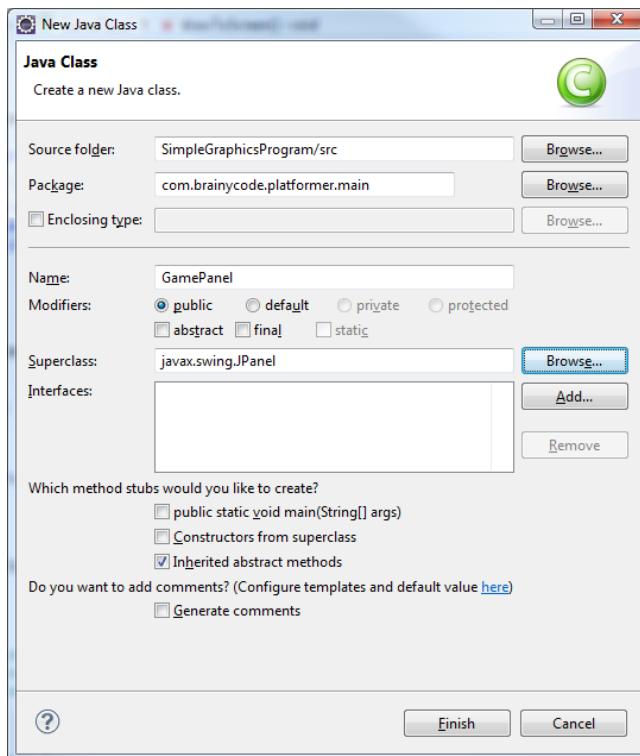


Figure 40 - New class GamePanel

The GamePanel name will probably have a warning indicator about the need to add “serial version id”

```
public class GamePanel extends JPanel {  
}
```

Figure 41 - Eclipse flagging the class name

There may be occasions where you will want to save the state of a class (e.g. when a user decides to save a game state) this will require that you serialize the class, in other words save the class to the hard drive. Later you may need to restore the information or unserialize the class. There are many problems unserializing when a later version of the class has added fields or changed. To mitigate such problems you can associate a serial id with the class. This will be used instead of reflection to unserialize the class.

5. Right-click on the class name, select Quick Fix and select “Add default serial version id.” (Going forward we will assume that you will do this for each new class).

A JPanel provides a general-purpose container to hold other components or widgets.

There are two graphics toolkits that come with Java – AWT and Swing. The Abstract Window Toolkit (AWT) was the original graphics toolkit that was delivered with the very first version of Java in 1995.

AWT provided the “windowing, graphics and user-interface widget toolkit.” So it was the library to use to create a GUI application that is, applications with windows, button, menus and textboxes. In order to come up with something very quickly that could provide graphic functionality across many machines and platforms AWT was purposely designed as a thin layer between Java and the actual platform¹²’s graphical APIs. That is the graphic component (windows, buttons, menus etc) are rendered by the platform operating system graphics library. There were two major problems with this approach. The first is the lowest common denominator of graphics and window functionality was provided and second all the applications took on the look and feel of their native platform so you could not get applications to look (and even behave) the same across platforms since it was their underlying operating systems APIs that was being used to draw and manage the screen.

In Java version 1.2, Sun Microsystems introduced the Swing toolkit. Swing was first developed by the then Netscape Communications Corporation in 1996. The goal was to develop sharper and more elegant looking GUI components than what was provided by AWT. Swing was developed so that applications would appear the same across different platforms. In addition, the look and feel was intended to be pluggable. The library provided a richer set of widgets that were implemented strictly in Java.

The current version of Java handles more easily the mixing of components from both toolkits (that was once a problem). The two libraries are not independent from each other as the class hierarchy diagram below illustrates:

¹² The first version of Java ran on the following platforms: Windows 95 and NT, Sun Solaris and later Mac OS 7.5.

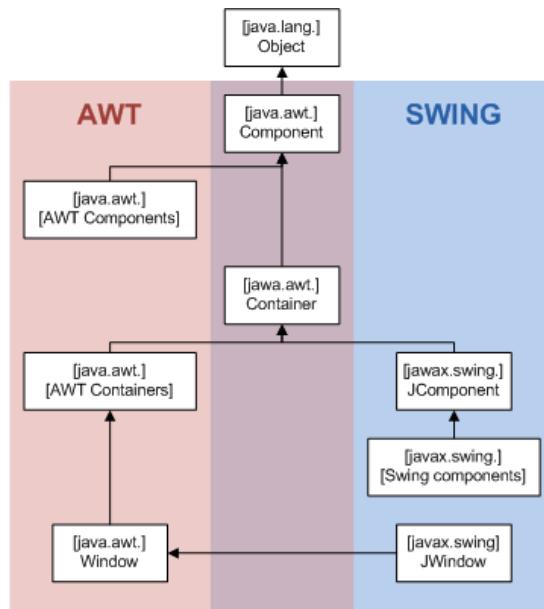


Figure 42 - AWT Swing class hierarchy (from <http://en.wikipedia.org/wiki/File:AWTSwingClassHierarchy.png>)

Both AWT and Swing are now part of the Java Foundations Classes (JFC). The fundamental GUI object shared by both AWT and Swing is the `java.awt.Component` class. A component is an object having a graphical representation that can be displayed on the screen. So all the entities we see on a Java GUI screen derives from the component class. There are two types from components – *lightweight* and *heavyweight*. A lightweight component is not associated with a native window (this is true for all Swing components) and a heavyweight component is associated with a native window (this is true for all AWT components). A container is just an object that can contain other components.

It is easy to tell the difference between classes that are associated with AWT from those that are associated with Swing – all Swing classes start with the letter J (e.g. `JComponent`, `JPanel`, `JFrame`, etc).

A `JPanel` is a key class to developing a professional looking GUI. Our `GamePanel` (extends `JPanel`) and allows us to exert complete control over what is drawn on the screen.

6. Add the following (minimal code) to the `GamePanel` class:

Table 2 - `GamePanel.java`

```

package com.brainycode.platformer.main;

import java.awt.Dimension;
import java.awt.Graphics;

import javax.swing.JPanel;

public class GamePanel extends JPanel {

    private static final long serialVersionUID = 1L;

    // dimensions
  
```

```
public static final int WIDTH = 640;
public static final int HEIGHT = 480;

public GamePanel() {
    super();
    setPreferredSize ( new Dimension(WIDTH, HEIGHT));
    setFocusable(true);
    requestFocus();
}

public void paintComponent(Graphics g) {
    super.paintComponent(g);
    g.drawString("Java 2D!", 300, 220);
}

}
```

We plan on using the GamePanel above to display on the screen the words “Java 2D!” The thing we need now is a window to place the GamePanel in.

Every GUI uses a window to display or layout the components of the GUI (e.g. input boxes, buttons). There are three types of windows:

- Applet – This window is created and managed by the container (e.g. web browser) that is displaying/running the Java Applet
- Dialog – This a window used to convey or obtain information from the user
- Frame – This is a top-level container that is not contained within other containers.

The Frame in our Java application will be used to create and manage our platform’s window. The Java class that captures the notion of Frame is the JFrame class.

Quick Question 1: Is JFrame from the AWT or Swing library?

We can think of a JFrame as being partitioned into “panes.” Each pane is positioned on top of one another as shown:

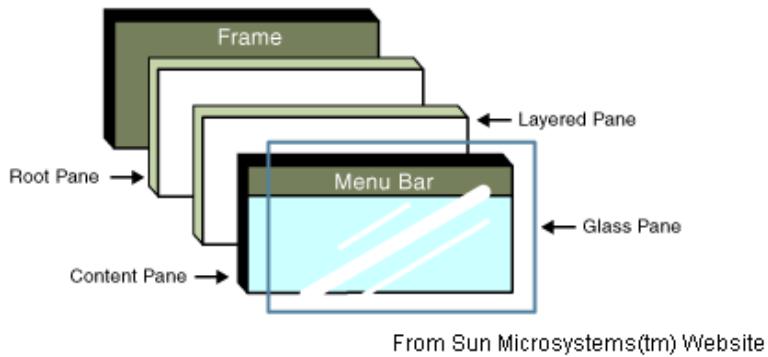


Figure 43 - The "panes" of a `JFrame`¹³

The “panes” are:

- Root Pane – This pane manages the content pane and the menu bar
- Layered Pane – This pane contains the menu bar and content pane and enables Z-ordering of other components. This container has depth in order to manage overlapping components (e.g. ensures popups appear on top of all other components.)

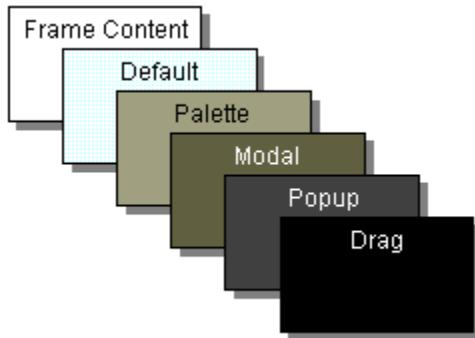


Figure 44 - Depth of a layered pane

- Content Pane – This pane holds all the root pane’s visible components except the menu bar. This is the pane where we place our `GamePanel` or any visible component (e.g. buttons, icons, etc)
- Glass Pane – This pane is used to intercept input events occurring over the top-level container. This pane is hidden by default.

¹³ The images are actually from <http://www.macs.hw.ac.uk/guidebook/?name=Introduction&page=3>

What is Z-ordering?

The Z-order is the order in which the objects in a window are drawn. If you had a window which was displaying three buttons in a panel you would want the buttons to be drawn after the panel not before!

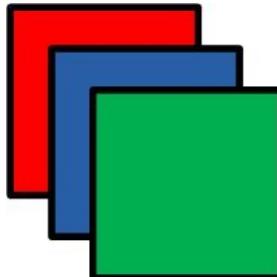


Figure 45 - Images with one z-order

The same squares in the same position on the screen but with opposite z-order will appear as:

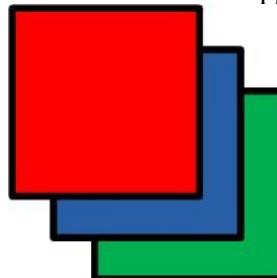


Figure 46 - Same set of squares drawn using reverse z-order

As you can see the window display can look totally different depending on the z-order of the components.

Each pane serves a different function to manage the GUI.

7. Create the Java class file Game.java in the package com.brainycode.platformer.main
8. Enter the following lines of code into the Game class:

Table 3 - Game.java

```
package com.brainycode.platformer.main;

import java.awt.Dimension;
import java.awt.Toolkit;

import javax.swing.JFrame;

public class Game {

    public static void main(String[] args) {
        JFrame window = new JFrame("Simple Graphics Program");
        window.setContentPane(new GamePanel());
        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        window.setResizable(false);
        window.pack();
    }
}
```

```
        window.setLocationRelativeTo(null);
        window.setVisible(true);

    }

}
```

The first thing we do is create our application window supplying the window title as an argument:

```
JFrame window = new JFrame("Simple Graphics Program");
```

We then establish our GamePanel as the windows or JFrame's content:

```
window.setContentPane(new GamePanel());
```

Note, that we create and set the GamePanel as the content pane.

We want the user to be able to click on the close window icon and have the application window close.

```
window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

We also want to disable the ability of the user to resize the window screen:

```
window.setResizable(false);
```

The pack method sizes the frame so that all its contents are at or above their preferred sizes. An alternative to pack is to establish a frame size explicitly by calling setSize or setBounds (which also sets the frame location). In general, using pack is preferable to calling setSize, since pack leaves the frame layout manager in charge of the frame size, and layout managers are good at adjusting to platform dependencies and other factors that affect component size.

```
window.pack();
```

We also want the window to be centered on the user's screen. JFrame provides a simple method to get that done:

```
window.setLocationRelativeTo(null);
```

The last thing we want to do is "see" or make the window visible.

```
window.setVisible(true);
```

9. Run the application.

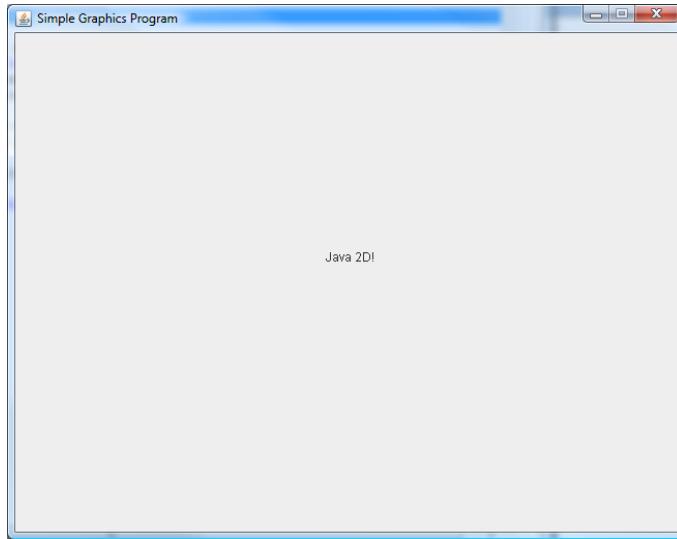


Figure 47- Our "Simple Graphics Program"

How did the text "Java 2D!" actually get displayed?

You may be wondering how the text was displayed since the only method that was invoked from the GamePanel class was the constructor:

```
public GamePanel() {
    super();
    setPreferredSize ( new Dimension(WIDTH, HEIGHT));
    setFocusable(true);
    requestFocus();
}
```

The constructor invokes the JPanel constructor (super()), sets the preferred size of the GamePanel component (width: 640, height: 480) and then directs that this component should have the focus.

Looking at the GamePanel class we see that it has another method:

```
public void paintComponent(Graphics g) {
    super.paintComponent(g);
    g.drawString("Java 2D!", 300, 220);
}
```

And using our powers of deduction, we know that somehow the paintComponent method must have been invoked in order to draw our string "Java 2D!". When you create a GUI application the painting of the screen can be triggered by the operating system (first time showing the window, restoring after minimization, moving the window around on the screen, etc) or your application can request a window repaint (after updating the enemies you want to show their new positions on the screen). Your code does not invoke the paintComponent () method directly. There is an *Event Dispatching* thread that is managed by the GUI framework that handles when the paint () or paintComponent () method of your visible components are invoked. The best you can do is request that a paint request be queued up (more on this later).

The *Event Dispatching Thread* sends to your `paintComponent()` method a `Graphics` object that is “pre-configured with appropriate state for drawing.” Your code can use the `Graphics` object to change the color, font, clipping area or translation of the component.

`paint()` vs `paintComponent()`

You may encounter different game or graphics programming books that override the `paint()` method instead of the `paintComponent()` method. The fact is that the Event Dispatcher invokes the `paint()` method which in turn will invoke the `paintComponent()` method, `paintBorder()` and `paintChildren()` methods. When the Event Dispatcher invokes `paint()` method of the top-level container (`JFrame`) it in turns invokes the `paint` methods of all components it contains. It is safer to always use `paintComponent()` since it will ensure you do not paint over child components.

EXERCISE 2-1: Update the program you just completed by adding a `System.out.println` at the end of `main` that prints out the window objects width and height.

Hint: Use the Eclipse auto-completion feature to see if the window object provides these values.

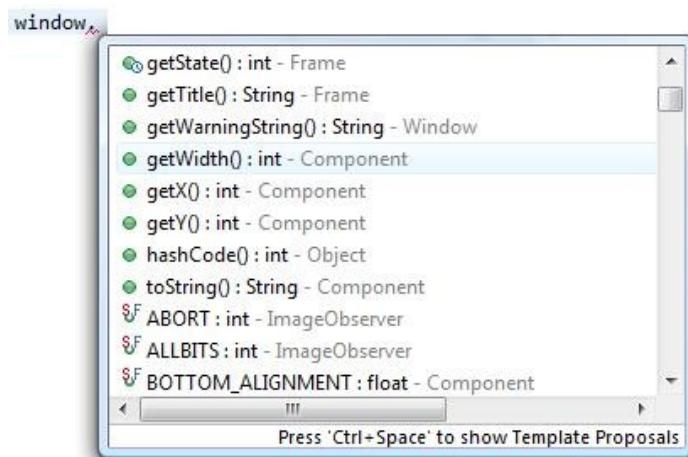


Figure 48 - Using Eclipse autocompletion

If you added the line you may see something like the following in the console window:

```
window width:646    window height: 506
```

Why is the window larger than the dimensions we set for the GamePanel?

It is interesting to see that the window is 646 x 506 but the GamePanel we added (the only component we added to the window!) is actually 640 x 480? If you examine the window in Figure 47 you will notice that the window has borders on the sides and the menu area on the top. This accounts for the extra pixels that are required to display the frame or window.

10. Add the new package `com.brainycode.platformer.applet`
11. Create a new class `GameApplet` under this package that extends the `JApplet` class.
12. Add the following code:

Table 4 - GameApplet.java

```

package com.brainycode.platformer.applet;

import java.awt.BorderLayout;
import java.awt.Container;

import javax.swing.JApplet;

import com.brainycode.platformer.main.GamePanel;

public class GameApplet extends JApplet {

    private static final long serialVersionUID = 1L;
    private GamePanel gamePanel;

    public void init() {
        Container pane = getContentPane();
        pane.setLayout(new BorderLayout());
        gamePanel = new GamePanel();
        pane.add(gamePanel);
        this.setSize(GamePanel.WIDTH,GamePanel.HEIGHT);
    }
}

```

13. Run the GameApplet class. It will use the AppletViewer as the Applet container

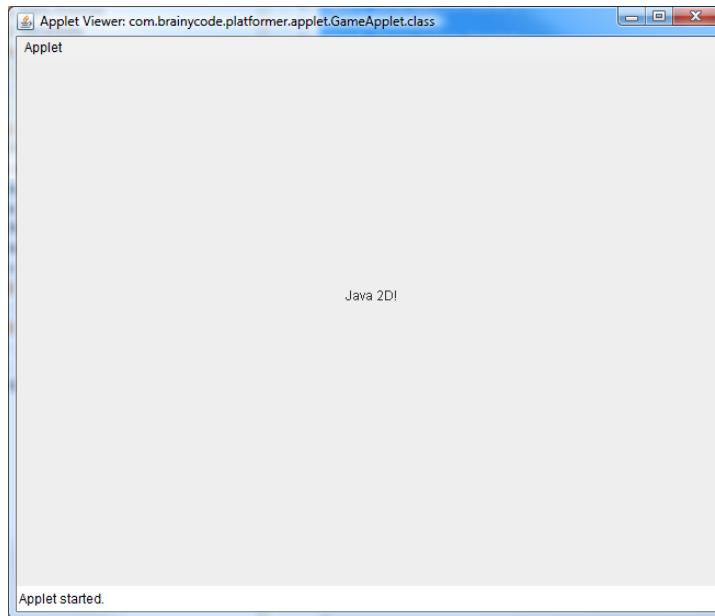


Figure 49 - Running your program as an Applet

It is pretty easy to create your programs to run either as an applet or application.

The applet code is pretty straightforward. The applet includes an init() method that is invoked by the applet container at startup. The applet container provides the content pane so the first thing to do is to obtain it in order to use it to add components (our GamePanel) to it:

```
Container pane = getContentPane();
```

Now that we have the content pane reference in the object pane we will establish the layout manager that will be used to organize the components in the applet window:

```
pane.setLayout(new BorderLayout());
```

A border layout lays out a container so that the components fit into one of five regions as shown below:

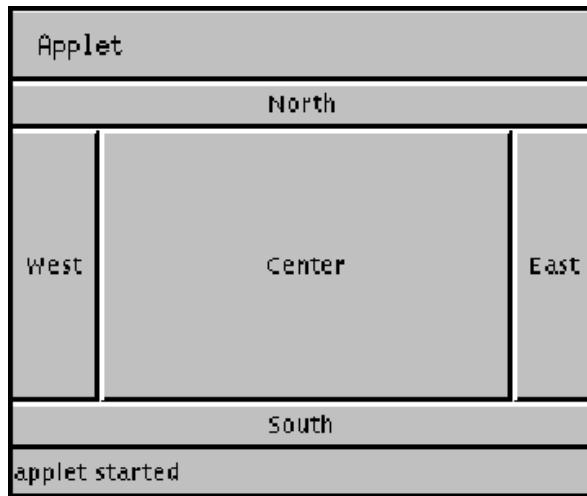


Figure 50 - Border layout

The next lines create and add the GamePanel to the content pane. By default since we did not specify what region to use it will be BorderLayout.CENTER. In addition, since we only have one component it will take up the entire screen real estate.

```
gamePanel = new GamePanel();
pane.add(gamePanel);
```

The last line set the width and height of the applet window to match the width and height of the GamePanel:

```
this.setSize(GamePanel.WIDTH,GamePanel.HEIGHT);
```

Let's continue this lab by seeing how we can convert our applications as executable jars that we can directly run on the desktop or as applets that we can run on the Internet.

14. Dismiss or close the appletview window.
15. Highlight the Project name “SimpleGraphicsProgram”
16. Select File → Export
17. Select Java → Runnable JAR file

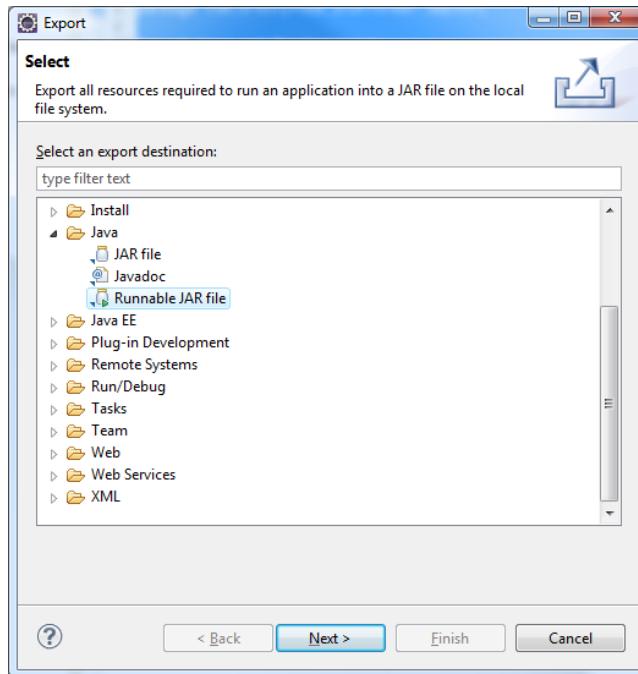


Figure 51 - Exporting Runnable JAR file

18. Click on Next >

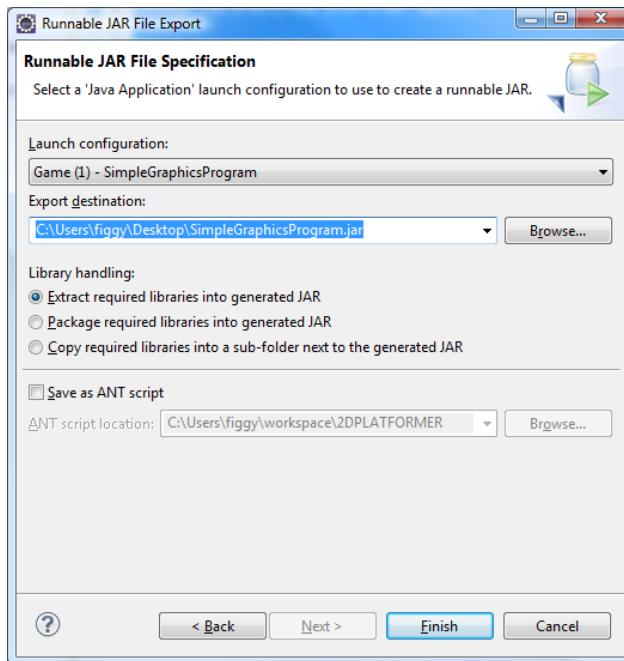


Figure 52 - Preparing executable jar for desktop

19. Click on Finish. You may see a dialog box that there was errors but just close the dialog to continue and complete the creation of the jar.

20. Find the SimpleGraphicsProgram icon on your desktop. Double-click to start. You will see displayed the same window as when you executed the application within Eclipse.

This will be the steps you will follow when you want to create an executable jar of any of the applications you create.

21. Create an HTML file by right-clicking in the com.brainycode.platformer.applet package and selecting New → Web → HTML File
22. Use the file name SimpleGraphicsProgram.html and click on Finish.
23. Enter the following code

Table 5 - SimpleGraphicsProgram.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Simple Graphics Program</title>
</head>
<body>
    <applet code="com.brainycode.platformer.applet.GameApplet.class"
archive="GameApplet.jar"
width="640" height="480"></applet>
</body>
</html>
```

24. Create a jar file for the applet by highlighting the SimpleGraphicsProgram and selecting File → Export.
25. Select Java → JAR file

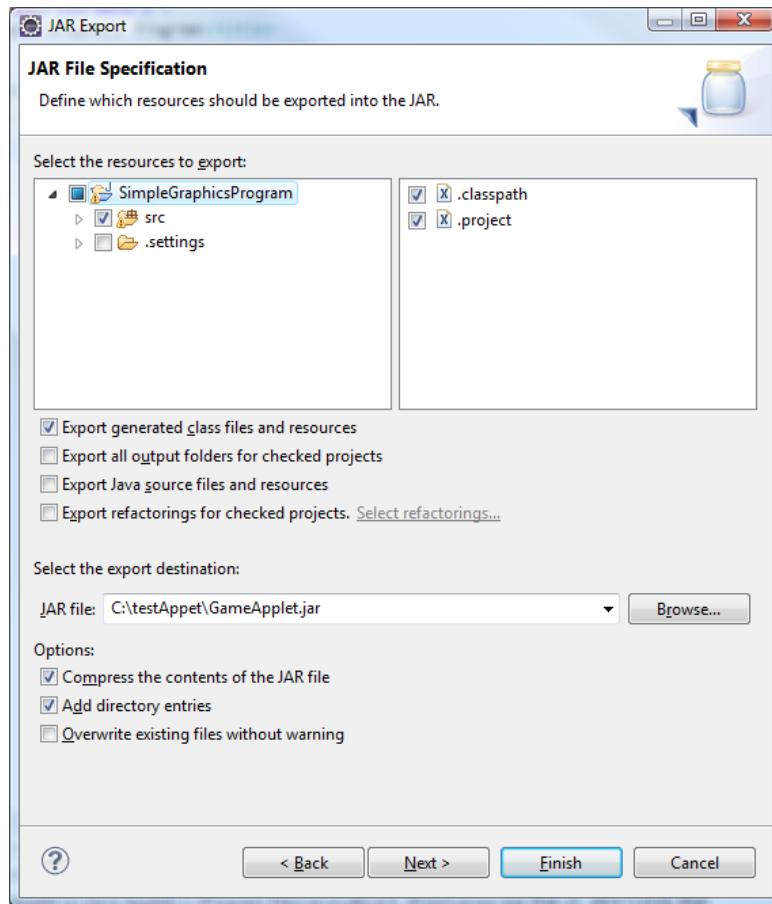


Figure 53 - Creating jar file for applet

Note, I am placing my applet jar file in a test directory C:\testApplet and naming the jar file GameApplet.jar.

26. Copy the html file into the same directory that you placed the jar file
27. Open the html file with your favorite browser by either double-clicking on the html file or entering: file:///C:/testApplet/SimpleGraphicsProgram.html

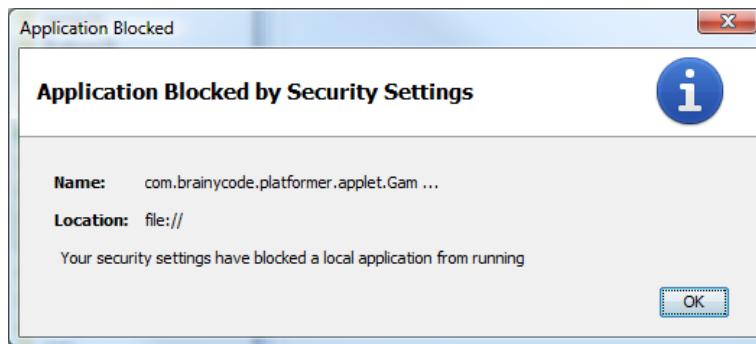


Figure 54 - Application Blocked by Security Settings dialog box

If you see the Application Blocked dialog box you will need to open the Control Panel and open the Java Control Panel. Select the Security tab and lower it to Medium.

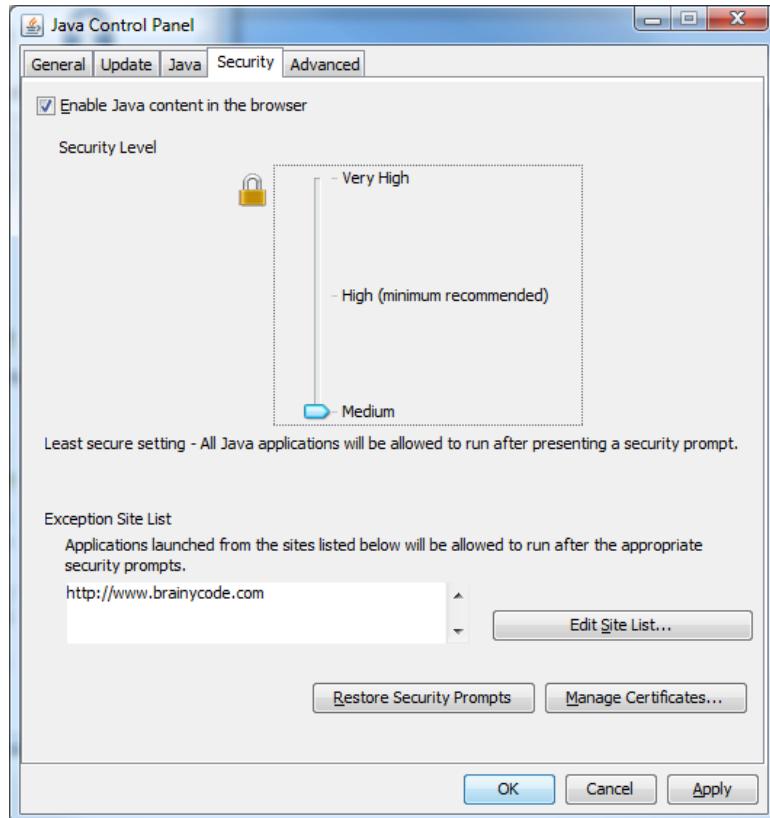


Figure 55 - Changing the Security in the Java Control Panel

Click on “OK” and try Step 27 again.

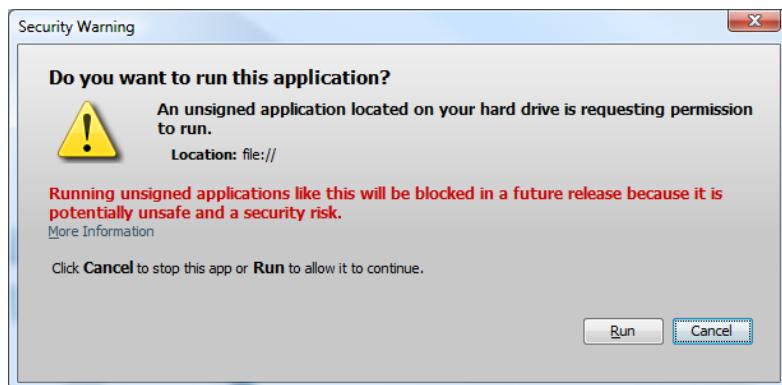


Figure 56 - Security Warning Prompt

Select “Run”

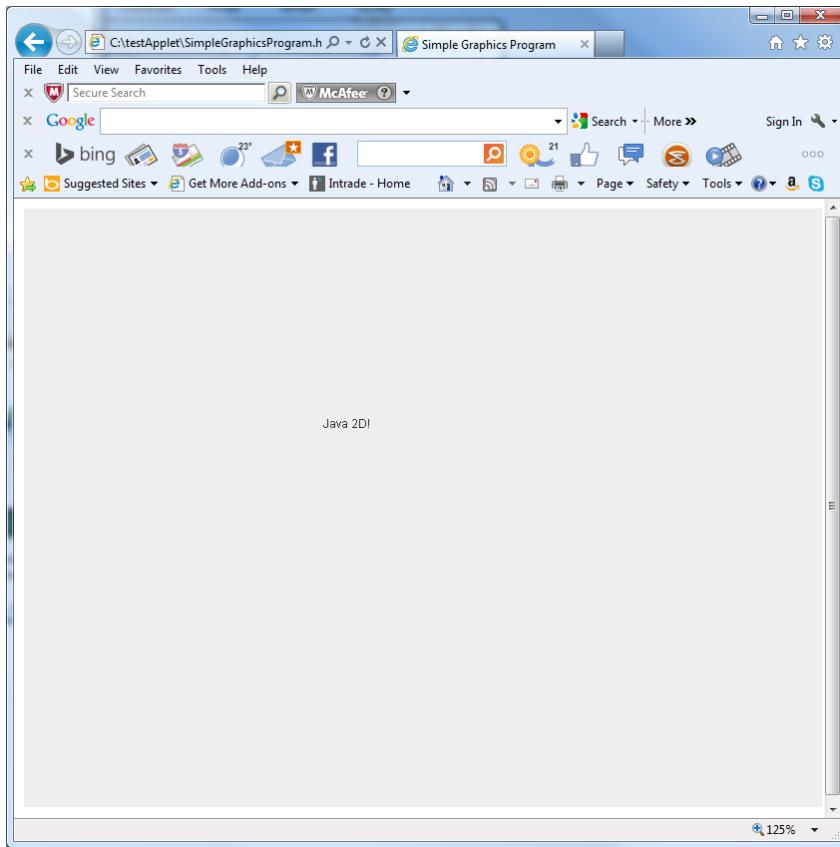


Figure 57 - The Java Applet running in IE

Drawing

Color

We can change certain aspects of the objects we draw – one example is the color. In our first lab we did not specify the color of the text so the current or default color associated with the graphics context was used for the background and the text. The method to make use of to change or set the draw color is

```
setColor(Color c);
```

The method format is:

```
graphicsObj.setColor(Color c)
```

Sets the graphicsObj current color to the one specified as an argument.

The Color class is used to specify a color using the sRGB¹⁴ color space. We can either use one of the static constants defined in the Color class (e.g. Color.BLACK) or specify the exact value by providing values for r(ed), g(reen) and b(lue). The values for r, g, and b range from 0-255.

```
// Two ways to set the current color to red
g.setColor(Color.RED);
g.setColor(new Color(255,0,0));
```

You can do a search on the Internet¹⁵ for an RGB Color chart and find out for example that a cool looking yellow can be specified using: `new Color(231,199,31)`.

Color Examples		
Color	Color HEX	Color RGB
Black	#000000	rgb(0,0,0)
Red	#FF0000	rgb(255,0,0)
Green	#00FF00	rgb(0,255,0)
Blue	#0000FF	rgb(0,0,255)
Yellow	#FFFF00	rgb(255,255,0)
Cyan	#00FFFF	rgb(0,255,255)
Magenta	#FF00FF	rgb(255,0,255)
Grey	#C0C0C0	rgb(192,192,192)
	#FFFFFF	rgb(255,255,255)

Figure 58 - Color Examples from http://www.w3schools.com/html/html_colors.asp

drawstring()

In the previous lab we displayed a text message to the screen “Java 2D!” The method we used was:

```
g.drawString("Java 2D!", 300, 220);
```

The method format is:

```
graphicsObj.drawString(String str, int x, int y)
```

Draws the text given by the specified string, using the graphics context's current font and color

The x and y values correspond to the bottom-left position of the rectangle that would enclose the string text.

¹⁴ See <http://www.w3.org/Graphics/Color/sRGB.html> for more information on Standard Default Color Space

¹⁵ Hereafter I will just use the term “google” for searching the Internet – but note I love www.bing.com!



Figure 59 - The meaning of the x and y coordinates

Centering the Text

My intention when selecting the coordinates (300, 220) was to “guess” on the location that would place the text at the center of the window. The correct way to try to center text in any area where you know the width and the height is not to guess the location but to figure it out more accurately using information about the window size and the text size.

The image below depicts the location we want to determine in order to center our text in any rectangular area we place it in.

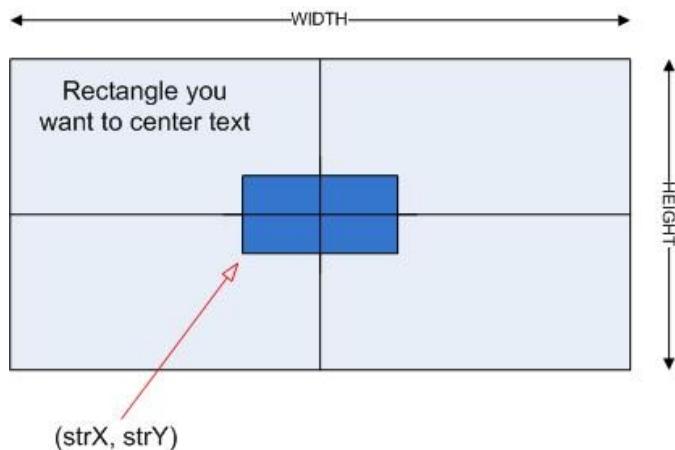


Figure 60 - Centering text

The larger grey area represents the larger area (in our case our GamePanel) that we want to center the text. The dark blue area represents the rectangular area encompassing the text we plan on drawing to the screen. We know the WIDTH and HEIGHT of the GamePanel. There exists a class – FontMetrics that we can use to obtain the rectangular bounds of our text. The FontMetrics class “encapsulates information about the rendering of a particular font on a particular screen.” The method in FontMetrics that we want to use is the getStringBounds method:

The method format is:

```
public Rectangular2D getStringBounds(String str, Graphics g) or  
fontMetricsObj.getStringBounds(String str, Graphics g)
```

Returns the bounds of the specified String in the specified Graphics object g.

A Rectangular2D object describes a rectangle defined by the top-left location (x,y) and the width and height (see Figure 67).

From the figure above you can see that we can calculate the location (strX, strY) by first finding the center of the GamePanel and by subtracting half the string's rectangle width and adding half the string rectangle height.

Lab 2-1: continued

Modify the GamePanel paintComponent() method by adding the following lines:

Table 6 - Centering the text

```
// Set the current color to red
g.setColor(Color.RED);

// Obtain the rectangular bounds of the text
FontMetrics metrics = g.getFontMetrics();
Rectangle2D strRect = metrics.getStringBounds("Java 2D!", g);

// Calculate the center of the screen
int centerPanelX = WIDTH / 2;
int centerPanelY = HEIGHT / 2;

// Calculate the starting point for the centered string
int strX = centerPanelX - (int)(strRect.getWidth() / 2);
int strY = centerPanelY + (int)(strRect.getHeight() / 2);

// Now draw centered (in red)
g.drawString("Java 2D!", strX, strY);
```

In Figure 61 the text in black shows my attempt to guess the (x,y) coordinates for the text to be centered on the screen. The text in red is the calculated value.

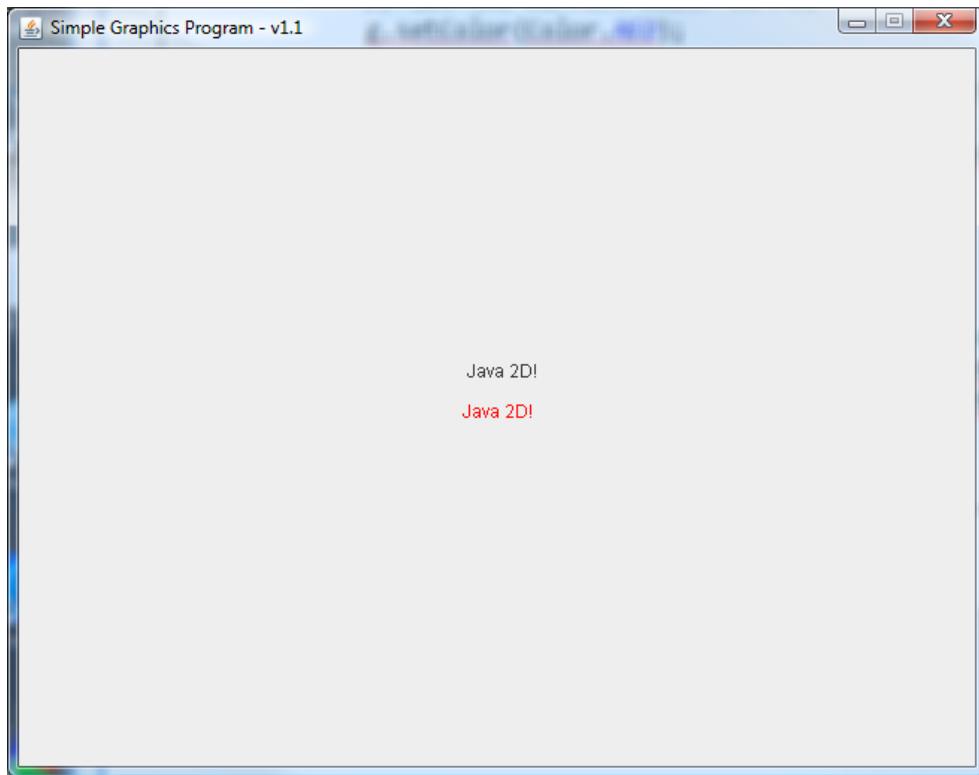


Figure 61 - Guess vs. Calculated center for text

My “guess” was not bad.

Drawing Lines

Another cool method to draw with is the `drawLine` method to draw – yes – lines on the screen.

The method format is:

```
graphicsObj.drawLine(int x1, int y1, int x2, int y2)
```

Draws a line, using the current color, between the points (x_1, y_1) and (x_2, y_2)

The first two arguments is the starting position (x_1, y_1) and the last two the end position (x_2, y_2) as shown below:

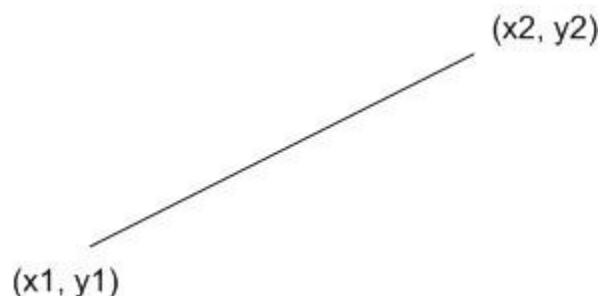


Figure 62 - Using `drawLine(x1,y1, x2, y2)`

Quick Exercise 1: Let's add two more lines to our last lab by drawing two lines to mark the center of the GamePanel.

Hint: To draw a horizontal line the y values stays at HEIGHT/2, and to draw the vertical line the x values stay at WIDTH /2.

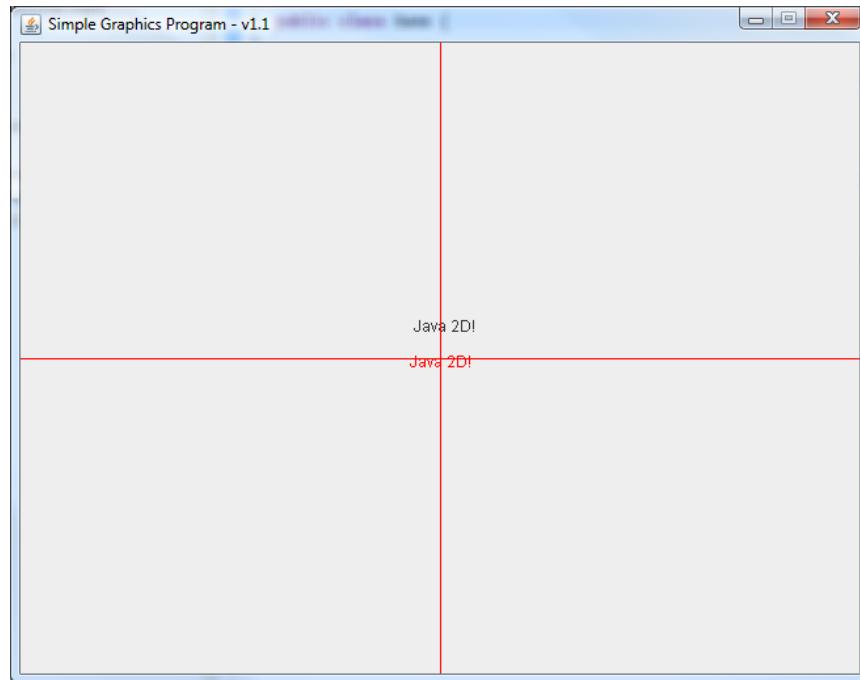


Figure 63 - Drawing center lines

An interesting fact is that there is no method to draw a point. There is no `drawPoint(x,y)` method in the `Graphics` class. One way to draw a point is to use `drawLine` and use the same `(x,y)` for the start and the end position.

Example: Drawing a red point at location (100, 100);

```
g.drawLine(100,100,100,100);
```

Fonts

We can also change the font by specifying the following:

- Font name – the default font used is “Dialog” we can specify any font available in the platform (or our own custom font). The `Font` class makes some font names readily available using static constants.
- Style – we can print the text plain, bold, italic, or bold and italic (respectively, `Font.PLAIN`, `Font.BOLD`, `Font.ITALIC`, or `Font.BOLD + Font.ITALIC`). Again, you use static constants within the `Font` class.

- Size – we can specify the font size. The typical font size for text is 10-pt or 12-pt.

We use the Font class to create a Font object and providing the parameters of name, style and size.

The constructor format is:

```
public Font (String name, int style, int size);
```

This method creates a new Font using the specified font family name, style and point size.

```
Font myFont = new Font(Font.MONOSPACED, Font.BOLD, 24);
```

In order to use the font when drawing text you have to set the current graphics context to myFont.

The method format is:

```
graphicsObj.setFont(Font f);
```

Sets the graphics's context font to the specified font.

Example:

```
g.setFont(myFont);
```

Add the two lines above after the super.paintComponent(g) line. This will have both our “guess” and calculated text print out using myFont.

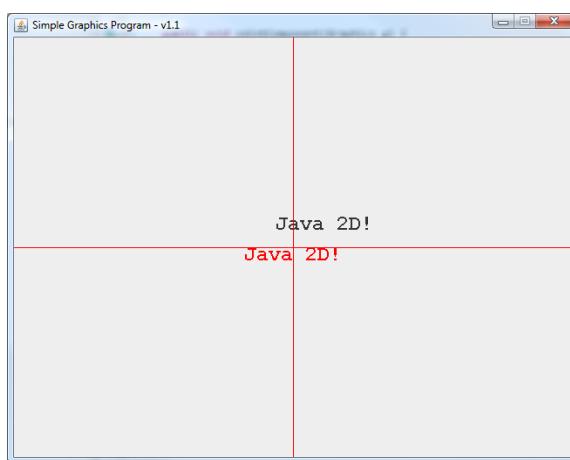


Figure 64 - Printing the text with a differnt Font

As you can see in the figure above the “guess” position text is more off center. The calculated one appears to require some adjustment (in the y direction) as well but it is good enough for our purposes.

Lab 2-2: Drawing a Star Field

1. Create a new Java Project call it StarField.
2. Make sure it has all the typical file layout under src we will use going forward:
 - a. com.brainycode.platformer.main

- i. Game.java – Change the title to “Star Field”
 - ii. GamePanel.java
3. . In the paintComponent() method of the GamePanel add the following code:

Table 7 - Creating a star field

```

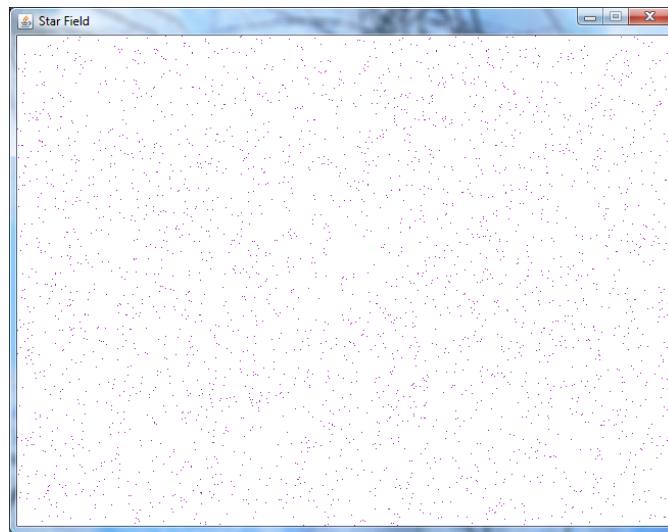
super.paintComponent(g);
// first make background
g.setColor(Color.WHITE);
g.fillRect(0, 0, WIDTH, HEIGHT);

// draw random stars
Random r = new Random();
Color randomColor = new Color(r.nextInt(256), r.nextInt(256),
r.nextInt(256));
g.setColor(randomColor);
for (int i=0; i < 3000; i++) {
    int x = Math.abs(r.nextInt() % WIDTH);
    int y = Math.abs(r.nextInt() % HEIGHT);
    System.out.println("x: " + x + "\ty:" + y);
    g.drawLine(x, y, x, y);
}

```

The result is shown below. The code uses the random number generator to generate a random position on the screen and draws a thousand white points on a black screen background.

The above is not very efficient but serves the purpose of demonstrating how to draw a point on the screen.

**Figure 65 - The star field**

Note: Print considerations mandated that I change the background from Color.BLACK to Color.WHITE and rather than printing out Color.WHITE for the stars I just generated a random color.

Random Number Generation

The use of a random number generator to add some variability to our games or programs is a common practice.

In Java we use the Random class to generate a stream of pseudorandom numbers. The reason for the designation of “pseudo” which commonly means “not real” is of course because the next value returned is determined by the algorithm used to generate the numbers. So the “next number” is not all that random. But, we can start anywhere in the sequence and the set of numbers are large enough to appear for all intents and purposes – random.

There are two constructors:

```
Random();
```

```
Random(long seed);
```

If you do not provide a seed value the random number generator will generate the same sequence (which may be good if you are testing or in my example since the stars appear in the same position on the screen each time the `paintComponent()` is called). To add some variability to the program you may want to provide a seed value based on the current time:

Example:

```
Random r = new Random(System.currentTimeMillis());
```

The above works to place the random number generator somewhere in the sequence that is determined by the long value returned by `System.currentTimeMillis()` – or the current time in milliseconds.

To get the next random number you can use the method `nextInt()`.

The method format is:

randomObj.nextInt()

Returns the next pseudorandom, uniformly distributed int value from the random number sequence.

Since we really want an integer value from `0..WIDTH-1` and `0..HEIGHT-1` so that the star point is drawn on the screen we scale the returned number by using the modulo % function.

Example:

```
r.nextInt() % WIDTH
```

In addition the random number can be any positive or negative number so we use `Math.abs()` to make our points positive.

We could have used another Random method to automatically do the scaling for us:

The method format is:

randomObj.nextInt(int n)

Returns a pseudorandom, uniformly distributed int value between 0 (inclusive) and the specified value (exclusive).

Example:

```
int x = r.nextInt(WIDTH);
```

Drawing Ellipses and Circles

Ellipses and circles can be drawn using the method drawOval().

The method format is:

```
graphicsObj.drawOval(int x, int y, int width, int height)
```

This method draws an ellipse/oval on the screen where x and y specify the top-left position of the rectangle enclosing the oval and width and height.

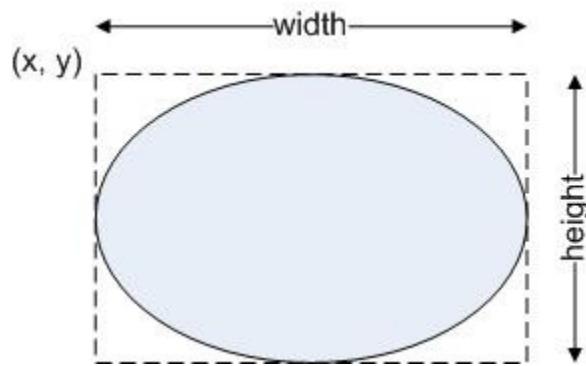


Figure 66 - Drawing an Oval

To draw a circle is simply a matter of making the width == height.

A method to use to draw an oval or circle filled with a particular color is the fillOval() method.

The method format is:

```
graphicsObj.fillOval(int x, int y, int width, int height)
```

The above fills in an ellipse/oval using the current graphics color. The parameters arguments are similar to the drawOval().

Drawing Rectangles

The method to draw rectangles and squares is drawRect()

The method format is:

```
graphicsObj.drawRect(int x, int y, int width, int height)
```

This method draws the outline of the triangle where the top-left position is at (x, y) on the screen. The arguments width and height define the rectangle width and height.

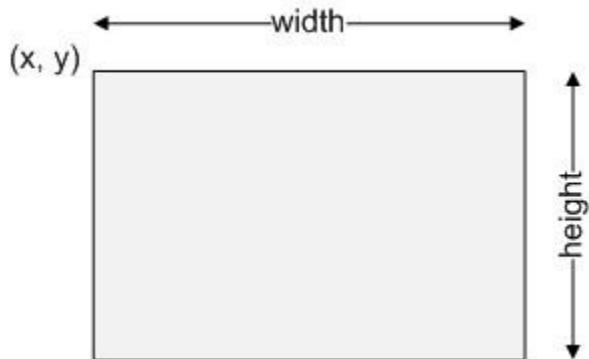


Figure 67 - Drawing a rectangle

If you want a rectangle of a particular color use the method `fillRect` method:

```
graphicsObj.fillRect(int x, int y, int width, int height)
```

Fills the specified rect. The rectangle fills up the pixels starting at (x, y) position on the screen up to $x+width-1$ and $y+height-1$.

Another useful draw method related to rectangles is the `clearRect()` method.

The method format is:

```
graphicsObj.clearRect(int x, int y, int width, int height)
```

This method clears the rectangle specified by filling it with the background color.

Additional draw methods

drawArc

The `drawArc()` method provides a way for us to draw part of an oval or circle.

The method format is:

```
graphicsObj.drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)
```

This method draws the outline of an oval arc defined by the rectangle using the `startAngle` as the start of the drawing and the `arcAngle` as the end.

Suppose we specified:

```
g.drawArc(300,200, 50, 50, 0, 180);
```

The above draws the arc within a rectangle that has its top-left position at (300, 200), has a width and height of 50 (a square) the arc starts at 0° and ends at 180° .

The image below depicts the various locations of angles around a circle:

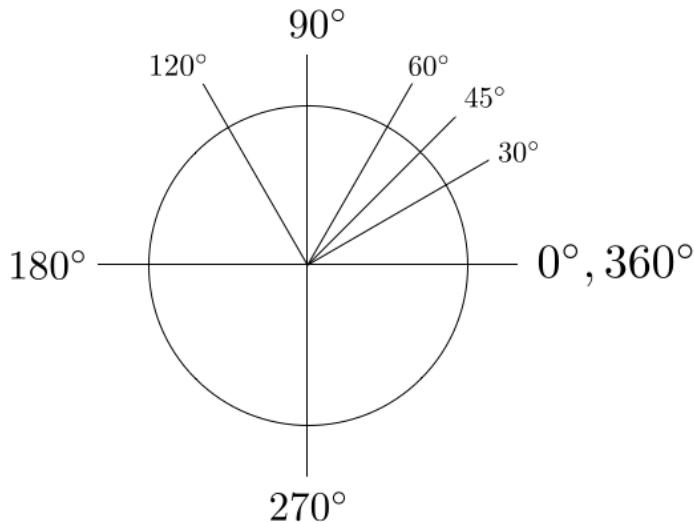


Figure 68 - Angles around a circle

The statement:

```
g.drawArc(300,200, 50, 50, 0, 180);
```

starts at 0 degrees and draws to 180 degrees.



Figure 69 - Drawing an arc

Exercise: How do you think we draw just the bottom half of the circle as shown below?



Figure 70 - Drawing the bottom half

Let's try a rectangle:

```
g.drawArc(300,200, 50, 20, 0, 180);
```



Figure 71 - The arc of a "flatter" rectangle from 0 ..180 degrees

Another method we can use is the fillArc method in order to obtain interesting effects:

The method format is:

```
graphicsObj.fillArc(int x, int y, int width, int height, int
startAngle, int arcAngle)
```

Fills a circular or oval using the enclosing rectangle defined by the starting top-left position (x,y) and width and height.

Example:

```
g.fillArc(300,200, 50, 20, 0, 180);
```



Figure 72 - fillArc example

Lab 2-3: Draw a Happy Face

1. Create a new Java Project call it HappyFace.
2. Make sure it has all the typical file layout under src we will use going forward:
 - a. com.brainycode.platformer.main
 - i. Game.java – Change the title to “Happy Face”
 - ii. GamePanel.java
3. In the paintComponent() method of the GamePanel add the following code:

```
// Draw the face
g.setColor (Color.YELLOW);
g.fillOval (100,100,100,100);

// Draw the eyes
g.setColor (Color.BLACK);
g.fillOval (129,128,12,24);
g.fillOval (159,128,12,24);

// Draw circle around face
g.drawOval(100,100,100,100);

// Draw smile
g.drawArc (125,160,50,14, 0, -180);
```

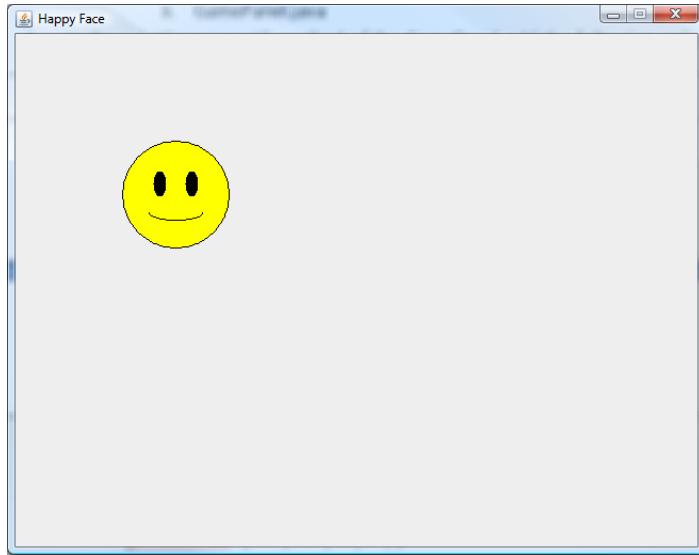


Figure 73 - Happy Face

Note: It may not look like a lot of work to get the Happy Face drawn but I had to do a bit of calculation in order to get the eyes and mouth set correctly around the yellow circle making up the face. It makes sense to use a drawing tool for our images!

Drawing Polygons

Polygons are any 2-dimensional shapes made of straight lines.



Figure 74 - A regular polygon (pentagon)

A polygon can be a regular polygon or irregular as shown below:

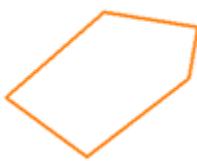


Figure 75 - An irregular polygon

To create a polygon you first instantiate a `Polygon` object and add the points making up the polygon (using the `addPoint()` method). The code that draws the polygon connects the points you add in sequence and closes the polygon by connecting the first and last point.

The following code prints a triangle:

```
Polygon poly = new Polygon();
// Make a triangle
poly.addPoint(100, 100);
poly.addPoint(150, 150);
poly.addPoint(50, 150);
g.drawPolygon(poly);
```

The drawing below is the result of creating a polygon and adding three points that are then drawn connected.

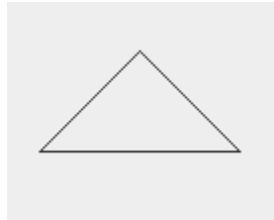


Figure 76 - Printing a triangle with polygon class

The method format is:

```
graphicsObj.drawPolygon(Polygon p)
```

Draws the outline of a polygon defined by the specified Polygon object p.

The method `fillPolygon()` can be used to draw a filled in polygon.



Figure 77 - Using fillPolygon()

The method format is:

```
graphicsObj.fillPolygon(Polygon p)
```

Fills a closed polygon.

Drawing Images

There is a Graphics method to draw images – `drawImage()` but before we can use `drawImage()` we have get the image and store as an `BufferedImage` object.

We use the static method `read()` from the `ImageIO` class to read in a `BufferedImage`.

The class `ImageIO` is commonly used to process an image file. Images come in various file formats. The common ones handled by `ImageIO` are:

File Format	Description	File extension
JPEG	This a common file format of photo quality images. The method used to represent the image is “lossy compression” which means that the data encoding compresses data by discarding (losing) some of it. The term JPEG is an acronym for “Joint Photographic Experts Group”.	.jpg
PNG	The Portable Network Graphics (PNG) is a raster graphics file format that supports lossless data compression. A raster graphics image is a bitmap (think dot matrix data structure) of pixels that is easily viewable on a computer monitor or printer.	.png
BMP	This bit map image format is a raster graphics image file format that was defined in order to have a device independent way to specify a color bit map image.	.bmp
WBMP	This image format stands for “Wireless Application Protocol bitmap format. This is a monochrome graphics file format intended for use on mobile devices. An image specified using monochrome is one using one color or shades of one color.	.wbmp
GIF	GIF stands for Graphics Interchange Format. It is a bitmap image format that was introduced by the one-time very popular dial-up service. This format supports animation (animated gif) and uses the Lempe-Ziv-Welch lossless data compression format. Note: A patent contention between CompuServe and Unisys threaten the use of GIF as a popular graphics format and hence led to the development of the patent-free format PNG.	.gif

These file formats are not the only graphics file formats but covers most of what you will encounter or use for our games.

The method format is:

```
public static BufferedImage read(InputStream input) throws Exception;
```

This method returns a `BufferedImage` object. The method decodes the image file specified as an `InputStream`.

There are many “flavors” of the static `read()` method. We will generally use the following:

```
BufferedImage myImage = ImageIO.read(getClass()
    .getResourceAsStream(<FILENAME>));
```

Now that we have the image in an `BufferedImage` object the next step is the draw the image using the `drawImage()` method. There are many “flavors” of the `drawImage()` method in the `Graphics` class. The basic one we will be using is:

The method format is:

```
drawImage(Image img, int x, int y, ImageObserver observer)
```

This method draws the specified image `img` where the top-left position of the image is at position `(x,y)` on the screen. The `observer` is any object that should be notified as the image is being processed.

We can use our `BufferedImage` `img` with this method since `BufferedImage` subclasses the `Image` class. For now we will use `null` value as the value for the `ImageObserver`.

Lab 2-4: Drawing an Image

1. Create a new Java Project call it `ReadImage`.
2. Make sure it has all the typical file layout under `src` we will use going forward:
 - a. `com.brainycode.platformer.main`
 - i. `Game.java` – Change the title to “Read/Display Image”
 - ii. `GamePanel.java`
3. Add a new resources folder
4. Add images folder to resources folder

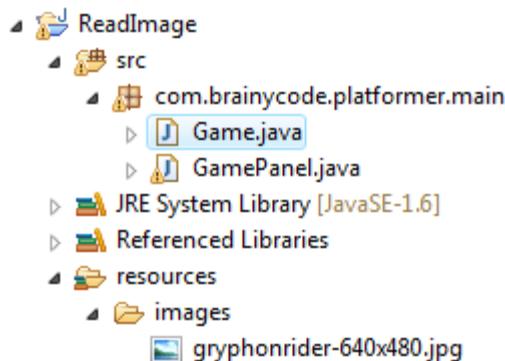


Figure 78 - Adding image resources folder

5. Find a 640x480 image (e.g. `gryphonrider-640x480.jpg`) and place in `resources/images` folder
6. Add the following code to `GamePanel`

Table 8 - GamePanel version for `ReadImage` project

```
package com.brainycode.platformer.main;

import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.image.BufferedImage;
import java.io.IOException;

import javax.imageio.ImageIO;
```

```

import javax.swing.JPanel;

public class GamePanel extends JPanel {

    private static final long serialVersionUID = 1L;

    // dimensions
    public static final int WIDTH = 640;
    public static final int HEIGHT = 480;

    BufferedImage img;

    public GamePanel() {
        super();
        setPreferredSize(new Dimension(WIDTH, HEIGHT));
        setFocusable(true);
        requestFocus();
        init();
    }

    private void init() {
        try {
            img =
ImageIO.read(getClass().getResourceAsStream("/images/gryphonrider-640x480.jpg"));
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        // first make background
        if (img != null) {
            g.drawImage(img, 0, 0, null);
        }
    }
}

```

The code adds a new `init()` method that obtains the image from the `InputStream`. In order to locate our resources we must add that folder to the `classpath`.

7. Right-click on the project name and select properties
8. Select “Java Build Path”
9. Click on “Libraries” tab
10. Click on “Add Class Folder...” button

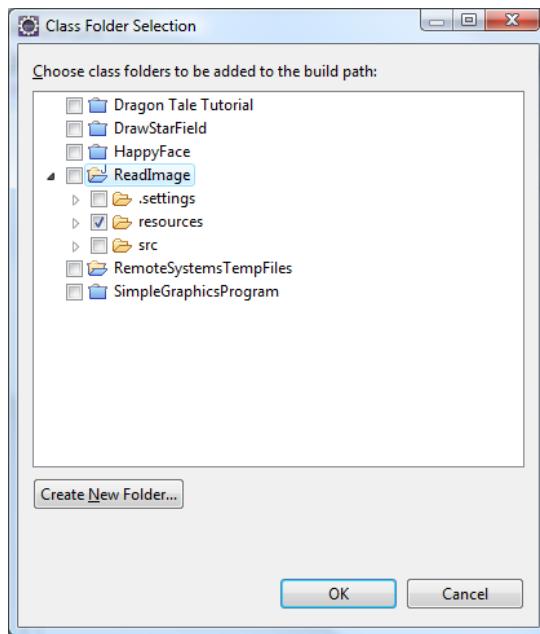


Figure 79- Adding resources folder to classpath

11. Click on resources folder and press OK.
12. Execute the program

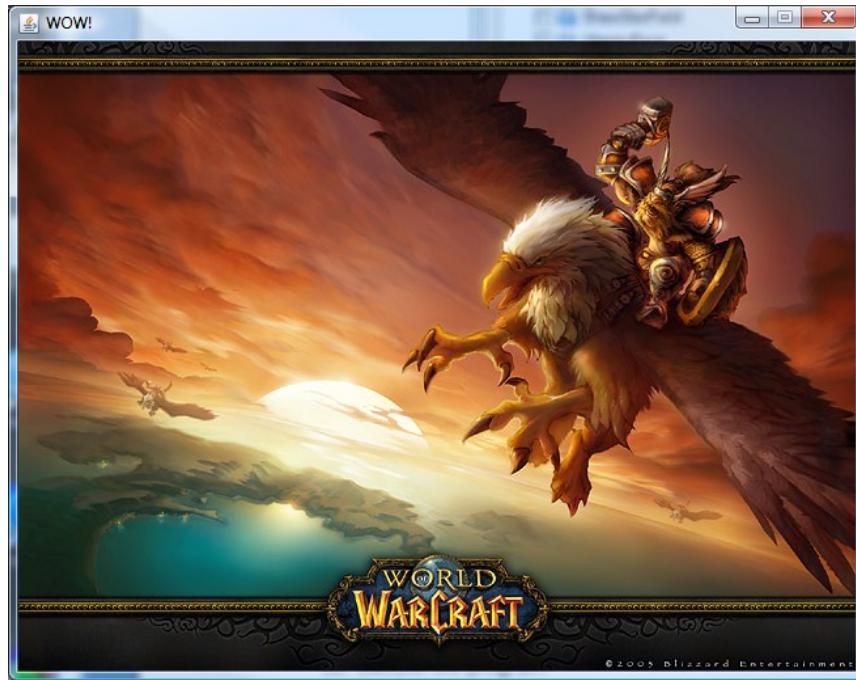


Figure 80 - Displaying WOW Image

Clipping
TBD

Graphics2D

The original java libraries provided very simple primitives to draw lines. In fact, one limitation was the fact that all lines could only be one pixel wide. The Java 2D API was created to address all the limitations and extend the drawing capability in the Java arsenal. Java 1.2 introduced the Java 2D API and new features to the graphics class via the Graphics2D class. At the time of its introduction the limitation in the language only allowed the underlying method to return a Graphics object hence Graphics2D extends Graphics class.

Taking another look at our paintComponent method:

```
public void paintComponent(Graphics g) {  
    super.paintComponent(g);  
    g.drawString("Java 2D!", 300, 220);  
}
```

The g object is actually a Graphics2D object and is typically cast to a Graphics2D object in order to get to the extended and new features added in the Java2D API.

```
Graphics2D g2 = (Graphics2D)g;
```

As the javadoc reads, “This Graphics2D class extends the Graphics class to provide more sophisticated control over geometry, coordinate transformations, color management, and text layout. This is the fundamental class for rendering 2-dimensional shapes, text and images on the Java platform.”

Making changes to the objects we draw

Alpha

TBD

Anti-aliasing

TBD

Scaling Images

TBD

Chapter 3 - Introducing the Game Loop

The key mechanic in the programming of the game engine is the implementation of the game loop. This is the area that drives your game engine. The basic function of the game loop is to perform the following actions:

```
while (game is not over) {
    check for user input;
    update all game objects;
    draw graphics;
}
```

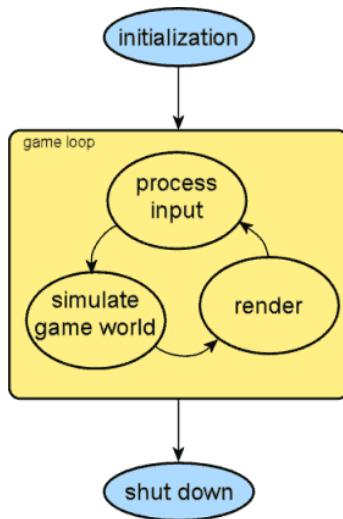


Figure 81 - Simple game loop (from: <https://sites.google.com/site/justinscsstuff/jogl-tutorial-3>)

The above figure depicts a simple game loop. The first thing to do is initialize the game components – create our game objects, read in images and sounds and set the starting point of the game. Within the game loop we iterate through the three main steps – process user input, update all game object (simulate game world) and then render or draw the current scene.

Process Input

We first check if the user has fired off any missiles or jumped to avoid the boulder coming the player's way since this will determine how the game objects will be created and destroyed. Players expect their input to get processed right away and since it determines what objects get created (e.g. creation of missile object when "fire" button is pressed) and destroyed it is vital that we capture all user input and make game updates to the screen in response.

The next chapter will cover keyboard processing and a later chapter will cover the use of other game devices (e.g. game controller or joystick).

Simulate Game World

The next step in the game loop is to update all game objects. If the player clicked on the “jump” button we need to start animating the actions of our game player jumping.



Figure 82 - Jump animation

It is in this part of the game loop that we process the AI (artificial intelligence), move enemies; resolve collisions (did the missile hit anything yet or is it off screen?).

Render

In this part of the game loop we display the current game screen. In this part we instruct each object to `draw()` itself (if it makes sense to do so since a particular game object can decide it is not visible or on screen and can skip the drawing).

Table 9 – The game loop in our program

```
public void run() {
    while(running) {
        // play the game
    }
}
```

It is currently missing all the key elements we need for a real game but we plan on adding that later - the start is there. The first thing that must be done is to set this up to run in its own thread.

What are Threads?

“The term thread is shorthand for thread of control, and a thread of control is, at its simplest, a section of code executed independently of other threads of control within a single program.”¹⁶

The notion of threading is “so ingrained in Java” that even our most basic and simple programs use it without our being aware of it. We should be accustomed to starting our Java programs by having the operating systems invoke our `main()` method. This starts your typical single-threaded application. A Java program allows you to spawn off one or more threads that will run independently from the main thread.

In a Java program each thread will have the following properties:

¹⁶ This section is composed from material in the “Java Threads” reference.

- ✚ Each thread will begin execution at a predefined well-known location. You are of course familiar with `main()` being the starting point of your application. You as a programmer get to decide the starting point of the other threads you start up.
- ✚ Each thread executes code from its starting location in an ordered, predefined sequence (the a given set of inputs)
- ✚ Each thread executes its code independently of the other threads in the program

Another thread that you know runs with your application is the garbage collector. But this thread is created and managed by the Java Virtual Machine (JVM). When you build GUI applications you know that the events (mouse click, keyboard entry, etc) are all handled by another thread managed outside your application code - Event Thread Dispatcher. You have to perform special actions to inform the system that you want to handle certain events within your program. Another type of thread that you don't manage but work with is the `paint()` event. When your GUI application or game requires painting your application has to implement or override special methods (e.g. `paint()` or `paintComponent()`) in order to draw the game display. In our programs we do not explicitly override the `paint()` method to draw on the screen but it essentially gets done because we subclass components that do invoke methods to refresh the screen.

Why do we need Threads in our game program?

Threading makes certain type of programs easier to construct. Dividing the program so that it can be split into separate tasks allows us to create conceptually (eventually) easier programs that coordinate the job the application is trying to execute.

Threading Experience

My first experience with threading came up when I was asked to fix a problem a particular application was having with respect to a bad user experience. The user used the application to manage different router¹⁷ devices. The job of the application was to download to the selected device a configuration file. The figure below shows you the setup.

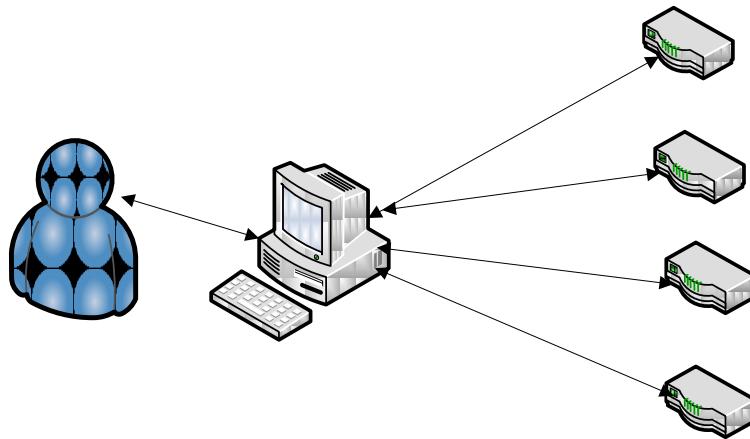


Figure 83 - User connecting to router devices

Very often the router device the user selected was not yet available and the application when reaching out to the device would wait several minutes (or at least the user reported it felt like several minutes) before the application would return from the “download request” operation to report back to the user that the operation failed because the router connection was not established. The user knew that if the application did not get a response from the router within several seconds that the router was not “up” and available yet but the actual time was more like minutes because of the built-in timeout the network connection software was set at. The application was written as a single-threaded application its logic was something like this:

```
try {
    connectToRouter(ipAddressForRouter);
    downloadConfigurationFile();
} catch (ConnectionTimedOutException ctoe) {
    // give the user the bad news ....after waiting for several minutes....what a waste of time!
}
```

The problem was in the `connectToRouter()` call. It would go off into the method trying to set up a connection to the router and the code waited and waited and waited until the connection timed out before returning with a connection timed out failure. The solution that I came up with (for what was available at the time) was to execute the `connectToRouter()` in its own thread and if the thread did not complete within a time limit that I set in some property file (5 seconds) than I would assume the router was not available. I then continued program execution and reported the bad news to the user in a more timely fashion. The user was much happier after I implemented this solution.

¹⁷ If you don't know what a router is that is fine just imagine some black box that uses some file called configuration file to figure out what to do.

“The popularity of threading increased when graphical interfaces became the standard for desktop computers because the threading system allowed the user to perceive better program performance.” In a game we have at least two things we want it to do at the same time “manage the screen display” and “execute the game loop.” In order to do this will require that we learn to create and manage threads.

The goal for our games programming is to create as few threads as possible but implement them we must. The most important thread for our program is the *Event Dispatcher Thread*. This is the thread that handles events (e.g. mouse click, keyboard entry, etc). You dictate which events you want to handle in your program by specifying an event listener. This topic will be covered in a future episode.

Creating a Thread

There are two ways to create threads.

- Create a class that extends `java.lang.Thread`
- Create a class that implements the interface `java.lang.Runnable`

In both cases your class must define a method named `run`.

Table 10 - the Thread run() method

```
public void run() {  
}
```

To terminate a thread just let the `run()` method exit naturally.

Let's recall that the `main()` method is also a thread that the operating systems starts. Let's look at a simple example.

Table 11 - SimpleThreadExample01.java

```
public class SimpleThreadExample01 extends Thread {  
  
    public void run() {  
        for (int i = 0; i < 1000; i++)  
            System.out.println("New thread executing " + i);  
    }  
  
    public static void main(String[] args) {  
        SimpleThreadExample01 myThread = new SimpleThreadExample01();  
        myThread.start();  
        for (int i = 0; i < 50; i++) {  
            System.out.println("Main thread executing " + i);  
        }  
    }  
}
```

In the above code the `main()` method creates and starts its own thread `myThread` by invoking the method `start()`. Invoking the `start()` method actually starts the execution of the `run()` method.

In the output you will see “New thread executing” messages mixed in with “Main thread executing” messages. This demonstrates that the `main()` thread and `myThread` are running at the same time. Since the machine is very fast each thread may print out many messages on the console before the second thread is given the CPU or maybe your output looks like the one below.

Table 12 - Output from SimpleThreadExample01

```
New thread executing 212
Main thread executing 247
New thread executing 213
Main thread executing 248
New thread executing 214
Main thread executing 249
New thread executing 215
Main thread executing 250
New thread executing 216
Main thread executing 251
New thread executing 217
Main thread executing 252
New thread executing 218
Main thread executing 253
```

In our program the main thread and the new thread (`myThread`) are two separate processes that are running. You can regard them as two separate programs running at the same time sharing the computer resources. The operating system is designed to give each program and each thread within each program a slice of the CPU to get work done. In our example, the work is going through a for loop.

The second way to create and start a thread is by using the `Runnable` interface.

Table 13 - SimpleThreadExample02.java

```
public class SimpleThreadExample02 implements Runnable {

    @Override
    public void run() {
        for (int i = 0; i < 1000; i++)
            System.out.println("New thread executing " + i);
    }

    public static void main(String[] args) {
        SimpleThreadExample02 runner = new SimpleThreadExample02();
        Thread myThread2 = new Thread(runner);
        myThread2.start();
        for (int i = 0; i < 500; i++) {
            System.out.println("Main thread executing " + i);
        }
    }
}
```

The output for the program should be quite similar to the previous one. Note, that in the example above we create a “runner” object based on our SimpleThreadExample02 class and provide the object as an argument to the Thread constructor for our thread object myThread2. We invoke the `run()` method of the thread the same way - by invoking the `start()` method of the thread.

To stop a thread you merely allow the `run()` method to exit. There are some deprecated methods in the `Thread` class that you should not use. One deprecated method is the `stop()` method and the other is the `suspend()` method.

At this time you don’t need to know much more about Threads and the code in this section does not use much more advanced ideas and concepts pertaining to threads.

When you start the main method of the program:

```
public static void main(String[] args) {
    JFrame window = new JFrame("Star Field");
    window.setContentPane(new GamePanel());
    window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    window.setResizable(false);
    window.pack();
    window.setLocationRelativeTo(null);
    window.setVisible(true);
}
```

There is a special method invoked by the graphics toolkit to create a hook with the platform’s underlying peer – `addNotify()`. This is a common method to establish the game thread and kick off the game engine. The graphics toolkit invokes the method when the component (`GamePanel`) `setVisible(true)` or `pack()` method is called.

```
public void addNotify() {
    super.addNotify();
    // Create game engine thread here
    //
}
```

Lab 3-1: A Quick Introduction to Using Ant

In this lab we will create a throw away “hello world” type program in order to illustrate how to use an Ant build file.

1. Make sure you have *.xml files associated with the Ant Build editor.
2. Select Window → Preferences → General → File Associations
3. If *.xml is not one of the file types listed click on “Add...” and enter “*.xml”
4. In the Associated editors panel select “Ant Editor” and click on “OK”
5. Create a new Java Project TestAntWithEclipse
6. Add the file Project.java with the code shown below:

Table 14 - Project.java

```
public class Project {

    public static void main(String[] args) {
        System.out.println("Wanna dance? Or Code?");
    }
}
```

7. Highlight the Project and add the file build.xml

8. Add the xml below to the build.xml file:

Table 15 - build.xml

```
<?xml version="1.0" ?>
<project default="main">
    <target name="main" depends="compile, jar">
        <echo>
            Building the .jar file.
        </echo>
    </target>

    <target name="compile">
        <javac srcdir=".src" destdir=".bin" includeantruntime="false" />
    </target>

    <target name="jar" depends="compile">
        <jar jarfile="Project.jar" basedir=".bin" includes="*.class">
            <manifest>
                <attribute name="Main-Class" value="Project.class" />
            </manifest>
        </jar>
    </target>
</project>
```

Note: When you type this into using Eclipse you may see errors crop up for targets that do not exist – just ignore and start to worry if any lines are flagged after you enter all the xml code.

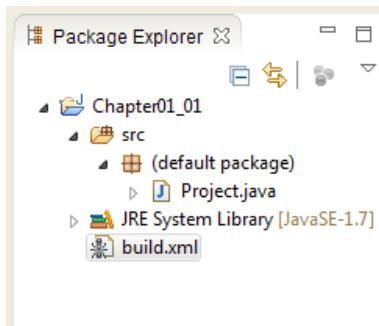


Figure 84 - Adding build.xml to the package

The package explorer should display the typical ant icon (see above) when you add a build.xml to the project.

9. Right-click on the build.xml file and select “Run As” then select “Ant Build...” to see all the possible target you can select.

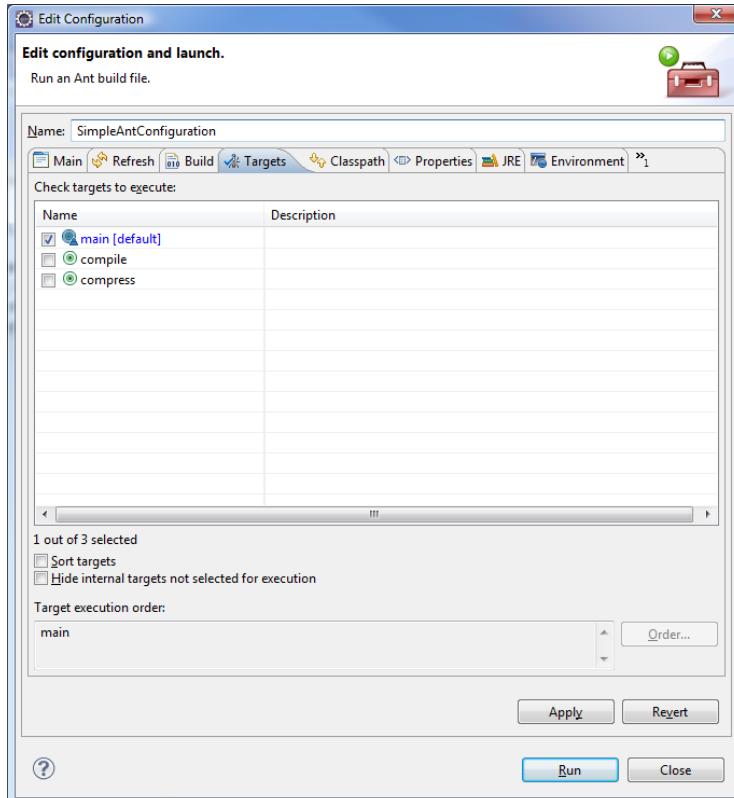


Figure 85 - Edit Configuration dialog

10. I changed the Name to “SimpleAntConfiguration” and left the default target “main” selected.
11. Click on Apply and then Run.

Table 16 - SimpleAntConfiguration output

```
Buildfile: C:\Users\figgy\workspace\ANT_TEST_PROJECTS\Chapter01_01\build.xml
compile:
[javac] Compiling 1 source file to
C:\Users\figgy\workspace\ANT_TEST_PROJECTS\Chapter01_01\bin
jar:
[jar] Building jar:
C:\Users\figgy\workspace\ANT_TEST_PROJECTS\Chapter01_01\Project.jar
main:
[echo] Building the .jar file.
[echo]
BUILD SUCCESSFUL
Total time: 2 seconds
```

After Ant builds your project you will have a Project.class file in the \bin directory and a Project.jar in the same directory as the build.xml.

12. Open an command prompt window and run/execute the Project.jar as shown below:

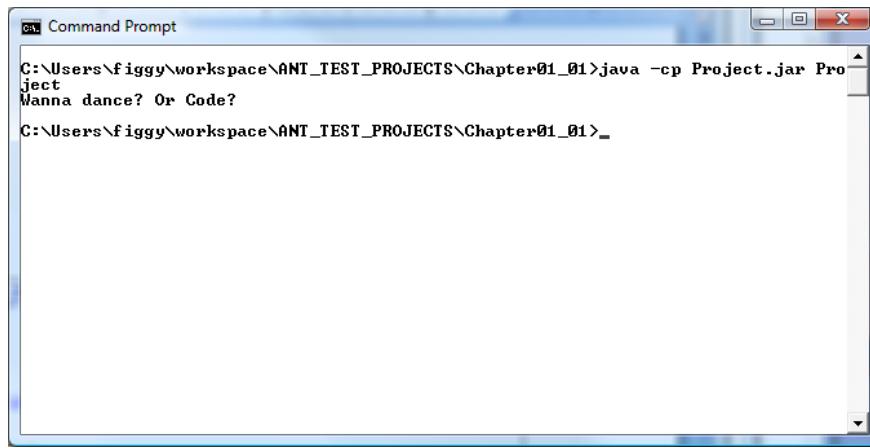


Figure 86 - Running the Ant created jar file from the command line

All Ant projects consist of one or more build files. The default build file name is build.xml.

```
<?xml version="1.0" ?>
```

All Ant build files start with the XML declaration statement indicating that the file is an xml document.

```
<project default="main">
:
</project>
```

The document element for all xml files is the <project> tag. The tag in our example uses the default attribute which specifies the target to invoke to kick start the build. This is a required attribute.

There are two other optional attributes that you can specify on the <project> tag:

- name – defines the project name
- basedir – the base directory from which all relative paths are resolved

In Ant a `target` corresponds to a set of tasks that you want Ant to execute.

In our `<project>` tag we specified the target to start with “main”, let’s examine this `<target>`.

```
<target name="main" depends="compile, jar">
    <echo>
        Building the .jar file.
    </echo>
</target>
```

The name is a required attribute in the `<target>` tag. We also use the `depends` attribute which contains a list of targets that must successfully execute before this target.

A target contains one or more tasks. In the above example after the targets listed execute “compile, jar” then the main target echo’s or prints to the console the message text:

main:

```
[echo]          Building the .jar file.

[echo]
```

Ant comes with many built in tasks – echo is just one.

```
<target name="compile">
    <javac srcdir=".src" destdir=".bin" includeantruntime="false" />
</target>
```

The compile task does the obvious – it compiles our one project file. The files are in the `srcdir` (source directory) `./src` and the destination of the `*.class` files shall be the same as the default directory when setting up an Eclipse Java project `./bin`. I added the `includeantruntime="false"` because there is an error message which seems to print out if it is missing.

After the successful execution of the task `javac` in the `compile` target Ant then executes the `jar` target:

```
<target name="jar" depends="compile">
    <jar jarfile="Project.jar" basedir=".bin" includes="*.class">
        <manifest>
            <attribute name="Main-Class" value="Project.class" />
        </manifest>
```

```
</jar>
</target>
```

The `jar` target contains the `jar` task which instructs Ant to create a jar file named `Project.jar`. In addition, we are instructing Ant to add a manifest detailing the starting main class file. The `basedir` attribute is the reference directory from where to jar. The `includes` attribute is a comma or space separated list of file patterns that must be included. A manifest file describes various aspects of your application according to the Jar file specification rules¹⁸.

When all is said and done you should see the following files:

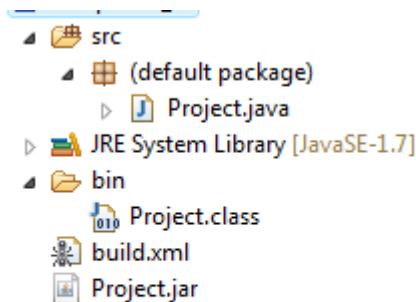


Figure 87 - Viewing all the files in Eclipse Project Explorer

You can see how running the Ant `build.xml` file generated the `Project.class` and created the `Project.jar`.

Fact: The `Project.class` probably got created before you even had a chance to run the Ant build file if you the “Build Automatically” flag on. Some folks turn it off by selecting `Project → Build Automatically` and clicking it off.

You can find out the various Ant tasks by heading over to:

<http://ant.apache.org/manual/tasksoverview.html>

See Appendix X for a list of Ant tasks you can use in your projects.

Lab 3-2: Bouncing Ball

This lab we take the code we created in Chapter 1 under the JUnit exercise and give life to the ball so we can actually see it move across the screen and bounce off the walls!

1. Create a Java Project (e.g. `BallBouncing`).
2. Create the package `com.brainycode.main`
 - a. Copy from previous lab the `Game` and `GamePanel` java classes
3. Create the package `com.brainycode.platformer.entities` and `com.brainycode.platformer.math`

¹⁸ See <http://docs.oracle.com/javase/7/docs/technotes/guides/jar/jar.html>

4. Add a test and lib folder. Make sure the test folder is set up as a “source folder” (Right-click on the folder, select Build Path, Use As Source Folder)
5. Add the same packages to the test folder as specified in step 3.
6. Add the file Vector2D.java to the com.brainycode.platformer.math package

Table 17 - Vector2D.java

```

package com.brainycode.platformer.math;
< b>/***
 * This is a class to represent a 2D vector for displacement of objects.
 * @author figgy
 */

```

```

public class Vector2D {

    /** The x position on the screen of the normalized vector (x,y) */
    private int x;
    /** The y position on the screen of the normalize vector (x,y) */
    private int y;

    /** This creates a vector setting the (x,y) position
     *      to the values provided.
     *
     * @param i  the x position of the normalized vector (x,y)
     * @param j the y position of the normalized vector (x,y)
     */
    public Vector2D(int i, int j) {
        x = i;
        y = j;
    }

    public int getX() {
        return x;
    }

    public void setX(int x) {
        this.x = x;
    }

    public int getY() {
        return y;
    }

    public void setY(int y) {
        this.y = y;
    }

    /**
     *      Adds the vector components provided to the current vector
     *
     * @param vector2  a normalized vector to be added
     */
    public void add(Vector2D vector2) {
}

```

```

        x = x + vector2.getX();
        y = y + vector2.getY();
    }

    /**
     * Multiplies each component of the vector by the provided argument
     *
     * @param i
     */
    public void multiply(int i) {
        x *= i;
        y *= i;
    }
}

```

I added simple javadoc comments in order to see the results when we do a build. In the future we will try to be more informational and careful. The Vector2D is not our final version of this class but merely a version to illustrate a simple game loop and how Eclipse, Ant and JUnit are used together to build applications.

7. Add the Ball.java to the com.brainycode.platformer.entities package

Table 18 - Ball.java

```

package com.brainycode.platformer.entities;

import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.Point;

import com.brainycode.platformer.main.GamePanel;
import com.brainycode.platformer.math.Vector2D;

/**
 * This class represents a small ball that moves across the screen.
 * @author figgy
 */
public class Ball {

    /**
     * The ball's top-left position on the screen. The class uses the
     * GamePanel.WIDTH and GamePanel.HEIGHT for the dimensions of the screen
     */
    private Point point;
    /**
     * The radius of the ball in pixels
     */
    private int radius;
    /**

```

```

        * The vector displacement indicating the speed and direction of the ball.
    */
private Vector2D displacement;

/**
 * Creates a Ball object located at position (x,y) on the screen.
 * The ball's radius is set to r pixels.
 *
 * @param x      The left position on the screen ranging from
0..GamePanel.WIDTH
 * @param y      The top position on the screen ranging from
0..GamePanel.HEIGHT
 * @param r      The radius of the ball. The ball will reside in a
rectangle of width 2 * r
 */
public Ball(int x, int y, int r) {
    point = new Point();
    point.x = x;
    point.y = y;
    radius = r;
}

/**
 * Returns the x position or left-most position of the ball on the screen.
 * @return the x position
 */
public int getX() {
    return point.x;
}

public void setX(int x) {
    point.x = x;
}

public void setY(int y) {
    point.y = y;
}

/**
 * Returns the y position or top-most position of the ball on the screen.
 * @return the y position
 */
public int getY() {
    return point.y;
}

public int getRadius() {
    return radius;
}

public void setRadius(int radius) {
    this.radius = radius;
}

public Vector2D getDisplacement() {

```

```

        return displacement;
    }

    public void setDisplacement(Vector2D displacement) {
        this.displacement = displacement;
    }

    /** We update the ball's centerPoint position to always remain on the screen
     * from x ranges from 0..GamePanel.WIDTH-BALL RADIUS-1
     * and y ranges from 0..GamePanel.HEIGHT-BALL RADIUS-1
     */
    public void update() {
        point.x += displacement.getX();
        point.y += displacement.getY();

        // see if we went off the left side
        if ( getX() < 0) {
            // bring the ball back to the screen and reverse course
            setX(0);
            // change direction in X
            displacement.setX(-displacement.getX());
        }

        // see if we went off the right side
        if (getX() > GamePanel.WIDTH - 2 * radius - 1) {
            setX(GamePanel.WIDTH - 2 * radius - 1);
            displacement.setX(-displacement.getX());
        }

        // see if we went off the top
        if (getY() < 0) {
            setY(0);
            displacement.setY(-displacement.getY());
        }

        if (getY() > GamePanel.HEIGHT - 2 * radius -1 ) {
            setY(GamePanel.HEIGHT - 2 * radius -1);
            displacement.setY(-displacement.getY());
        }

        // assert x and y values will display the ball on the screen
    }

    /**
     * Draws the ball on the screen and a Ball Position message
     * @param g graphics object for drawing
     */
    public void draw(Graphics2D g) {
        g.setColor(Color.BLUE);
        g.fillOval(getX(), getY(), getRadius() * 2, getRadius() * 2);
        g.setColor(Color.BLACK);
        g.drawString("Ball Position: (" + getX() + "," + getY() + ")", 500, 460);
    }
}

```

```
}
```

The Ball and Vector2D classes come from our JUnit tutorial from Chapter 1 so they should require few explanations other than the fact that Ball class will no longer be a fictitious conceptual ball but will be something we actually see on the screen.

The update() method moves the ball on the screen using the displacement vector which defines its direction and speed on the screen. The new method is the draw() method which draws the ball and text detailing its current position on the screen.

We expect that the game loop will manage to invoke the update() method for the ball object and when the screen gets drawn that a call to the Ball's draw() method will be made.

8. Add/Update GamePanel.java

Table 19 - GamePanel.java

```
package com.brainycode.platformer.main;

import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;

import javax.swing.JPanel;

import com.brainycode.platformer.entities.Ball;
import com.brainycode.platformer.math.Vector2D;

public class GamePanel extends JPanel implements Runnable{

    private static final long serialVersionUID = 1L;
    /** The screen WIDTH of the drawing panel */
    public static final int WIDTH = 640;
    /** The screen HEIGHT of the drawing panel */
    public static final int HEIGHT = 480;

    /** The ball object we will move around on the screen */
    private Ball ball;

    /** The game thread that represents our game loop */
    private Thread thread;
    /** A boolean value indicating that our game is running */
    private boolean running;
    /** The number of frames per second we want the game to run in */
    private int FPS = 60;
    /** The time slice that should elapse before we update the screen */
    private long targetTime = 1000 / FPS;

    public GamePanel() {
        super();
        setPreferredSize(new Dimension(WIDTH, HEIGHT));
    }
}
```

```

        setFocusable(true);
        requestFocus();
    }

    /** This is invoked automatically but the AWT framework when our component
     * the GamePanel is added to the screen. We tap into this method in order to
     * start up our game loop in its own thread.
    */
    public void addNotify() {
        super.addNotify();
        if (thread == null) {
            thread = new Thread(this);
            thread.start();
        }
        running = true;
    }

    /** Initializes our game object - the ball and sets it's initial
     * direction and speed on the screen.
    */
    public void init() {
        ball = new Ball(100,100, 10);
        Vector2D displacement = new Vector2D(2,3);
        ball.setDisplacement(displacement);
    }

    /** Updates the ball's position. This should be called for every frame
     * or at least 60 times per second.
    */
    public void update() {
        ball.update();
    }

    /** Updates the game objects */
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        if (ball != null) {
            ball.draw((Graphics2D) g);
        }
    }

    /** The game loop:
     *  This is simple game loop that only:
     *  - updates the game world
     *  - renders the screen (indirectly by requesting a repaint)
     *
    */
    @Override
    public void run() {
        init();
        long start;
        long elapsed;
        long wait;

        while (running) {

```

```

        start = System.nanoTime();

        update();

        elapsed = System.nanoTime() - start;
        wait = targetTime - elapsed / 1000000;
        if (wait < 0) wait = 5;
        try {
            Thread.sleep(wait);
        } catch (Exception e) {
            e.printStackTrace();
        }
        repaint();
    }

}

```

This version of the GamePanel introduces our initial attempt at a game loop. It will be good enough for this example but we will develop it and get closer to our desired game loop.

For now, we have the game loop starting when the GamePanel is added to the Frame or Window. The run() method initializes things which in our case means the creation of a ball object that will be moving on the screen. The while loop is the interesting part.

```
start = System.nanoTime();
```

We first get the current time in nanoseconds.

```
update();
```

We then update the game which in our case means moving the ball along using the displacement vector we initialized it with.

```
elapsed = System.nanoTime() - start;
```

We now check how much time has elapsed since the loop started. Now one of three things can be true:

- Exactly `targetTime` has elapsed – which is perfect that means we are on target for meeting our desired frames per second (FPS)
- Less than `targetTime` has elapsed – so we should wait by calculating the difference
- More than `targetTime` has elapsed – oh, oh, we are not going to be able to update the screen as fast as we thought.

If wait == 0 then we meet criteria #1 above and just request a repaint, if wait > 0 then we wait the difference by requesting to sleep:

```
Thread.sleep(wait);
```

If wait < 0 we still wait at least 5 milliseconds to give other threads and parts of the program a chance to clean up and move their threads along.

```
if (wait < 0) wait = 5;
```

If a machine is especially slow and cannot meet the desired frame rate we usually do more than what we outlined in this game loop. We want all users regardless of the machine to experience the game the same way. We will discuss this more later.

The last line in the while loop is the statement:

```
repaint();
```

The repaint() is informs AWT that we would like it to perform its magic and invoke our paint methods so we can draw all the object on the screen.

9. Add/Update Game.java

Table 20 - Game.java

```
package com.brainycode.platformer.main;

import javax.swing.JFrame;
/**
 * This class is the application's main window it creates a JFrame window
 * and populates it with the GamePanel.
 *
 * @author figgy
 */
public class Game {

    public static void main(String[] args) {
        JFrame window = new JFrame("Bouncing Ball");
        window.setContentPane(new GamePanel());
        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        window.setResizable(false);
        window.pack();
        window.setLocationRelativeTo(null);
        window.setVisible(true);
    }
}
```

10. Copy your latest copy of junit (e.g. junit-4.9b2.jar) into the lib directory

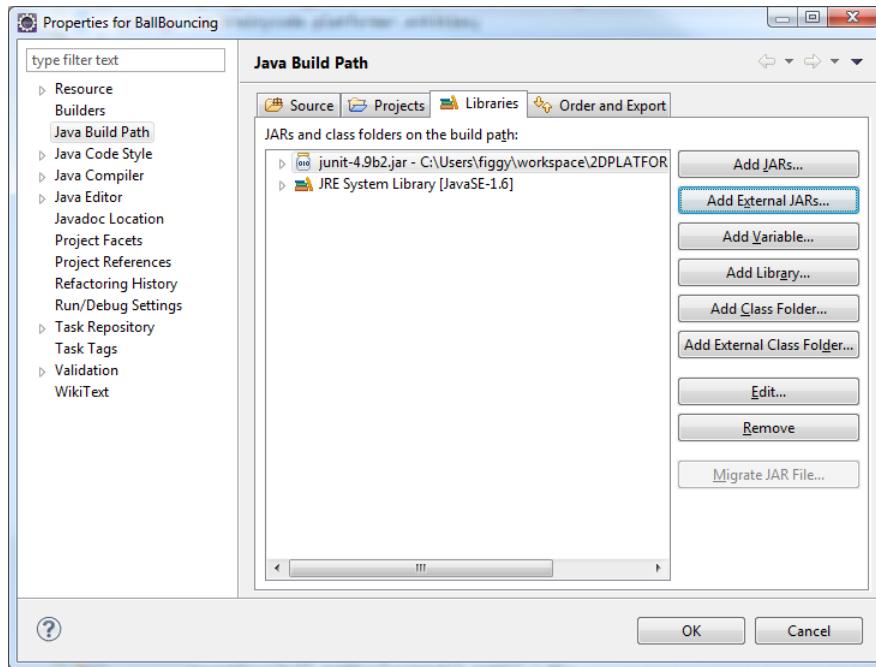


Figure 88 - Adding junit jar to the build path

11. Add the external to the build path (so Eclipse does not complain)
12. Create the test case TestBall.java in test directory under the package
com.brainycode.platformer.entities. Note: the test cases are exactly as the code in Chapter 1.

Table 21 - TestBall.java

```
package com.brainycode.platformer.entities;

import static org.junit.Assert.*;

import org.junit.Test;

import com.brainycode.platformer.main.GamePanel;
import com.brainycode.platformer.math.Vector2D;

public class TestBall {

    @Test
    public void testConstructor() {
        Ball ball = new Ball(100,100, 50);
        assertTrue(ball.getX() == 100);
        assertTrue(ball.getY() == 100);
    }

    @Test
    public void moveBall() {
        Ball ball = new Ball(100,100, 50);
    }
}
```

```

        Vector2D displacement = new Vector2D(2,3);
        ball.setDisplacement(displacement);
        ball.update();
        assertTrue(ball.getX() == 102);
        assertTrue(ball.getY() == 103);
    }

    @Test
    public void moveBallOffRightScreen() {
        Ball ball = new Ball(589,100, 50);
        Vector2D displacement = new Vector2D(2,3);
        ball.setDisplacement(displacement);
        ball.update();
        assertTrue(ball.getX() == GamePanel.WIDTH - (2 * ball.getRadius()) -1);
        assertTrue(ball.getY() == 103);

        // check if the ball has reversed direction
        assertTrue(ball.getDisplacement().getX() < 0);
    }

    @Test
    public void moveBallOffLeftScreen() {
        Ball ball = new Ball(1,100, 50);
        // set ball to move left and down
        Vector2D displacement = new Vector2D(-2,3);
        ball.setDisplacement(displacement);
        ball.update();
        assertTrue(ball.getX() == 0);
        assertTrue(ball.getY() == 103);

        // check if the ball has reversed direction
        assertTrue(ball.getDisplacement().getX() > 0);
    }

    @Test
    public void moveBallOffTopScreen() {
        Ball ball = new Ball(200,0, 50);
        Vector2D displacement = new Vector2D(2,-3);
        ball.setDisplacement(displacement);
        ball.update();
        assertTrue(ball.getY() == 0);
        assertTrue(ball.getX() == 202);

        // check if the ball has reversed direction
        assertTrue(ball.getDisplacement().getY() > 0);
    }

    @Test
    public void moveBallOffBottomScreen() {
        Ball ball = new Ball(200, 378, 50);
        // set ball to move left and down
        Vector2D displacement = new Vector2D(2,3);
        ball.setDisplacement(displacement);
        ball.update();
        assertTrue(ball.getY() == GamePanel.HEIGHT - ball.getRadius() * 2 - 1);
    }
}

```

```

        assertTrue(ball.getX() == 202);

        // check if the ball has reversed direction
        assertTrue(ball.getDisplacement().getY() < 0);
    }

}

```

13. Add the TestVector2D.java to the test folder package com.brainycode.platformer.math

```

package com.brainycode.platformer.math;
import static org.junit.Assert.*;

import org.junit.Test;

public class TestVector2D {

    @Test
    public void testConstructor() {
        Vector2D displacement = new Vector2D(2, 3);
        assertTrue(displacement.getX() == 2);
        assertTrue(displacement.getY() == 3);
    }

    @Test
    public void testAdd() {
        Vector2D displacement = new Vector2D(2, 3);

        Vector2D vector2 = new Vector2D(-1, -1);
        displacement.add(vector2);
        assertTrue(displacement.getX() == 1);
        assertTrue(displacement.getY() == 2);
    }

    @Test
    public void testMultiply() {
        Vector2D displacement = new Vector2D(2, 3);

        displacement.multiply(-1);
        assertTrue(displacement.getX() == -2);
        assertTrue(displacement.getY() == -3);
    }
}

```

At this point we can run each testcase individually but this is where a build tool shines. We will create an Ant build.xml file that will do the following:

- Compile the java files

- Compile the test files
- Execute the test cases and create reports
- Jar the *.class files into an executable jar package
- Create javadocs from the javadoc comments we added to the code

14. Create the following build.xml file and run:

```
<?xml version="1.0"?>
<project name="Ant-BouncingBall" default="main" basedir=".">
    <!-- create a log file -->
    <record name="build.log" loglevel="verbose" action="start" />

    <!-- Sets variables which can later be used. -->
    <!-- The value of a property is accessed via ${} -->
    <property name="src.dir" location="src" />
    <property name="test.dir" location="test" />
    <property name="build.dir" location="bin" />
    <property name="dist.dir" location="dist" />
    <property name="docs.dir" location="docs" />
    <property name="lib.dir" location="lib" />
    <property name="reports.tests" location="reports" />

    <!-- Deletes the existing build, docs and dist directory-->
    <target name="clean">
        <delete dir="${build.dir}" />
        <delete dir="${docs.dir}" />
        <delete dir="${dist.dir}" />
        <delete dir="${reports.tests}" />
    </target>

    <!-- Creates the build, docs and dist directory-->
    <target name="makedir">
        <mkdir dir="${build.dir}" />
        <mkdir dir="${docs.dir}" />
        <mkdir dir="${dist.dir}" />
        <mkdir dir="${reports.tests}" />
    </target>

    <!-- Compiles the java code (including the usage of library for JUnit -->
    <target name="compile" depends="clean, makedir">
        <javac srcdir="${src.dir}" destdir="${build.dir}"
includeanruntime="false">
            </javac>
            <javac srcdir="${test.dir}" destdir="${build.dir}"
includeanruntime="false">
                <classpath>
                    <path element location="${lib.dir}/junit-4.9b2.jar" />
                </classpath>
            </javac>
    </target>
```

```

<target name="compileRunTestCases" depends="clean, makedir, compile">
    <junit printsummary="yes" haltonfailure="yes">
        <classpath>
            <pathelement location="${lib.dir}/junit-4.9b2.jar" />
            <pathelement location="${build.dir}" />
        </classpath>

        <formatter type="plain" />

        <batchtest fork="yes" todir="${reports.tests}">
            <fileset dir="${test.dir}">
                <include name="**/*Test*.java" />
                <exclude name="**/AllTests.java" />
            </fileset>
        </batchtest>
    </junit>
</target>

<!-- Creates Javadoc -->
<target name="docs" depends="compile">
    <javadoc packagenames="src" sourcepath="${src.dir}"
destdir="${docs.dir}">
        <!-- Define which files / directory should get included, we
include all -->
        <fileset dir="${src.dir}">
            <include name="**" />
        </fileset>
    </javadoc>
</target>

<!--Creates the deployable jar file -->
<target name="jar" depends="compile">
    <jar destfile="${dist.dir}\BouncingBall.jar" basedir="${build.dir}">
        <manifest>
            <attribute name="Main-Class"
value="com.brainycode.platformer.main.Game" />
        </manifest>
    </jar>
</target>

<target name="main" depends="compile, jar, docs">
    <description>Main target</description>
</target>

<target name="mainWithTest" depends="compileRunTestCases, jar, docs">
    <description>Main and Test target</description>
</target>

</project>

```

The Ant script introduces a key feature of Ant – the use of *properties*. Properties are name-value pairs (similar to how a Java developer uses properties files). It allows you to introduce constants into your program so you do not repeatedly reference values that can change.

```
<property name="src.dir" location="src" />
```

You reference a property value as: \${src.dir}. This allows you to place all name/value pairs in one place in your script and reference throughout.

Since admittedly I was having issues with the junit test cases I added the line:

```
<!-- create a log file -->
<record name="build.log" loglevel="verbose" action="start"/>
```

The above line creates a detailed log file named build.log. The problem was I needed to add an additional class path but the error messages displayed by Ant was not too helpful.

The next section contains all the property names I use throughout the script.

```
<!-- The value of a property is accessed via ${} -->
<property name="src.dir" location="src" />
<property name="test.dir" location="test" />
<property name="build.dir" location="bin" />
<property name="dist.dir" location="dist" />
<property name="docs.dir" location="docs" />
<property name="lib.dir" location="lib" />
<property name="reports.tests" location="reports" />
```

src.dir – is the directory of the source Java class files. I refer to them in the compilation area.

test.dir – is the directory name of the test cases, again I need to refer to them for compiling and running my junit test cases.

build.dir – is the location of all the *.class files

dist.dir – is the directory where I will place my jar file

docs.dir – is the directory for the javadoc files I plan on creating (hence the reason for the javadoc¹⁹ comments in a program that truly does not require it).

lib.dir – is the directory where any external library files will be placed and referenced

reports.tests – is the directory where the junit reports will be kept

¹⁹ Javadoc makes sense when you plan on creating code to share with the world. This simple program does not qualify but the idea here is to show you how to include as part of the build.

There are two possible targets to use when executing the Ant build but the Run As → Ant Build... (when right-clicking on the build.xml) shows us the following:

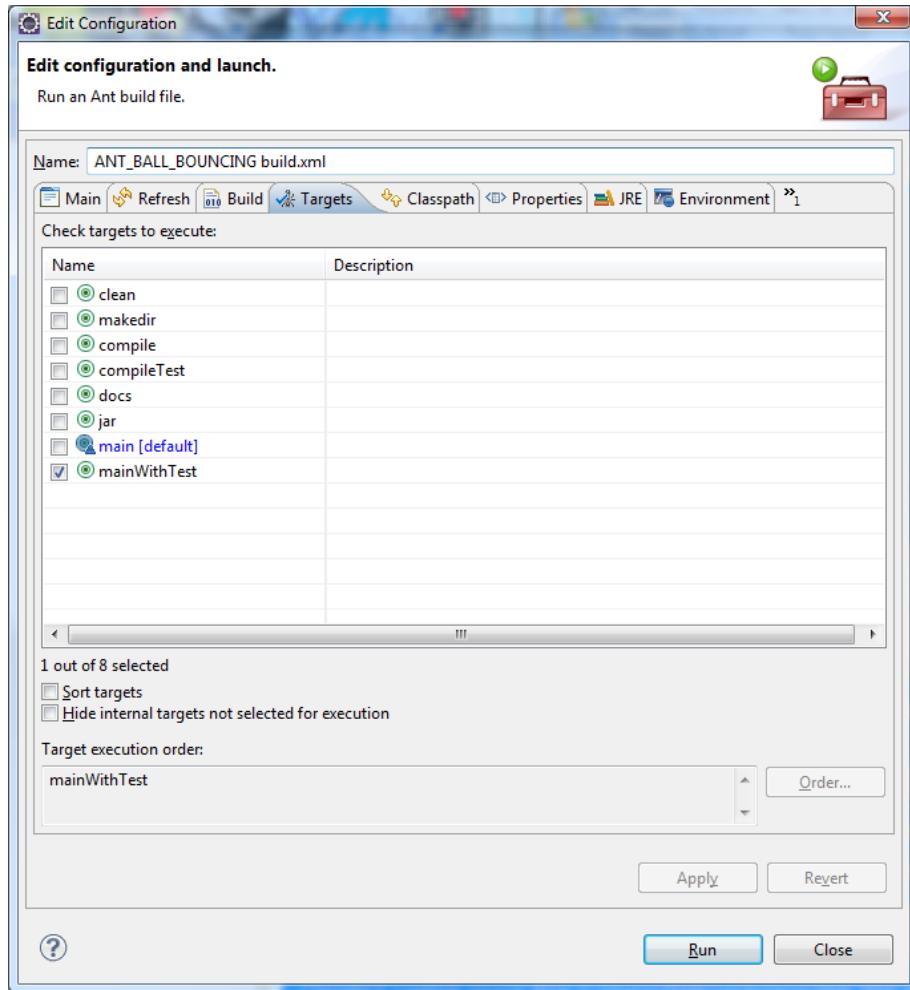


Figure 89 - Ant Targets -(main is the default)

The default is main (for those times when you want to skip the testcases). Here are the two key targets that one can choose:

```
<target name="main" depends="compile, jar, docs">
    <description>Main target</description>
</target>

<target name="mainWithTest" depends="compileRunTestCases, jar, docs">
    <description>Main and Test target</description>
</target>
```

Both are similar except the compileTest executes the jUnit test cases.

The other targets can easily be figured out. The result of executing the Ant script follows:

Table 22 - Executing the “main” Ant script

```

Buildfile: C:\Users\figgy\workspace\2DPLATFORMER\BallBouncing\build.xml
clean:
[delete] Deleting directory C:\Users\figgy\workspace\2DPLATFORMER\BallBouncing\bin
makedir:
[mkdir] Created dir: C:\Users\figgy\workspace\2DPLATFORMER\BallBouncing\bin
[mkdir] Created dir: C:\Users\figgy\workspace\2DPLATFORMER\BallBouncing\docs
[mkdir] Created dir: C:\Users\figgy\workspace\2DPLATFORMER\BallBouncing\dist
[mkdir] Created dir: C:\Users\figgy\workspace\2DPLATFORMER\BallBouncing\reports
compile:
[javac] Compiling 4 source files to
C:\Users\figgy\workspace\2DPLATFORMER\BallBouncing\bin
[javac] Compiling 2 source files to
C:\Users\figgy\workspace\2DPLATFORMER\BallBouncing\bin
jar:
[jar] Building jar:
C:\Users\figgy\workspace\2DPLATFORMER\BallBouncing\dist\BouncingBall.jar
docs:
[javadoc] Generating Javadoc
[javadoc] Javadoc execution
[javadoc] Loading source file
C:\Users\figgy\workspace\2DPLATFORMER\BallBouncing\src\com\brainycode\platformer\entities\Ball.java...
[javadoc] Loading source file
C:\Users\figgy\workspace\2DPLATFORMER\BallBouncing\src\com\brainycode\platformer\main\Game.java...
[javadoc] Loading source file
C:\Users\figgy\workspace\2DPLATFORMER\BallBouncing\src\com\brainycode\platformer\main\GamePanel.java...
[javadoc] Loading source file
C:\Users\figgy\workspace\2DPLATFORMER\BallBouncing\src\com\brainycode\platformer\math\Vector2D.java...
[javadoc] Constructing Javadoc information...
[javadoc] Standard Doclet version 1.6.0_20
[javadoc] Building tree for all the packages and classes...
[javadoc] Building index for all the packages and classes...
[javadoc] Building index for all classes...
main:
BUILD SUCCESSFUL
Total time: 9 seconds

```

If you run the testcases you will see the additional lines:

```

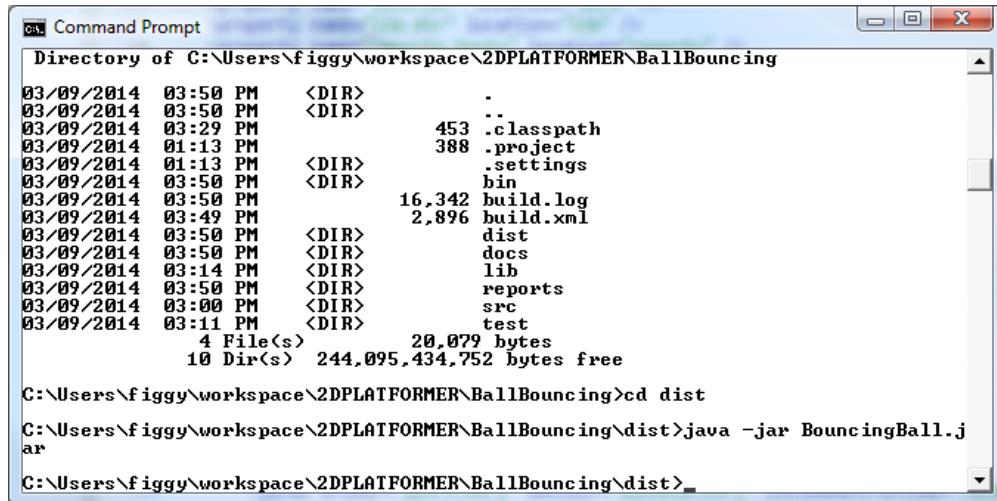
compileRunTestCases:
[junit] Running com.brainycode.platformer.entities.TestBall
[junit] Tests run: 6, Failures: 0, Errors: 0, Time elapsed: 0.025 sec
[junit] Running com.brainycode.platformer.math.TestVector2D
[junit] Tests run: 3, Failures: 0, Errors: 0, Time elapsed: 0.017 sec

```

And report files shall be generated.

15. Navigate to the jar file and execute to test (or just double-click on the jar in Windows Explorer).

The instruction to execute from the command line is “java –jar BouncingBall.jar”



```

C:\ Command Prompt
Directory of C:\Users\figgy\workspace\2DPLATFORMER\BallBouncing
03/09/2014  03:50 PM  <DIR>      .
03/09/2014  03:50 PM  <DIR>      ..
03/09/2014  03:29 PM          453 .classpath
03/09/2014  01:13 PM          388 .project
03/09/2014  01:13 PM  <DIR>      .settings
03/09/2014  03:50 PM  <DIR>      bin
03/09/2014  03:50 PM          16,342 build.log
03/09/2014  03:49 PM          2,896 build.xml
03/09/2014  03:50 PM  <DIR>      dist
03/09/2014  03:50 PM  <DIR>      docs
03/09/2014  03:14 PM  <DIR>      lib
03/09/2014  03:50 PM  <DIR>      reports
03/09/2014  03:00 PM  <DIR>      src
03/09/2014  03:11 PM  <DIR>      test
                           4 File(s)    20,079 bytes
                           10 Dir(s)   244,095,434,752 bytes free

C:\Users\figgy\workspace\2DPLATFORMER\BallBouncing>cd dist
C:\Users\figgy\workspace\2DPLATFORMER\BallBouncing\dist>java -jar BouncingBall.jar
C:\Users\figgy\workspace\2DPLATFORMER\BallBouncing\dist>_

```

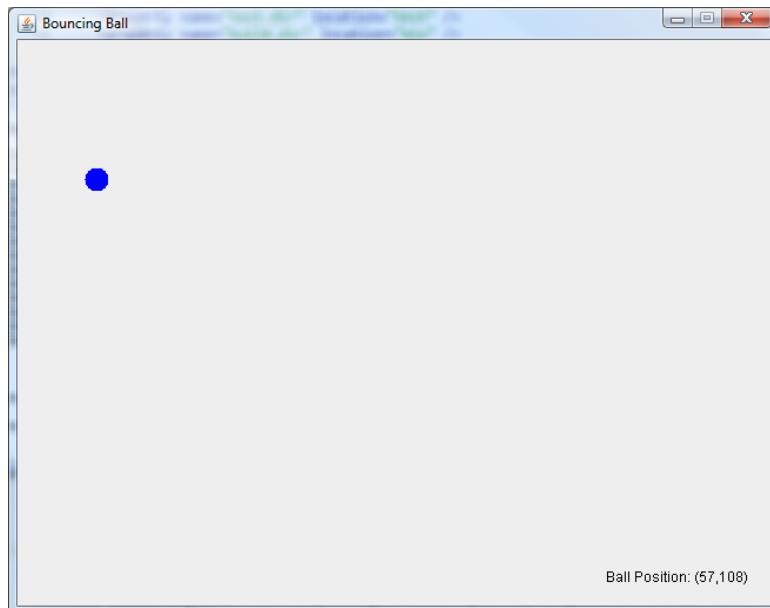


Figure 90 - Our simple ball example

So at this point we have demonstrated how to use Ant and JUnit within the Eclipse environment. In addition, we threw in the use and creation of javadocs, test reports and an executable jar file.

We will illustrate our game engine by displaying an animated GIF. An animated GIF is an image that consists of several frames that when shown in sequence display an animated or moving scene.

Note: This program will run fine under Eclipse without the Ant build file. The Ant build file is useful for a run where we want to build our final jar file and run all the test cases.

Lab 3-3: Displaying an animated GIF

In this lab we use code I located online that processes an animated gif file. The intentions of this lab is to illustrate how to decipher code you find online (GifDecoder.java) and incorporate into your programs. In addition, since an animated gif consist of several image frames that are shown in sequence in order to give the illusion of movement or animation this is an ideal use of our game loop where an update() moves to the next frame and draw() draws it to the screen. We should see the animated gif on screen under our control. Note: We do not use Ant for this project.

1. Create a new Java Project – AnimatedGIF
 - a. Add Game and GamePanel classes from a previous Lab.
2. Create the package com.brainycode.external.imageprocessing
3. Add the Java class GifDecoder.java (found online at: <http://www.java2s.com/Code/Java/2D-Graphics-GUI/GiffileEncoder.htm>)

Table 23 - GifDecoder.java

```
package com.brainycode.external.imageprocessing;

import java.awt.AlphaComposite;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics2D;
import java.awt.Rectangle;
import java.awt.image.BufferedImage;
import java.awt.image.DataBufferInt;
import java.io.BufferedInputStream;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.URL;
import java.util.ArrayList;

/**
 * Class GifDecoder - Decodes a GIF file into one or more frames. <br>
 *
 * <pre>
 * Example:
 *     GifDecoder d = new GifDecoder();
 *     d.read("sample.gif");
 *     int n = d.getFrameCount();
 *     for (int i = 0; i < n; i++) {
 *         BufferedImage frame = d.getFrame(i); // frame i
 *     }
 * </pre>
 */
```

```
*      int t = d.getDelay(i); // display duration of frame in milliseconds
*      // do something with frame
*
*  }
* </pre>
*
* No copyright asserted on the source code of this class. May be used for any
* purpose, however, refer to the Unisys LZW patent for any additional
* restrictions. Please forward any corrections to kweiner@fmsoftware.com.
*
* @author Kevin Weiner, FM Software; LZW decoder adapted from John Cristy's
*         ImageMagick.
* @version 1.03 November 2003
*
*/
public class GifDecoder {

    /**
     * File read status: No errors.
     */
    public static final int STATUS_OK = 0;

    /**
     * File read status: Error decoding file (may be partially decoded)
     */
    public static final int STATUS_FORMAT_ERROR = 1;

    /**
     * File read status: Unable to open source.
     */
    public static final int STATUS_OPEN_ERROR = 2;

    protected BufferedInputStream in;

    protected int status;

    protected int width; // full image width

    protected int height; // full image height

    protected boolean gctFlag; // global color table used

    protected int gctSize; // size of global color table

    protected int loopCount = 1; // iterations; 0 = repeat forever

    protected int[] gct; // global color table

    protected int[] lct; // local color table

    protected int[] act; // active color table

    protected int bgIndex; // background color index

    protected int bgColor; // background color
```

```
protected int lastBgColor; // previous bg color
protected int pixelAspect; // pixel aspect ratio
protected boolean lctFlag; // local color table flag
protected boolean interlace; // interlace flag
protected int lctSize; // local color table size
protected int ix, iy, iw, ih; // current image rectangle
protected Rectangle lastRect; // last image rect
protected BufferedImage image; // current frame
protected BufferedImage lastImage; // previous frame
protected byte[] block = new byte[256]; // current data block
protected int blockSize = 0; // block size
// last graphic control extension info
protected int dispose = 0;

// 0=no action; 1=leave in place; 2=restore to bg; 3=restore to prev
protected int lastDispose = 0;

protected boolean transparency = false; // use transparent color
protected int delay = 0; // delay in milliseconds
protected int transIndex; // transparent color index
protected static final int MaxStackSize = 4096;
// max decoder pixel stack size
// LZW decoder working arrays
protected short[] prefix;
protected byte[] suffix;
protected byte[] pixelStack;
protected byte[] pixels;
protected ArrayList frames; // frames read from current file
protected int frameCount;

static class GifFrame {
    public GifFrame(BufferedImage im, int del) {
        image = im;
    }
}
```

```
        delay = del;
    }

    public BufferedImage image;

    public int delay;
}

/**
 * Gets display duration for specified frame.
 *
 * @param n
 *         int index of frame
 * @return delay in milliseconds
 */
public int getDelay(int n) {
    //
    delay = -1;
    if ((n >= 0) && (n < frameCount)) {
        delay = ((GifFrame) frames.get(n)).delay;
    }
    return delay;
}

/**
 * Gets the number of frames read from file.
 *
 * @return frame count
 */
public int getFrameCount() {
    return frameCount;
}

/**
 * Gets the first (or only) image read.
 *
 * @return BufferedImage containing first frame, or null if none.
 */
public BufferedImage getImage() {
    return getFrame(0);
}

/**
 * Gets the "Netscape" iteration count, if any. A count of 0 means repeat
 * indefinitely.
 *
 * @return iteration count if one was specified, else 1.
 */
public int getLoopCount() {
    return loopCount;
}

/**
 * Creates new frame image from current data (and previous frames as
 * specified by their disposition codes).

```

```

/*
protected void setPixels() {
    // expose destination image's pixels as int array
    int[] dest = ((DataBufferInt) image.getRaster().getDataBuffer())
        .getData();

    // fill in starting image contents based on last image's dispose code
    if (lastDispose > 0) {
        if (lastDispose == 3) {
            // use image before last
            int n = frameCount - 2;
            if (n > 0) {
                lastImage = getFrame(n - 1);
            } else {
                lastImage = null;
            }
        }

        if (lastImage != null) {
            int[] prev = ((DataBufferInt) lastImage.getRaster()
                .getDataBuffer()).getData();
            System.arraycopy(prev, 0, dest, 0, width * height);
            // copy pixels

            if (lastDispose == 2) {
                // fill last image rect area with background color
                Graphics2D g = image.createGraphics();
                Color c = null;
                if (transparency) {
                    c = new Color(0, 0, 0, 0); // assume
background is
                    // transparent
                } else {
                    c = new Color(lastBgColor); // use given
background
                }
                // color
            }
            g.setColor(c);
            g.setComposite(AlphaComposite.Src);
            g.fillRect(lastRect);
            g.dispose();
        }
    }

    // copy each source line to the appropriate place in the destination
    int pass = 1;
    int inc = 8;
    int iline = 0;
    for (int i = 0; i < ih; i++) {
        int line = i;
        if (interlace) {
            if (iline >= ih) {

```

```

        pass++;
        switch (pass) {
        case 2:
            iline = 4;
            break;
        case 3:
            iline = 2;
            inc = 4;
            break;
        case 4:
            iline = 1;
            inc = 2;
        }
    }
    line = iline;
    iline += inc;
}
line += iy;
if (line < height) {
    int k = line * width;
    int dx = k + ix; // start of line in dest
    int dlim = dx + iw; // end of dest line
    if ((k + width) < dlim) {
        dlim = k + width; // past dest edge
    }
    int sx = i * iw; // start of line in source
    while (dx < dlim) {
        // map color and insert in destination
        int index = ((int) pixels[sx++]) & 0xff;
        int c = act[index];
        if (c != 0) {
            dest[dx] = c;
        }
        dx++;
    }
}
}

/**
 * Gets the image contents of frame n.
 *
 * @return BufferedImage representation of frame, or null if n is invalid.
 */
public BufferedImage getFrame(int n) {
    BufferedImage im = null;
    if ((n >= 0) && (n < frameCount)) {
        im = ((GifFrame) frames.get(n)).image;
    }
    return im;
}

/**
 * Gets image size.
*

```

```

        * @return GIF image dimensions
    */
public Dimension getFrameSize() {
    return new Dimension(width, height);
}

/**
 * Reads GIF image from stream
 *
 * @param BufferedInputStream
 *          containing GIF file.
 * @return read status code (0 = no errors)
 */
public int read(BufferedInputStream is) {
    init();
    if (is != null) {
        in = is;
        readHeader();
        if (!err()) {
            readContents();
            if (frameCount < 0) {
                status = STATUS_FORMAT_ERROR;
            }
        }
    } else {
        status = STATUS_OPEN_ERROR;
    }
    try {
        is.close();
    } catch (IOException e) {
    }
    return status;
}

/**
 * Reads GIF image from stream
 *
 * @param InputStream
 *          containing GIF file.
 * @return read status code (0 = no errors)
 */
public int read(InputStream is) {
    init();
    if (is != null) {
        if (!(is instanceof BufferedInputStream))
            is = new BufferedInputStream(is);
        in = (BufferedInputStream) is;
        readHeader();
        if (!err()) {
            readContents();
            if (frameCount < 0) {
                status = STATUS_FORMAT_ERROR;
            }
        }
    } else {
}

```

```

        status = STATUS_OPEN_ERROR;
    }
    try {
        is.close();
    } catch (IOException e) {
    }
    return status;
}

/**
 * Reads GIF file from specified file/URL source (URL assumed if name
 * contains ":" or "file:")
 *
 * @param name
 *      String containing source
 * @return read status code (0 = no errors)
 */
public int read(String name) {
    status = STATUS_OK;
    try {
        name = name.trim().toLowerCase();
        if ((name.indexOf("file:") >= 0) || (name.indexOf(":/") > 0)) {
            URL url = new URL(name);
            in = new BufferedInputStream(url.openStream());
        } else {
            in = new BufferedInputStream(new FileInputStream(name));
        }
        status = read(in);
    } catch (IOException e) {
        status = STATUS_OPEN_ERROR;
    }

    return status;
}

/**
 * Decodes LZW image data into pixel array. Adapted from John Cristy's
 * ImageMagick.
 */
protected void decodeImageData() {
    int NullCode = -1;
    int npix = iw * ih;
    int available, clear, code_mask, code_size, end_of_information, in_code,
old_code, bits, code, count, i, datum, data_size, first, top, bi, pi;

    if ((pixels == null) || (pixels.length < npix)) {
        pixels = new byte[npix]; // allocate new pixel array
    }
    if (prefix == null)
        prefix = new short[MaxStackSize];
    if (suffix == null)
        suffix = new byte[MaxStackSize];
    if (pixelStack == null)
        pixelStack = new byte[MaxStackSize + 1];
}

```

```

// Initialize GIF data stream decoder.

data_size = read();
clear = 1 << data_size;
end_of_information = clear + 1;
available = clear + 2;
old_code = NullCode;
code_size = data_size + 1;
code_mask = (1 << code_size) - 1;
for (code = 0; code < clear; code++) {
    prefix[code] = 0;
    suffix[code] = (byte) code;
}

// Decode GIF pixel stream.

datum = bits = count = first = top = pi = bi = 0;

for (i = 0; i < npix;) {
    if (top == 0) {
        if (bits < code_size) {
            // Load bytes until there are enough bits for a
code.
            if (count == 0) {
                // Read a new data block.
                count = readBlock();
                if (count <= 0)
                    break;
                bi = 0;
            }
            datum += (((int) block[bi]) & 0xff) << bits;
            bits += 8;
            bi++;
            count--;
            continue;
        }
        // Get the next code.

        code = datum & code_mask;
        datum >>= code_size;
        bits -= code_size;

        // Interpret the code

        if ((code > available) || (code == end_of_information))
            break;
        if (code == clear) {
            // Reset decoder.
            code_size = data_size + 1;
            code_mask = (1 << code_size) - 1;
            available = clear + 2;
            old_code = NullCode;
            continue;
        }
    }
}

```

```

        if (old_code == NullCode) {
            pixelStack[top++] = suffix[code];
            old_code = code;
            first = code;
            continue;
        }
        in_code = code;
        if (code == available) {
            pixelStack[top++] = (byte) first;
            code = old_code;
        }
        while (code > clear) {
            pixelStack[top++] = suffix[code];
            code = prefix[code];
        }
        first = ((int) suffix[code]) & 0xff;

        // Add a new string to the string table,
        if (available >= MaxStackSize)
            break;
        pixelStack[top++] = (byte) first;
        prefix[available] = (short) old_code;
        suffix[available] = (byte) first;
        available++;
        if (((available & code_mask) == 0)
                && (available < MaxStackSize)) {
            code_size++;
            code_mask += available;
        }
        old_code = in_code;
    }

    // Pop a pixel off the pixel stack.

    top--;
    pixels[pi++] = pixelStack[top];
    i++;
}
for (i = pi; i < npix; i++) {
    pixels[i] = 0; // clear missing pixels
}

/**
 * Returns true if an error was encountered during reading/decoding
 */
protected boolean err() {
    return status != STATUS_OK;
}

/**
 * Initializes or re-initializes reader

```

```

/*
protected void init() {
    status = STATUS_OK;
    frameCount = 0;
    frames = new ArrayList();
    gct = null;
    lct = null;
}

/**
 * Reads a single byte from the input stream.
 */
protected int read() {
    int curByte = 0;
    try {
        curByte = in.read();
    } catch (IOException e) {
        status = STATUS_FORMAT_ERROR;
    }
    return curByte;
}

/**
 * Reads next variable length block from input.
 *
 * @return number of bytes stored in "buffer"
 */
protected int readBlock() {
    blockSize = read();
    int n = 0;
    if (blockSize > 0) {
        try {
            int count = 0;
            while (n < blockSize) {
                count = in.read(block, n, blockSize - n);
                if (count == -1)
                    break;
                n += count;
            }
        } catch (IOException e) {
        }

        if (n < blockSize) {
            status = STATUS_FORMAT_ERROR;
        }
    }
    return n;
}

/**
 * Reads color table as 256 RGB integer values
 *
 * @param ncolors
 *          int number of colors to read
 * @return int array containing 256 colors (packed ARGB with full alpha)
*/

```

```

/*
protected int[] readColorTable(int ncolors) {
    int nbytes = 3 * ncolors;
    int[] tab = null;
    byte[] c = new byte[nbytes];
    int n = 0;
    try {
        n = in.read(c);
    } catch (IOException e) {
    }
    if (n < nbytes) {
        status = STATUS_FORMAT_ERROR;
    } else {
        tab = new int[256]; // max size to avoid bounds checks
        int i = 0;
        int j = 0;
        while (i < ncolors) {
            int r = ((int) c[j++]) & 0xff;
            int g = ((int) c[j++]) & 0xff;
            int b = ((int) c[j++]) & 0xff;
            tab[i++] = 0xff000000 | (r << 16) | (g << 8) | b;
        }
    }
    return tab;
}

/**
 * Main file parser. Reads GIF content blocks.
 */
protected void readContents() {
    // read GIF file content blocks
    boolean done = false;
    while (!(done || err())) {
        int code = read();
        switch (code) {

            case 0x2C: // image separator
                readImage();
                break;

            case 0x21: // extension
                code = read();
                switch (code) {
                    case 0xf9: // graphics control extension
                        readGraphicControlExt();
                        break;

                    case 0xff: // application extension
                        readBlock();
                        String app = "";
                        for (int i = 0; i < 11; i++) {
                            app += (char) block[i];
                        }
                        if (app.equals("NETSCAPE2.0")) {
                            readNetscapeExt();
                        }
                }
        }
    }
}

```

```

        } else
            skip(); // don't care
        break;

    default: // uninteresting extension
        skip();
    }
    break;

    case 0x3b: // terminator
    done = true;
    break;

    case 0x00: // bad byte, but keep going and see what happens
    break;

    default:
        status = STATUS_FORMAT_ERROR;
    }
}

/**
 * Reads Graphics Control Extension values
 */
protected void readGraphicControlExt() {
    read(); // block size
    int packed = read(); // packed fields
    dispose = (packed & 0x1c) >> 2; // disposal method
    if (dispose == 0) {
        dispose = 1; // elect to keep old image if discretionary
    }
    transparency = (packed & 1) != 0;
    delay = readShort() * 10; // delay in milliseconds
    transIndex = read(); // transparent color index
    read(); // block terminator
}

/**
 * Reads GIF file header information.
 */
protected void readHeader() {
    String id = "";
    for (int i = 0; i < 6; i++) {
        id += (char) read();
    }
    if (!id.startsWith("GIF")) {
        status = STATUS_FORMAT_ERROR;
        return;
    }

    readLSD();
    if (gctFlag && !err()) {
        gct = readColorTable(gctSize);
        bgColor = gct[bgIndex];
    }
}

```

```

        }

    /**
     * Reads next frame image
     */
    protected void readImage() {
        ix = readShort(); // (sub)image position & size
        iy = readShort();
        iw = readShort();
        ih = readShort();

        int packed = read();
        lctFlag = (packed & 0x80) != 0; // 1 - local color table flag
        interlace = (packed & 0x40) != 0; // 2 - interlace flag
        // 3 - sort flag
        // 4-5 - reserved
        lctSize = 2 << (packed & 7); // 6-8 - local color table size

        if (lctFlag) {
            lct = readColorTable(lctSize); // read table
            act = lct; // make local table active
        } else {
            act = gct; // make global table active
            if (bgIndex == transIndex)
                bgColor = 0;
        }
        int save = 0;
        if (transparency) {
            save = act[transIndex];
            act[transIndex] = 0; // set transparent color if specified
        }

        if (act == null) {
            status = STATUS_FORMAT_ERROR; // no color table defined
        }

        if (err())
            return;

        decodeImageData(); // decode pixel data
        skip();

        if (err())
            return;

        frameCount++;

        // create new image to receive frame data
        image = new BufferedImage(width, height,
                                  BufferedImage.TYPE_INT_ARGB_PRE);

        setPixels(); // transfer pixel data to image

        frames.add(new GifFrame(image, delay)); // add image to frame list
    }
}

```

```

        if (transparency) {
            act[transIndex] = save;
        }
        resetFrame();
    }

    /**
     * Reads Logical Screen Descriptor
     */
    protected void readLSD() {

        // logical screen size
        width = readShort();
        height = readShort();

        // packed fields
        int packed = read();
        gctFlag = (packed & 0x80) != 0; // 1 : global color table flag
        // 2-4 : color resolution
        // 5 : gct sort flag
        gctSize = 2 << (packed & 7); // 6-8 : gct size

        bgIndex = read(); // background color index
        pixelAspect = read(); // pixel aspect ratio
    }

    /**
     * Reads Netscape extenstion to obtain iteration count
     */
    protected void readNetscapeExt() {
        do {
            readBlock();
            if (block[0] == 1) {
                // loop count sub-block
                int b1 = ((int) block[1]) & 0xff;
                int b2 = ((int) block[2]) & 0xff;
                loopCount = (b2 << 8) | b1;
            }
        } while ((blockSize > 0) && !err());
    }

    /**
     * Reads next 16-bit value, LSB first
     */
    protected int readShort() {
        // read 16-bit value, LSB first
        return read() | (read() << 8);
    }

    /**
     * Resets frame state for reading next image.
     */
    protected void resetFrame() {

```

```

        lastDispose = dispose;
        lastRect = new Rectangle(ix, iy, iw, ih);
        lastImage = image;
        lastBgColor = bgColor;
        int dispose = 0;
        boolean transparency = false;
        int delay = 0;
        lct = null;
    }

    /**
     * Skips variable length blocks up to and including next zero length block.
     */
    protected void skip() {
        do {
            readBlock();
        } while ((blockSize > 0) && !err());
    }
}

```

The details on how the program is beyond the scope of these notes but in the spirit of always looking under the covers we go over the class and how it works in a short document available at brainycode.com.

We are interested in using it with just requires that we examine the javadoc example:

```

GifDecoder d = new GifDecoder();
d.read("sample.gif");
int n = d.getFrameCount();
for (int i = 0; i < n; i++) {
    BufferedImage frame = d.getFrame(i); // frame i
    int t = d.getDelay(i); // display duration of frame in
    milliseconds
    // do something with frame
}

```

4. Create the package com.brainycode.platformer.entities.

This package is where we will place all classes related to our game entities.

5. Create the java class AnimatedGIF.java²⁰

Table 24 - AnimatedGIF.java

```

package com.brainycode.platformer.entities;

import java.awt.Graphics2D;
import java.awt.Point;
import java.awt.image.BufferedImage;

```

²⁰ This class will evolve as we learn to process spritesheets and combine with this class.

```

import com.brainycode.external.imageprocessing.GifDecoder;

public class AnimatedGIF {

    private Point position;           // the position of gif on screen
    private BufferedImage[] frames;   // the internal frames

    private GifDecoder d;

    private int numFrames = -1;
    private int currentFrame = -1;

    private boolean isReady = false;

    public AnimatedGIF(int x, int y) {
        position = new Point();
        position.x = x;
        position.y = y;
    }

    public void read(String animatedGIFFFile) {
        d = new GifDecoder();
        d.read(getClass().getResourceAsStream(animatedGIFFFile));
        numFrames = d.getFrameCount();
        currentFrame = 0;
        frames = new BufferedImage[numFrames];
        for (int i=0; i < numFrames; i++) {
            frames[i] = d.getFrame(i);
        }
        isReady = true;
    }

    public void draw(Graphics2D g) {
        if (isReady) {
            g.drawImage(frames[currentFrame], position.x, position.y, null);
        }
    }

    public void nextFrame() {
        ++currentFrame;
        if (currentFrame >= numFrames) {
            currentFrame = 0;
        }
    }
}

```

The class AnimatedGIF will be used to read in and process/display the frames within an animated gif. The user will created an AnimatedGIF object and provide the initial position on the screen.

6. Update the GamePanel.java for this project to the following:

Table 25 - GamePanel.java for the dancing banana

```

package com.brainycode.platformer.main;

import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;

import javax.swing.JPanel;

import com.brainycode.platformer.entities.AnimatedGIF;

public class GamePanel extends JPanel implements Runnable{

    private static final long serialVersionUID = 1L;

    // dimensions
    public static final int WIDTH = 640;
    public static final int HEIGHT = 480;

    private AnimatedGIF dancingBanana;

    private Thread thread;
    private boolean running;

    private int FPS = 10;
    private long targetTime = 1000 / FPS;

    public GamePanel() {
        super();
        setPreferredSize ( new Dimension(WIDTH, HEIGHT));
        setFocusable(true);
        requestFocus();
    }

    private void init() {
        dancingBanana = new AnimatedGIF(160, 100);
        dancingBanana.read("/images/dancing-banana.gif");
        running = true;
    }

    public void addNotify() {
        super.addNotify();
        // Create game engine thead here
        if (thread == null) {
            thread = new Thread(this);
            thread.start();
        }
    }

    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D) g;
        dancingBanana.draw(g2);
    }
}

```

```
}
```



```
@Override
public void run() {
    init();
    long start;
    long elapsed;
    long wait = 100;

    while (running) {
        start = System.nanoTime();

        elapsed = System.nanoTime() - start;
        wait = targetTime - elapsed / 1000000;
        if (wait < 0 ) wait = 5;
        try {
            Thread.sleep(wait);
        } catch (Exception e) {
            e.printStackTrace();
        }
        dancingBanana.nextFrame();

        repaint();
    }
}
```

7. Create the resources directory
8. Under the resources directory create an images directory
9. Place the image dancing-banana.gif into resources/images
10. Make sure the resources directory is in the classpath

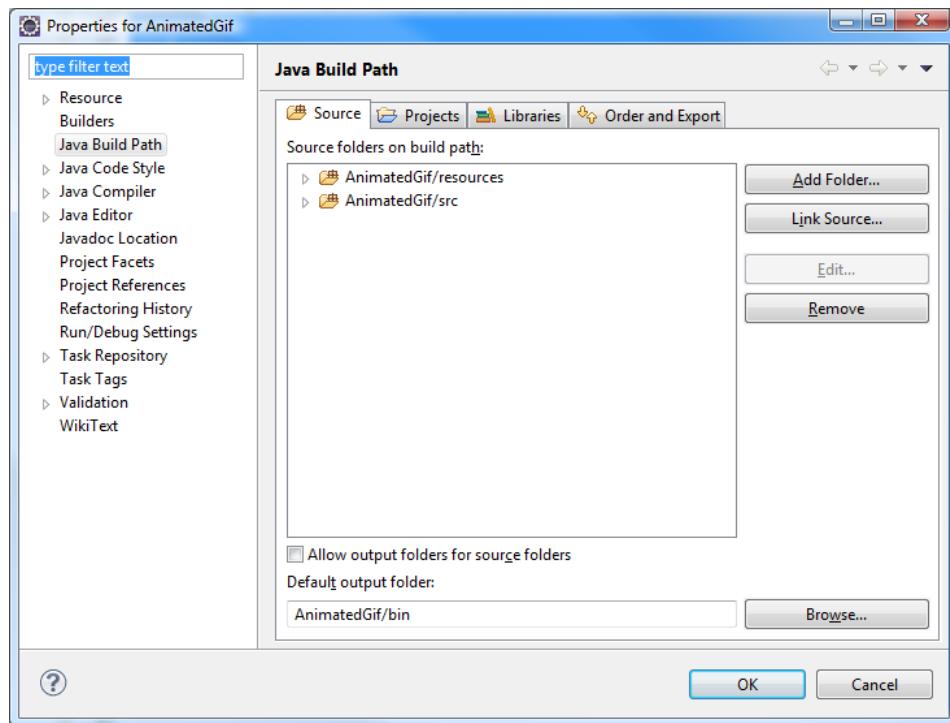


Figure 91 - Adding resources to the class path

11. Run the Game.

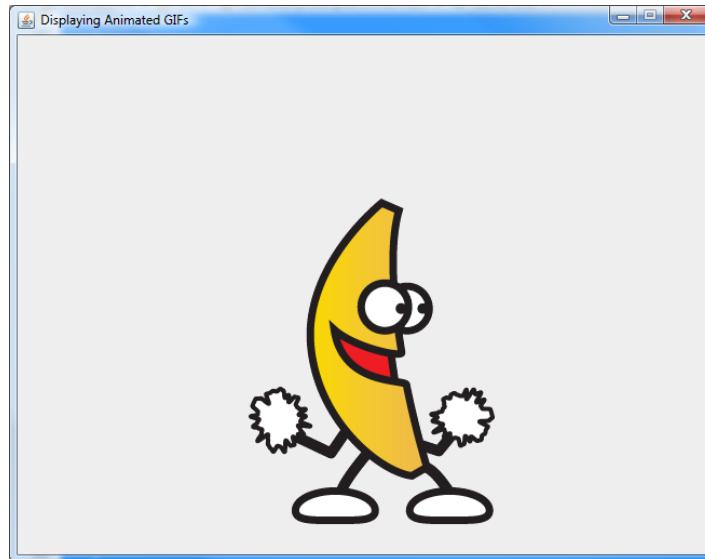


Figure 92 - Our "Dancing Banana!"

Keyboard Processing

[Creating Pong](#)

[Creating Breakout](#)

[Creating MindSweeper](#)

March 30, 2014

[BUILD A 2D PLATFORMER - JAVA]

Bibliography

Boese, Elizabeth Sugar. Learn to Program with Java Applet Game Examples, 2d Edition. Self-published.

Davison, Andrew. Killer Game Programming in Java. 1005 Gravenstein Highway North, Sebastopol, CA 95472: O'Reilly Media, 2005.

Holzner, Steve. Ant The Definitive Guide, 2nd Edition. O'Reilly Media, Inc. 2005.

Kent, Steven L. The Ultimate History of Video Games. Three Rivers Press. 2010.

Knudsen, Jonathan. Java 2D Graphics. O'Reilly. 1999

Melissinos, Chris. The Art of Video Games: From Pac-Man to Mass Effect. Welcome Books, First Edition. March 5, 2012.

Appendix A: Web References

1. <http://www.brainycode.com> – This is the author's website. As this book develops – versions are placed online for review. I also updated any images or files you may want to use in order to complete the labs and exercises.
2. <http://www.gamedev.net/> - A great place for game development discussions and tutorials.
3. <http://www.romnation.net> – A place to obtain tons of ROMs to play classic games on emulators
4. http://en.wikipedia.org/wiki/Abstract_Window_Toolkit
5. <http://home.cogeco.ca/~ve3ll/jatutorg.htm>
6. <http://docs.oracle.com/javase/1.4.2/docs/api/java.awt/Canvas.html>
7. http://en.wikibooks.org/wiki/Java_Programming/Canvas
8. <http://docs.oracle.com/javase/tutorial/uiswing/components/frame.html>
9. <http://www.javablogging.com/what-is-serialversionuid/>
10. <http://stackoverflow.com/questions/285793/what-is-a-serialversionuid-and-why-should-i-use-it>
11. <http://ant.apache.org> – for information on Ant
12. <http://junit.org/> - for information on JUnit. A great framework for creating test cases.

Appendix B – Using Eclipse

Eclipse Tutorial.

These notes will not be long enough to do justice the breadth and depth of using Eclipse in your program development.

Appendix C – Using JUnit (test-driven development)

JUnit Tutorial

The purpose of this tutorial is to walk you through the steps one would use when developing any Java program while trying to make use of JUnit. JUnit is a testing framework used to unit test as much of your program²¹ as possible. The key difference between JUnit and informal²² of testing your code is that the testcases you develop are easy to run, repeatable, and does not require any special scaffolding to implement or execute. If you installed the recommended version of Eclipse it will have JUnit already installed and available for your use. The general rules are the following:

Directory Setup

When you create the default Java Project in Eclipse you will have a src directory that will be the place you create and manage your java class files. By default Eclipse will place the generated *.class files in a /bin directory using the same packaging in under the src directory. You will also create a new source directory named /test. If you have a java class file under /src by the name com.brainycode.game.physics.CollisionDetection then you will have a class under /test with the name com.brainycode.game.physics.TestCollisionDetection.

Your test cases will be independent of each other and organized under /test so you can run and execute them all when you build your project.

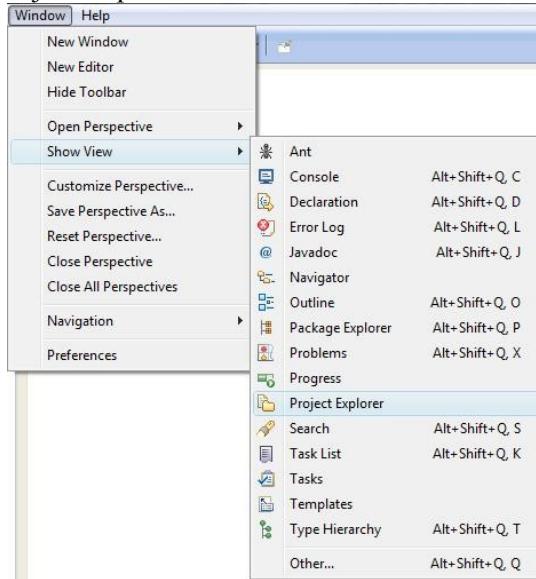
²¹ Ideally you want to test all your code and that should be our goal.

²² The usual way may have been to create a main method for a class and have the developer directly run it.

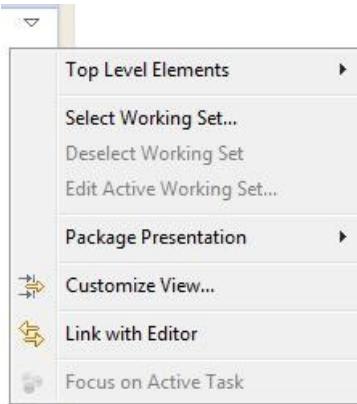
Where are the class files?

The typical Java or J2EE perspective displays the Package Explorer view pane on the left. This view pane does not display the class files typically stored in the \bin directory. In order to see your class files you will need to do the following:

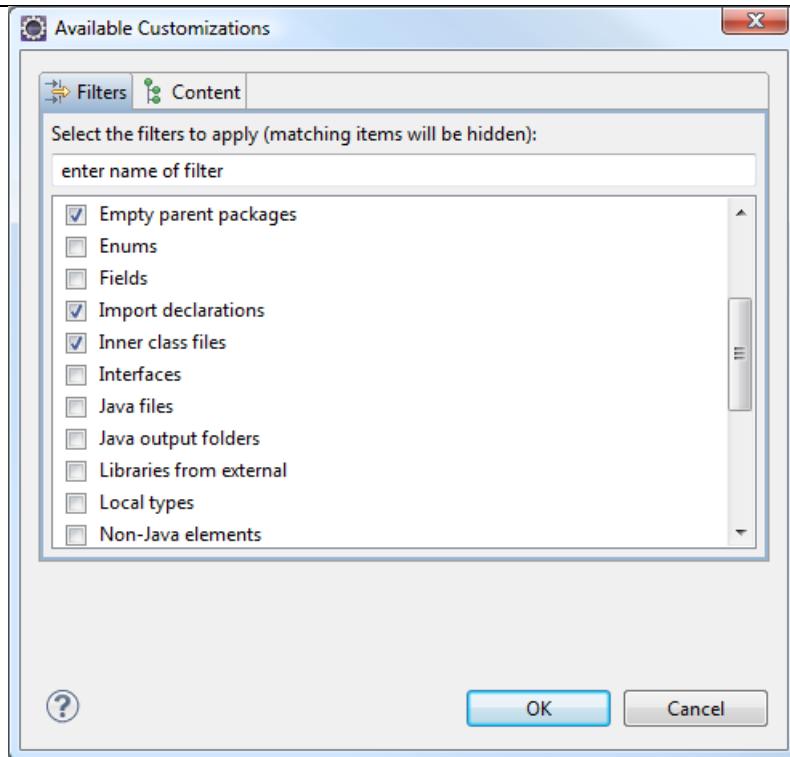
1. Window -> View -> Project Explorer



2. In the Project Explorer view click on the down arrow and select "Customize View..."



3. Unclick "Java output folders"



4. Click on “OK”

You will now see the bin directory in the Project Explorer.

We will use the latest version of JUnit which allows us to use *annotations* to identify and manage our testcases.

What is an Annotation?

Annotations, a form of metadata, provide data about a program that is not part of the program itself. Annotations have no direct effect on the operation of the code they annotate.

Example:

`@Test`

See: <http://docs.oracle.com/javase/tutorial/java/annotations/> for more information if you are new to the concept.

Imports

I normally recommend letting Eclipse automatically insert the import statements as you build your test cases. The key imports you will see are the following:

- org.junit.Test – This is an annotation (@Test) that tells JUnit that the method which it is associated with can be run as a test case.
- org.junit.Ignore – The annotation @Ignore is used to temporarily disable a test or group of tests

- org.junit.Assert – This class contains of assert methods that you will use to testing the success of your test case

Creating a test case

You will typically use the Eclipse New → Java → JUnit Testcase to create a class that will serve as your test case.

In this exercise will demonstrate how you would go about building the test case before actually creating the classes. This is to demonstrate the spirit of test driven development.

STEP 1: Download the latest version of JUnit²³. I usually unzip under C:\ as C:\JUnit4.9.

STEP 2: Create a new Eclipse workspace called JAVA_TEST_PROJECTS (I usually create under my accounts workspace).

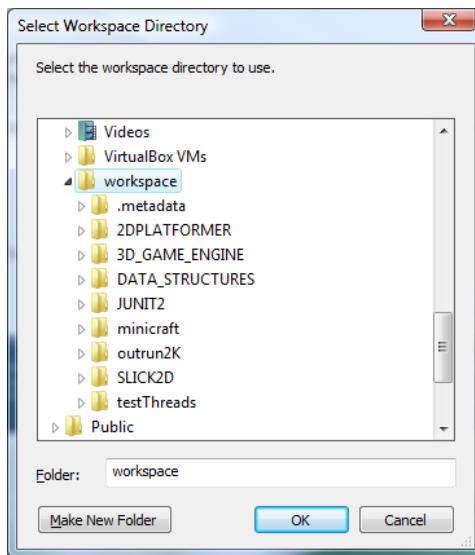


Figure 93 - workspace directory

STEP 3: Click on “Make New Folder” button and enter “JAVA_TEST_PROJECTS”

²³ You can skip this part if you downloaded and installed the Eclipse package that includes it.

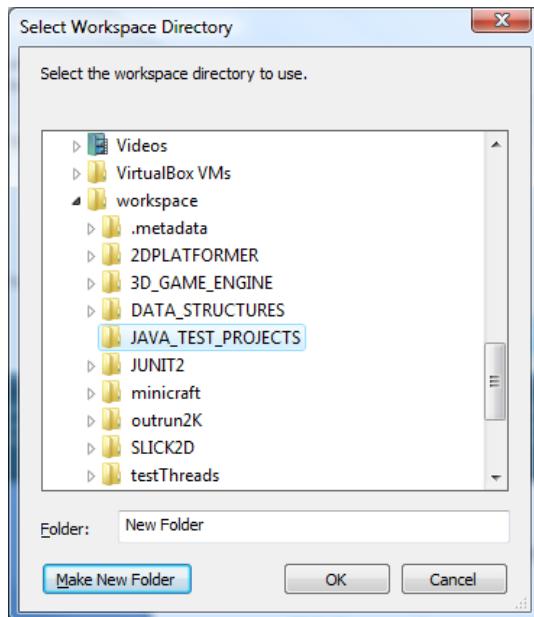


Figure 94 - Selecting the new workspace directory

STEP 4: Highlight the new directory workspace and click on “OK”.

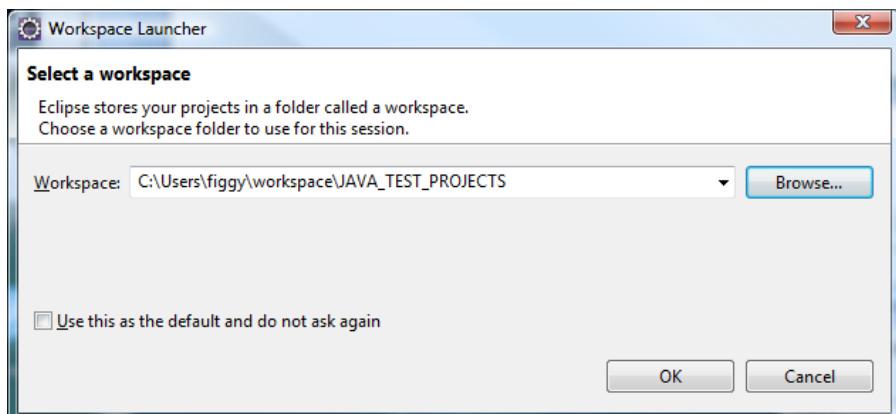


Figure 95 - Workspace Launcher

STEP 5: Click on “OK” to launch the new workspace.

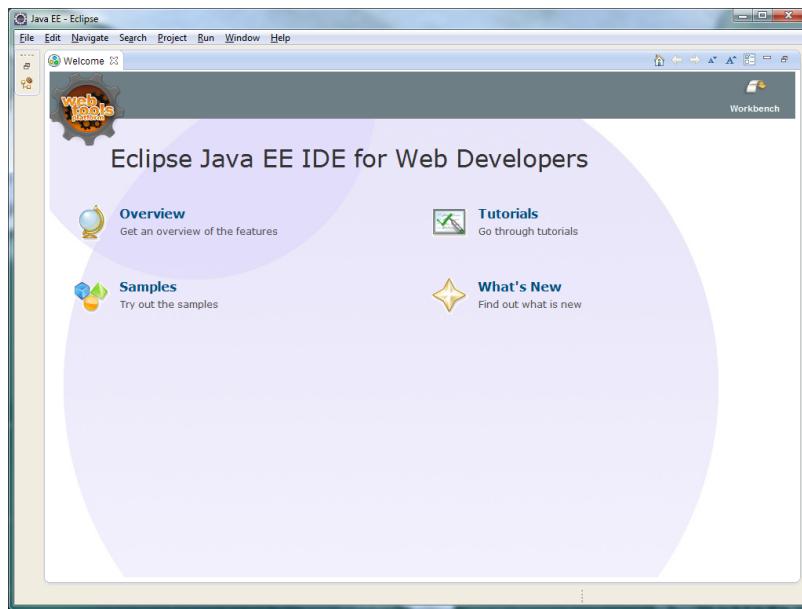


Figure 96 - Workspace "Welcome" screen

STEP 6: Enter the Workbench to start building Eclipse projects.

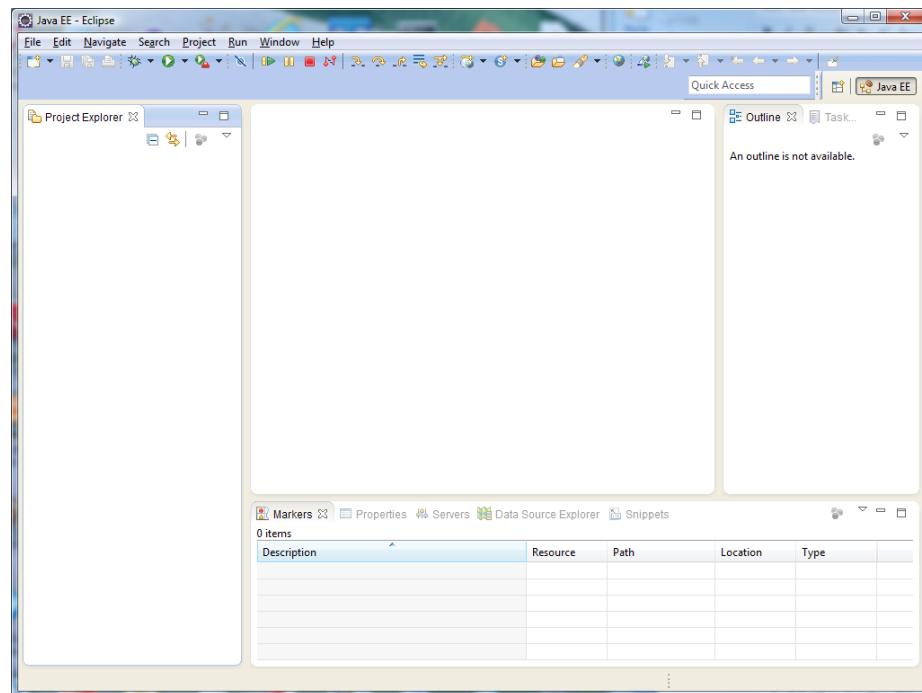


Figure 97 - An empty workbench

STEP 7: Right-click and create a new Java Project

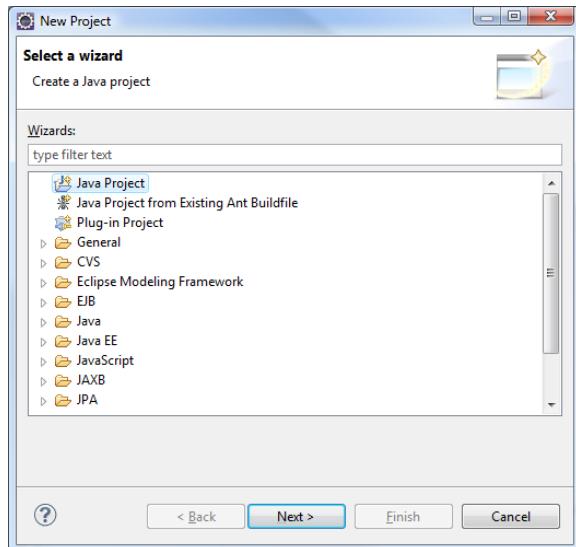


Figure 98 - Creating a Java Project

STEP 8: Click Next >

STEP 9: Enter Project name: TEST_JUNIT

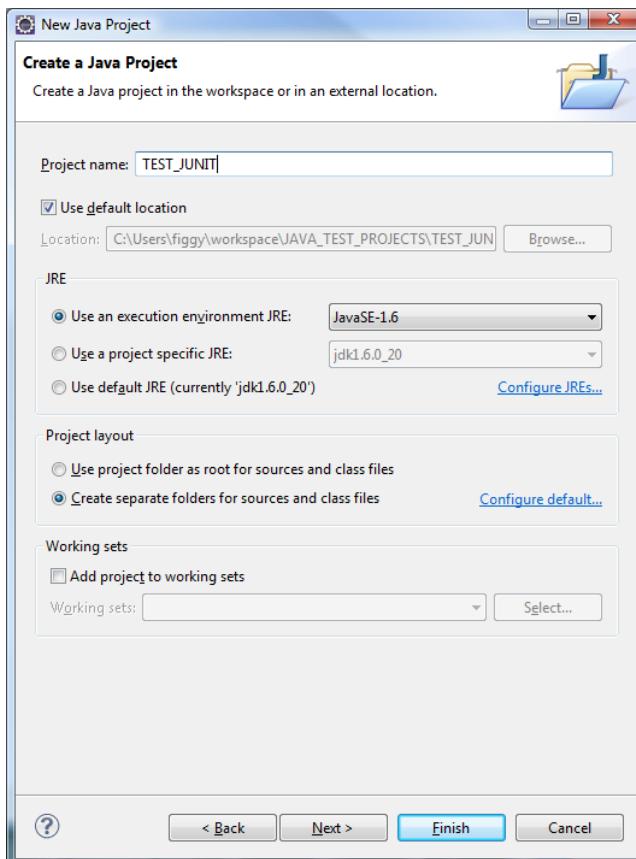


Figure 99 - Creating a Java Project

STEP 10: Click “Finish”



Figure 100 - Open Associated Perspective?

STEP 11: If the “Open Associated Perspective?” dialog box opens up. Click on the “Remember my decision” check box and select “Yes” button. This will open the views that naturally go with the Java perspective.

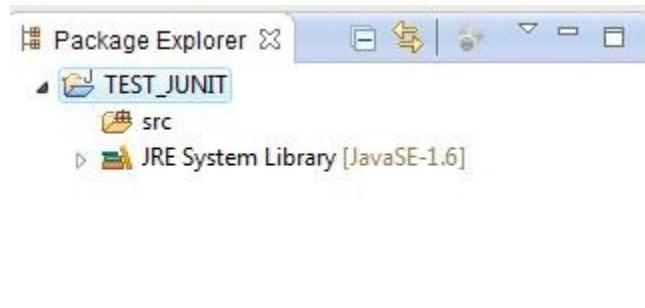


Figure 101 - TEST_JUNIT Package Explorer

The goal of this exercise is to test how to use JUnit. We are going to create two classes one representing a ball moving on a 640 by 480 screen and the other a vector that specifies the magnitude (how fast) and direction the ball moves. The ball will have two key properties that we will make use of in this exercise its current position on the screen and a vector variable representing the direction and speed in which the ball is moving. We will not be actually showing a ball moving in this small test. We want to illustrate how we can test concepts represented by methods in our classes and test them out independent of viewing on a GUI or having graphical representations of an object. We also want to illustrate how to perform test-driven development. Where we create the test case first and work to make it pass. The first thing to conceptualize is a “virtual screen.”

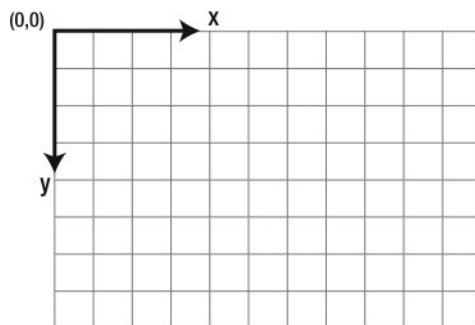


Figure 102 - Our virtual screen

We will place a ball object on our virtual screen and move it about until it hits a screen edge and change its direction so it effectively “bounces off the edges.” The top-left position is screen coordinate (0,0) as an object moves right on the screen the x-coordinate increases, as an object moves down the y coordinate increases.

So if our screen is 640 across and 480 down, that is 640×480 , the screen coordinates goes from 0..639 (left to right) to 0..479 (up and down). Let’s reiterate this again. The x-coordinate can range from 0..639 and the y-coordinate value can range from 0..479. The top-left location is (0,0) the top-right location is (639, 0). The bottom-right coordinate is (0, 479) and the right-bottom coordinate is (639, 479).

STEP 12: Highlight the project as shown above and right-click and create a new source folder titled test.

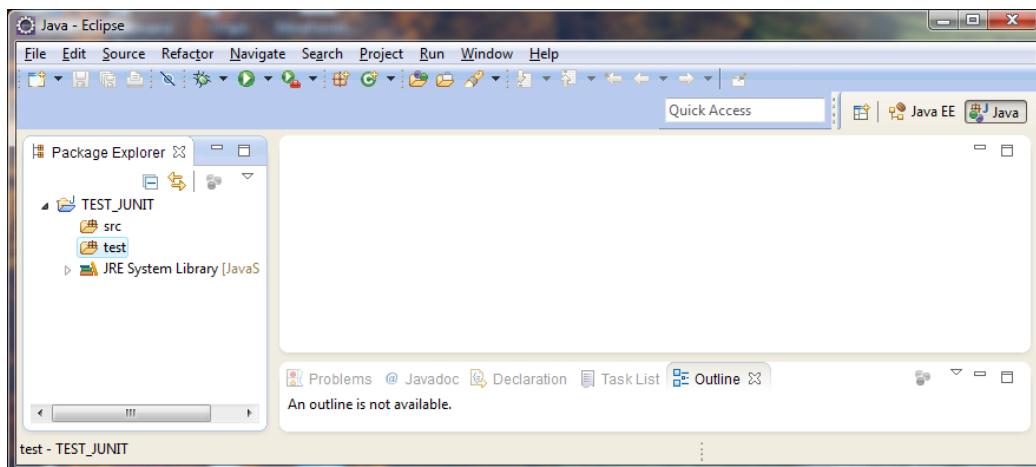


Figure 103 - Adding test folder

STEP 13: Highlight the test folder and add the new package com.brainycode.example.JUnit.

STEP 14: Highlight the new package and add the new “JUnit Test Case” TestVector2D.

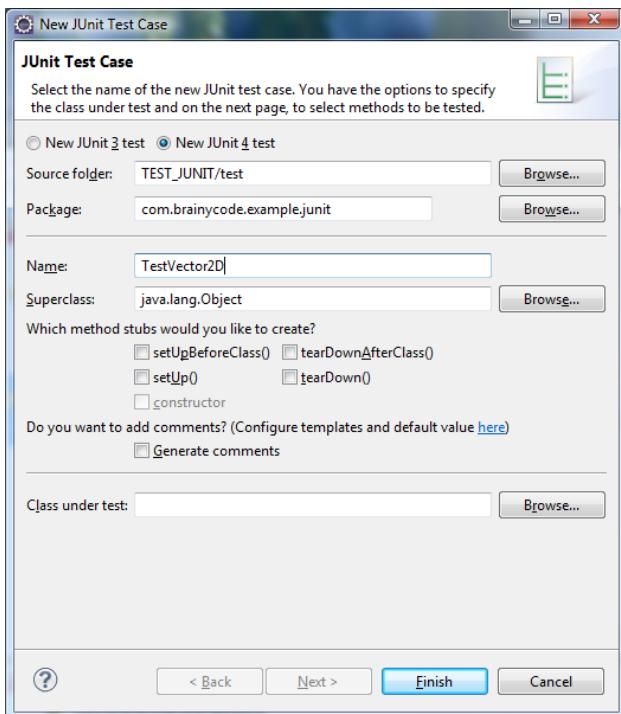


Figure 104 - Creating JUnit testcase TestVector2D

STEP 15: Click Finish. If you are prompted to add JUnit4 library to the build path then click “OK”

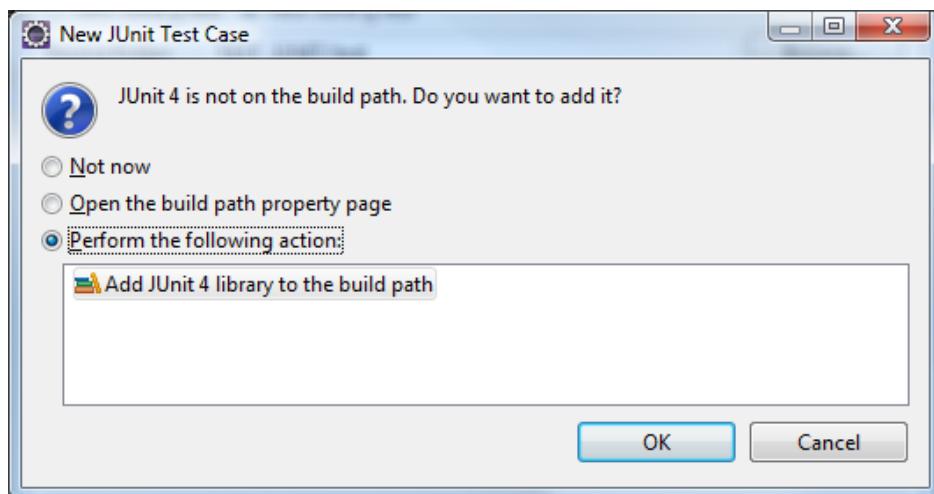


Figure 105 - Dialog prompt to add JUnit 4 library to the build path

Table 26 - Default testcase code

```
package com.brainycode.example.JUnit;

import static org.junit.Assert.*;
import org.junit.Test;

public class TestVector2D {
```

```

@Test
public void test() {
    fail("Not yet implemented");
}
}

```

The @Test is the annotation to inform the test module that the method following it is to be run as a test case.

If you right-click and Run As → JUnit Test you will see it fail because of the fail(<message>) statement.

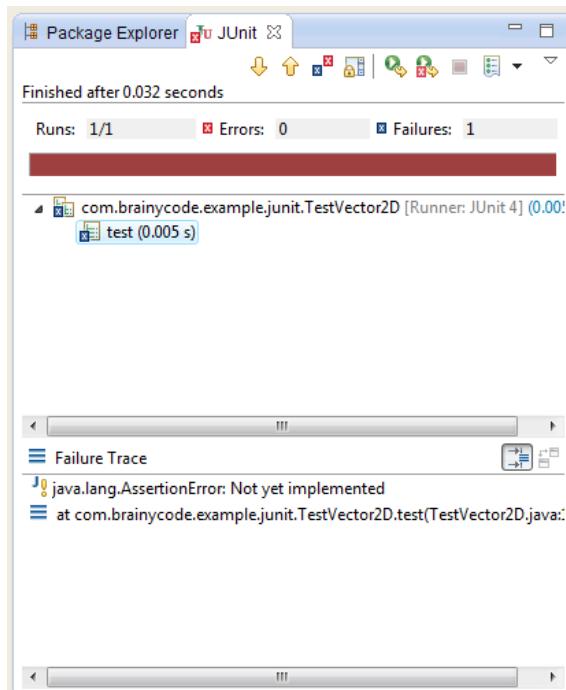


Figure 106 - Failed unimplemented testcase

STEP 16: Add the following code to the test() method:

```

@Test
public void testConstructor() {
    Vector2D displacement = new Vector2D(2, 3);
    assertTrue(displacement.getX() == 2);
    assertTrue(displacement.getY() == 3);
}

```

Eclipse will flag Vector2D as not existing.

The intention of the test case is create a ball at location (100, 100) and then use the displacement vector of (2,3) to move or update the ball. This will basically perform an addition on respective components so that after an update the ball should be at location (102, 103).

STEP 17: To add the code for the Vector2D right-click on the Vector2D and select Quick Fix. Select the option to **create the class**. Update the “Source folder” to TEST_JUNIT/src.

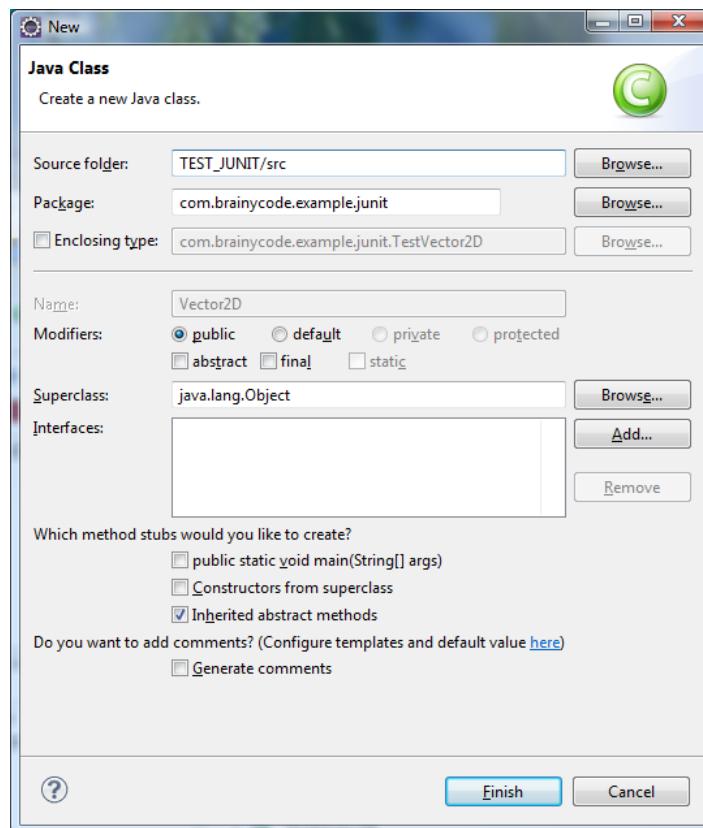


Figure 107 - Creating a class from code!

STEP 18: Eclipse will create the following default class:

```
package com.brainycode.example.JUnit;

public class Vector2D {
```

STEP 19: Return back to your TestVector2D class. The “[new Vector2D\(2,3\);](#)” is flagged. Right-click and select the option to create the Vector2D constructor as shown below:

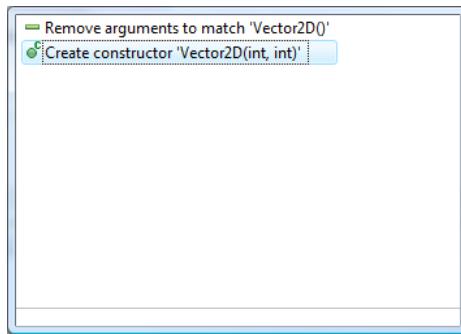


Figure 108 - Creating Vector2D constructor

STEP 20: Fill in the Vector2D class as shown (I only filled in the two lines in the constructor and let the Quick Fix selections and the Eclipse generate getters and setters do the rest.

Table 27 - Vector2D class for JUnit test

```
package com.brainycode.example.JUnit;

public class Vector2D {

    private int x;
    private int y;

    public Vector2D(int i, int j) {
        x = i;
        y = j;
    }

    public int getX() {
        return x;
    }

    public void setX(int x) {
        this.x = x;
    }

    public int getY() {
        return y;
    }

    public void setY(int y) {
        this.y = y;
    }
}
```

STEP 21: Run the testcase again.

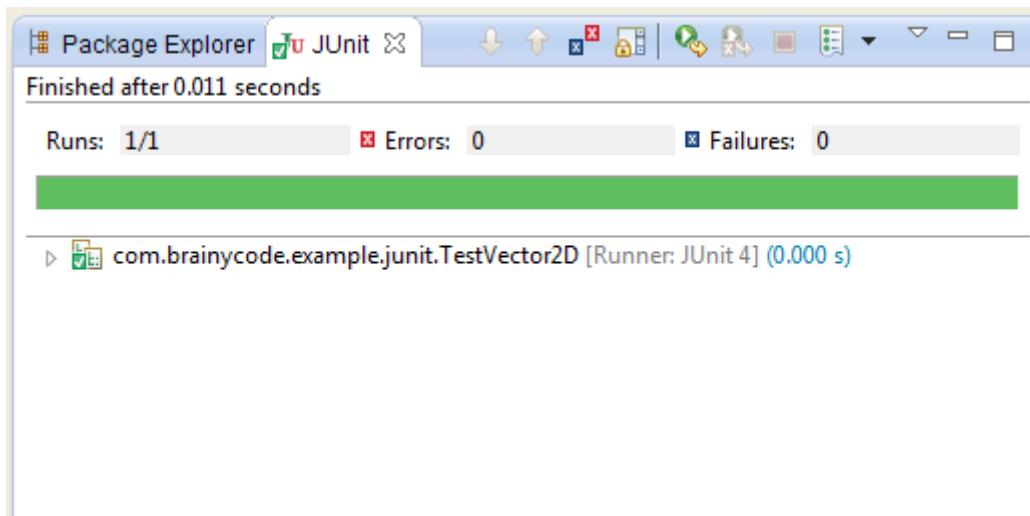


Figure 109 - Successful test case

The key statements in our testcase are:

```
assertTrue(displacement.getX() == 2);
assertTrue(displacement.getY() == 3);
```

We send to the assertTrue method a conditional that should be true if our testcase passes. In this case, since we are creating a displacement object we are testing that the x and y field values are being initialized correctly.

JUnit has many different variations of the assert statement.

Table 28 - Variations of the assert method

method name / parameters	description
<code>assertTrue (test)</code> <code>assertTrue ("message", test)</code>	Causes this test method to fail if the given boolean test is not true.
<code>assertFalse (test)</code> <code>assertFalse ("message", test)</code>	Causes this test method to fail if the given boolean test is not false.
<code>assertEquals (expectedValue, value)</code> <code>assertEquals ("message", expectedValue, value)</code>	Causes this test method to fail if the given two values are not equal to each other. (For objects, it uses <code>thequals</code> method to compare them.) The first of the two values is considered to be the result that you expect; the second is the actual result produced by the class under test.
<code>assertNotEquals (value1, value2)</code> <code>assertNotEquals ("message", value1, value2)</code>	Causes this test method to fail if the given two values are equal to each other. (For objects, it uses <code>thequals</code> method to compare them.)
<code>assertNull (value)</code>	Causes this test method to fail if the given value is

<code>assertNull("message", value)</code>	not null.
<code>assertNotNull(value)</code> <code>assertNotNull("message", value)</code>	Causes this test method to fail if the given value <i>is</i> null.
<code>assertSame(expectedValue, value)</code> <code>assertSame("message", expectedValue, value)</code>) <code>assertNotSame(value1, value2)</code> <code>assertNotSame("message", value1, value2)</code>	Identical to <code>assertEquals</code> and <code>assertNotEquals</code> respectively, except that for objects, it uses the <code>==</code> -operator rather than the <code>equals</code> method to compare them. (The difference is that two objects that have the same state might be <code>equals</code> to each other, but not <code>==</code> to each other. An object is only <code>==</code> to itself.)
<code>fail()</code> <code>fail("message")</code>	Causes this test method to fail.

STEP 22: Add a new test case:

```
@Test
public void testAdd() {
    Vector2D displacement = new Vector2D(2, 3);

    Vector2D vector2 = new Vector2D(-1, -1);
    displacement.add(vector2);
    assertTrue(displacement.getX() == 1);
    assertTrue(displacement.getY() == 2);
}
```

Eclipse will mark the `displacement.add` method as missing. So add new method using “Quick Fix” to the `Vector2D` class.

```
public void add(Vector2D vector2) {
    x = x + vector2.getX();
    y = y + vector2.getY();

}
```

Run all the testcases. They should all pass.

STEP 23: Add the last testcase for multiplication and add the method to the `Vector2D` class as shown in previous steps.

```
@Test
public void testMultiply() {
    Vector2D displacement = new Vector2D(2, 3);

    displacement.multiply(-1);
    assertTrue(displacement.getX() == -2);
    assertTrue(displacement.getY() == -3);
```

```
}
```

New Vector2D method:

```
public void multiply(int i) {
    x *= i;
    y *= i;
}
```

Note how the value provided is multiplied against each vector component.

Run all test cases:

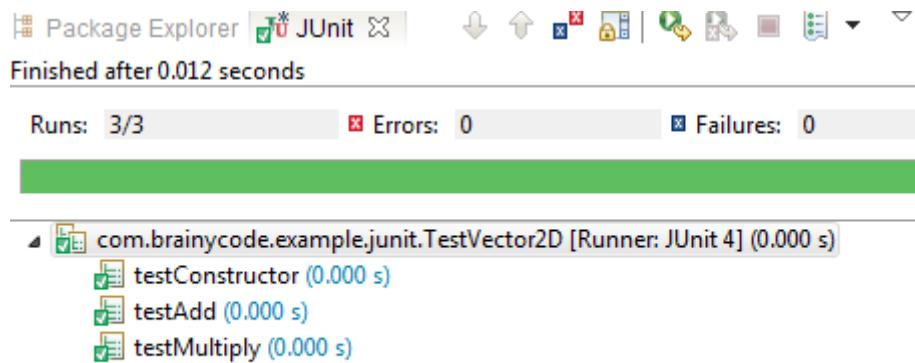


Figure 110 - Running all testcases

STEP 24: Create a TestBall java class in the test directory.

Table 29 - Initial TestBall class

```
package com.brainycode.example.JUnit;

import static org.junit.Assert.*;
import org.junit.Test;

public class TestBall {

    @Test
    public void testConstructor() {
        Ball ball = new Ball(100,100, 50);
        assertTrue(ball.getX() == 100);
        assertTrue(ball.getY() == 100);
    }
}
```

Create the class TestBall but use a Point object to save the x,y top-left location the ball and radius to save the ball's radius.

When we think of an object that represents a circle (our ball) we think of two key properties: the center point and the radius. This is quite natural from learning basic math. We would like to represent our ball using these same properties but the reality is that when drawing a ball object on the screen the key properties is the top-left position of the rectangle that would be enclosing the circle and the width and height of that rectangle²⁴.

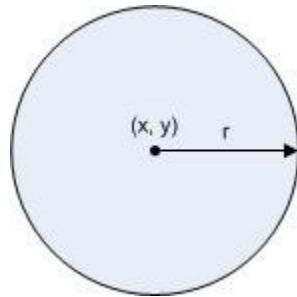


Figure 111 - Our familiar notion of properties of a ball

The Java graphics tools require that we describe the circle or ball we want to build as shown below:

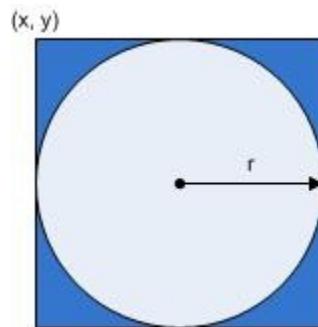


Figure 112 - Using two other properties to specify a ball

We specify a circle by NOT specifying its center point and radius but by specifying the top-left point of the enclosing rectangle and the width and height which in our case for the circle is $2 * r$ for both.

We mention this point because in the construction of our Ball object the Point object we create will be the top-left position NOT the center point.

²⁴ Note, that with the top-left position and width and height you can determine the radius and the center point position, this is just an alternative way to specify the circle or in fact any oval shape.

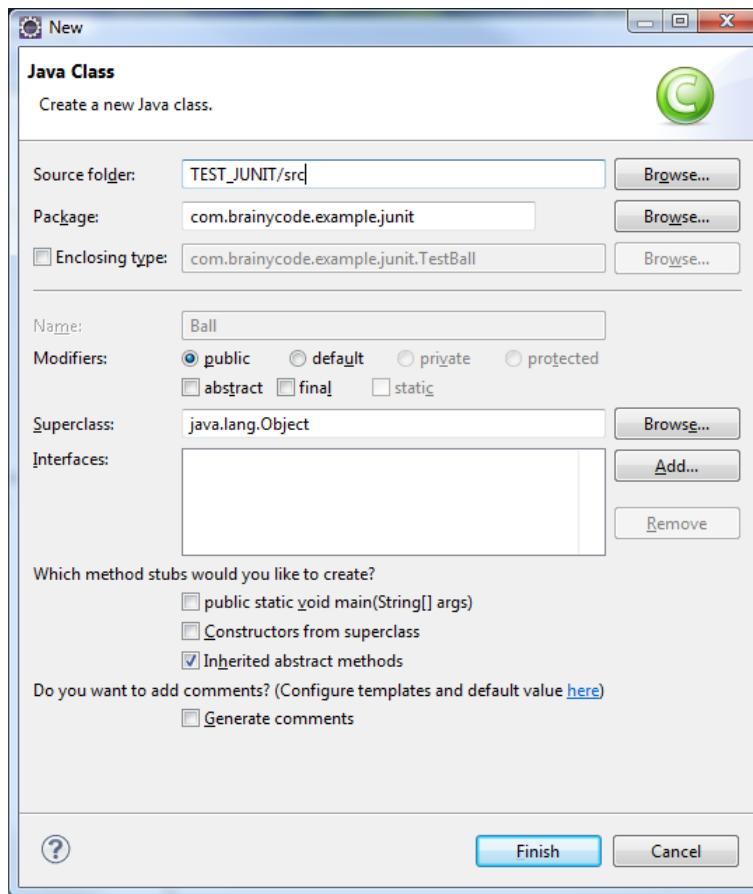


Figure 113 - Creating the ball class

STEP 25: Create the Ball class constructor automatically using Eclipse and make the code look like example below:

Table 30 - JUnit test Ball.java

```
package com.brainycode.example.JUnit;

import java.awt.Point;

import com.brainycode.example.JUnit.Vector2D;

public class Ball {

    private Point point;           // the top-left position of the ball on the
screen
    private int radius;            // the radius of the ball
    private Vector2D displacement; // how fast/direction the ball is moving

    public Ball(int x, int y, int r) {
        point = new Point();
        point.x = x;
        point.y = y;
        radius = r;
    }
}
```

```

}

public int getX() {
    return point.x;
}

public int getY() {
    return point.y;
}

public int getRadius() {
    return radius;
}

public void setRadius(int radius) {
    this.radius = radius;
}

public Vector2D getDisplacement() {
    return displacement;
}

public void setDisplacement(Vector2D displacement) {
    this.displacement = displacement;
}

public void update() {
    point.x += displacement.getX();
    point.y += displacement.getY();
}

}

```

STEP 26: Add a new test to move the ball on the screen:

```

@Test
public void moveBall() {
    Ball ball = new Ball(100,100, 50);
    Vector2D displacement = new Vector2D(2,3);
    ball.setDisplacement(displacement);
    ball.update();
    assertTrue(ball.getX() == 102);
    assertTrue(ball.getY() == 103);
}

```

Fix any flagged errors by adding the update() method to the Ball class.

```

public void update() {
    centerPoint.x += displacement.getX();

```

```

        centerPoint.y += displacement.getY();
    }
}

```

[Figure 114 - Test case failing](#)

Run all the testcases in TestBall – they should all pass.

Let's pretend now that the ball is on a 640 by 480 screen. We never want the ball to go off the screen. In our next case we are going to place the ball at the location (438, 100) and have a displacement of (2, 3). The ball is moving in the positive x and y direction (that means to the right and down). When we update the ball a piece of the ball will be off the screen since the updated location for the top-left position of the ball will be (440, 103).

$$\text{New x position} = \text{Old x position} + \text{displacement.x} = 438 + 2 = 440$$

$$\text{New y position} = \text{Old y position} + \text{displacement.y} = 100 + 3 = 103$$

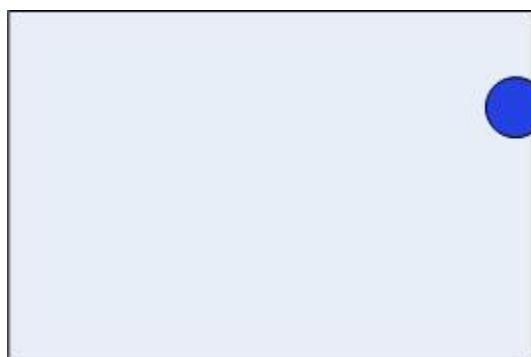
How do we know the ball is now off the screen?

When we calculate the rectangle enclosing the ball the x-most pixel position is

$$\text{Right most x pixel position} = \text{new x position} + \text{ball diameter} = 440 + 2 * \text{radius} = 440 + 100 = 640.$$

$$\text{Bottom most y pixel position} = \text{new y position} + \text{ball diameter} = 103 + 2 * \text{radius} = 103 + 100 = 203.$$

Since the x range is only from 0..639 we can conclude that a piece of the ball has passed the wall.



[Figure 115 - Ball passing the wall](#)

Two things must be done to give the illusion that the ball is going to bounce off the wall. The ball has to be placed back on the playing screen and the vector displacement must be adjusted. To give the illusion of bouncing the ball should be placed flush against the wall or at SCREEN_WIDTH – (2 * RADIUS) – 1 in order for the right-most pixel to be flush against right side of the screen.



Figure 116 - Ball bouncing

In addition to the ball being flush against the right side the displacement vector needs to be adjusted so the ball continues in the opposite direction. We know that that displacement is $(+x, +y)$ that is some positive x and some positive y value (that is why it is moving in the downward diagonal direction). The simplest way to adjust the displacement vector is the reverse the sign of the x direction $\rightarrow (-x, +y)$. We will see in Chapter X how this is actually changing the ball 90 degrees from impact to the wall.

STEP 27: Add a new test case to test the ball moving to the right and bouncing off the right wall.

Table 31 - testcase for bouncing off right wall

```
@Test
public void moveBallOffRightScreen() {
    Ball ball = new Ball(589,100, 50);
    Vector2D displacement = new Vector2D(2,3);
    ball.setDisplacement(displacement);
    ball.update();
    assertTrue(ball.getX() == Screen.WIDTH - (2 * ball.getRadius()) -1);
    assertTrue(ball.getY() == 103);

    // check if the ball has reversed direction
    assertTrue(ball.getDisplacement().getX() < 0);
}
```

Add a new Screen class that will merely hold the dimensions of our virtual screen:

Table 32 - Screen.java

```
package com.brainycode.example.JUnit;

public class Screen {

    public static final int WIDTH = 640;
    public static final int HEIGHT = 480;
}
```

If you run the test case it will fail.

Note: You can run a specific test case by moving the cursor in the Eclipse editing window over the test case method and clicking on the Run icon.

What's the problem?

We need to make the Ball's update() method smarter about detecting and adjusting the ball when it detects it went off to screen.

The code to add:

```
public void update() {
    // Move the ball
    point.x += displacement.getX();
    point.y += displacement.getY();

    // see if we went off the right side
    if (getX() > Screen.WIDTH - 2 * radius - 1) {
        setX(Screen.WIDTH - 2 * radius - 1);
        displacement.setX(-displacement.getX());
    }
}
```

In the update() method we adjust the top-left position of the ball and then test if it went off to the right. We know it goes off if the rectangle width is off the screen. Screen.WIDTH - 2 * radius - 1 represents the buffer zone around the screen where the ball cannot enter otherwise we will draw some of the ball offscreen. So if the x value is greater than Screen.WIDTH - 2 * radius - 1 we need to reset x for the ball so it is flush with the screen. In addition, we reverse the sign of the x displacement component so that the ball will start moving AWAY from the right edge.

The above will require that you add two new setter methods to the Ball class:

Table 33 - Ball's additional setter methods

<pre><code>public void setX(int newX) { point.x = newX; } public void setY(int newY) { point.y = newY; }</code></pre>
--

If you re-run the testcase – it will now pass!

As you can see in order to set up a test case we painfully figure out what we should get back from a working program – fail – make changes to the code to make the testcase pass.

This is test-driven development!

Let's add a testcase for the ball moving and hitting the left wall.

Table 34 - Testing the ball moving left and down

```

@Test
public void moveBallOffLeftScreen() {
    Ball ball = new Ball(1,100, 50);
    // set ball to move left and down
    Vector2D displacement = new Vector2D(-2,3);
    ball.setDisplacement(displacement);
    ball.update();
    assertTrue(ball.getX() == 0);
    assertTrue(ball.getY() == 103);

    // check if the ball has reversed direction
    assertTrue(ball.getDisplacement().getX() > 0);
}

```

At this point you should convince yourself that setting the ball at position (1,100) and moving left and down by (-2, 3) will make the ball bounce off the left-side of the screen.

The testcase moveBallOffLeftScreen() will now fail. To get it to pass add the following code to the Ball.update() method:

Table 35 - Additional code for Ball.update()

```

// see if we went off the left side
if ( getX() < 0) {
    // bring the ball back to the screen and reverse course
    setX(0);
    // change direction in X
    displacement.setX(-displacement.getX());
}

```

The code above checks if the ball went off the left-side by checking the x value and sets it to be flush against the left-side of the screen and reverses the vector displacement so it now will start moving to the right.

The testcase should now pass.

EXERCISE 1-3: Add testcases for moveBallOffTopScreen() and moveBallOffBottomScreen to adjust the y component of the ball's top-left position on the screen. Also, add code the Ball.update() in order to make to make it work. In addition, add the code necessary to adjust the y-component of the ball in the Ball.update() method.In the next chapter we will learn enough about drawing to create and move a ball we can see.

There are many online tutorials on how to use JUnit. I highly recommend the website

Appendix D – Ant

Ant Tutorial

Appendix E – Setting up on a Linux Machine

Appendix F – Setting up on a Macintosh Machine

Appendix G – MAME

MAME stands for Multiple Arcade Machine Emulator. It was built to provide a way for fans of old archade game games such as Donkey Kong, Space Invaders, Galaga and more can play those great games on their computers.

Obtain and Install MAME

To get started you will need to head over to mamdev.org download page and download. The file will be an executable file that will unzip the files on your hard drive. I recommend using C:\MAME. When you unzip the files you will see the following directory layout:

Name	Date modified	Type	Size
artwork	1/24/2014 9:53 PM	File Folder	
cfg	1/24/2014 9:57 PM	File Folder	
ctrlr	1/24/2014 9:53 PM	File Folder	
docs	1/24/2014 9:53 PM	File Folder	
hash	1/24/2014 9:53 PM	File Folder	
hlsl	1/24/2014 9:53 PM	File Folder	
roms	1/24/2014 9:53 PM	File Folder	
samples	1/24/2014 9:53 PM	File Folder	
web	1/24/2014 9:53 PM	File Folder	
chdman	12/24/2013 3:01 AM	Application	1,119 KB
jedutil	12/24/2013 3:01 AM	Application	207 KB
Idresample	12/24/2013 3:01 AM	Application	1,003 KB
Idverify	12/24/2013 3:01 AM	Application	1,035 KB
ledutil	12/24/2013 3:01 AM	Application	121 KB
mame	12/24/2013 3:07 AM	Application	79,538 KB
mame.sym	12/24/2013 3:02 AM	SYM File	29,438 KB
romcmp	12/24/2013 3:01 AM	Application	189 KB
unidasm	12/24/2013 3:07 AM	Application	2,690 KB
whatsnew	12/24/2013 2:25 AM	Text Document	22 KB

Figure 117 - MAME Directory structure

I recommend creating a desktop shortcut to start the Mame application. But, before you start . . .

Obtain Game ROMS

The best place to obtain game roms is on <http://www.romnation.net>. When you search for a particular game rom you may find more than one version available. In addition, you should know that legally you must own a copy of the ROM (maybe like me you actually have an arcade cabinet or two in your possession). Now, to be honest, I think most people do not own copies of the games they intend to play on their computer systems.

In certain cases you will need to upload several machine specific ROMS (from the circuit board) in order to play a game. MAME is very good at informing you on what is missing and the Internet search engine is

useful for locating the missing files. I will only caution that many website offering free ROMS can be places to obtain computer viruses and Trojans so thread lightly and be cautious.

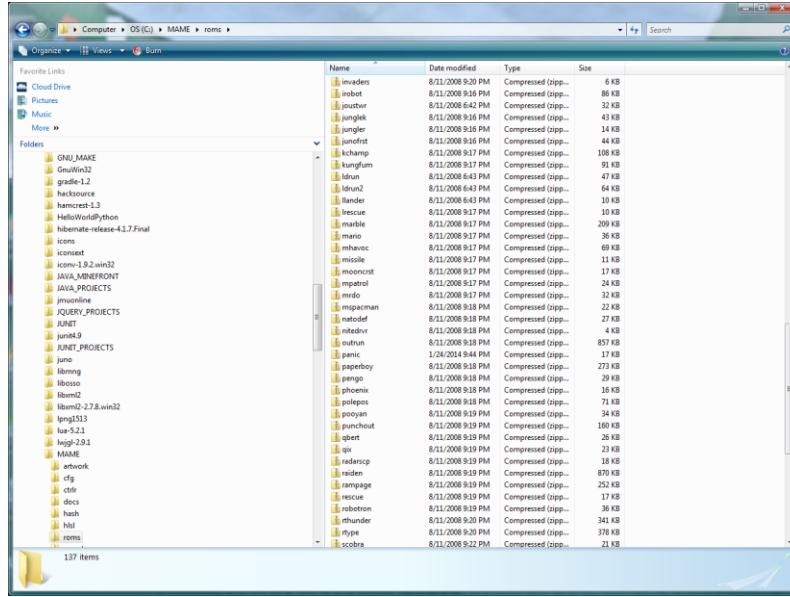


Figure 118 - My ROM directory

The ROMs you download from the Internet should be places (zipped) in the MAME/roms directory.

Configure your system

When you start MAME it starts up in full-screen mode use ALT+ENTER to toggle between full-screen and Windows mode.

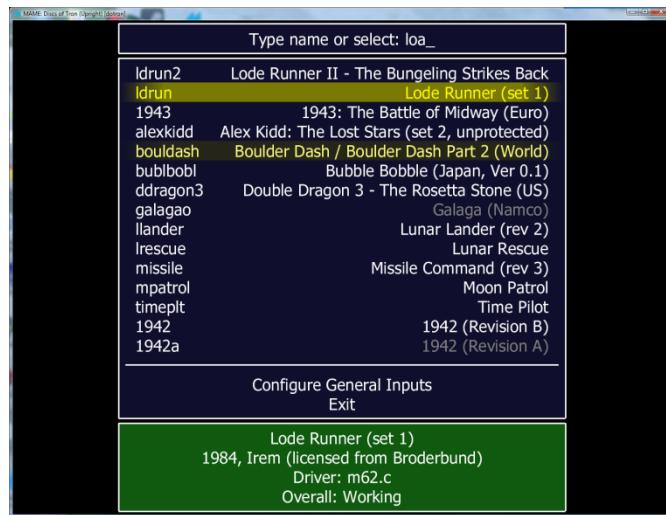


Figure 119 - MAME running in windows mode

There are alternative windows ui's available for MAME (e.g. MAMEUI)

Keyboard Commands to start and control a game

When you start a game remember the following controls:

[TAB] – brings up the configuration menu

5 – Drops a quarter into the slot

1 – Is the “Start” button

[LEFT-CTRL] [LEFT-ALT] [SPACEBAR] are your action buttons.

Use the arrow keys to move your avatar/player around on the screen.

Start a game and start playing and enjoy!!

Appendix H – Using Ant

In these notes we have explained some of the Ant tasks that may come up when you creating a project. Here is a sample list of obtained from the Ant web site:

Archive Tasks

Task Name	Description
BUnzip2	Expands a file packed using GZip or BZip2.
BZip2	Packs a file using the GZip or BZip2 algorithm. This task does not do any dependency checking; the output file is always generated
Cab	Creates Microsoft CAB archive files. It is invoked similar to the Jar or Zip tasks. This task will work on Windows using the external <i>cabarc</i> tool (provided by Microsoft), which must be located in your executable path.
Ear	An extension of the Jar task with special treatment for files that should end up in an Enterprise Application archive.
GUnzip	Expands a GZip file.
GZip	GZips a set of files.
Jar	Jars a set of files.
Jlink	<i>Deprecated.</i> Use the <code>zipfileset</code> and <code>zipgroupfileset</code> attributes of the Jar or Zip tasks instead.
Manifest	Creates a manifest file.
Rpm	Invokes the <i>rpm</i> executable to build a Linux installation file. This task currently only works on Linux or other Unix platforms with RPM support.
SignJar	Signs a jar or zip file with the <i>javasign</i> command-line tool.
Tar	Creates a tar archive.
Unjar	Unzips a jarfile.

Untar	Untars a tarfile.
Unwar	Unzips a warfile.
Unzip	Unzips a zipfile.
War	An extension of the Jar task with special treatment for files that should end up in the WEB-INF/lib, WEB-INF/classes, or WEB-INF directories of the Web Application Archive.
Zip	Creates a zipfile.

Audit/Coverage Tasks

Task Name	Description
JDepend	Invokes the JDepend parser. This parser "traverses a set of Java source-file directories and generates design-quality metrics for each Java package".

Compile Tasks

Task Name	Description
Depend	Determines which classfiles are out-of-date with respect to their source, removing the classfiles of any other classes that depend on the out-of-date classes, forcing the re-compile of the removed classfiles. Typically used in conjunction with the Javac task.
Javac	Compiles the specified source file(s) within the running (Ant) VM, or in another VM if the <code>fork</code> attribute is specified.
Apt	Runs the annotation processor tool (apt), and then optionally compiles the original code, and any generated source code.
JspC	Runs the JSP compiler. It can be used to precompile JSP pages for fast initial invocation of JSP pages, deployment on a server without the full JDK installed, or simply to syntax-check the pages without deploying them. The Javac task can be used to compile the generated Java source. (For Weblogic JSP compiles, see the WLjspc task.)
NetRexxC	Compiles a NetRexx source tree within the running (Ant) VM.

Rmic	Runs the <i>rmic</i> compiler on the specified file(s).
Wljspc	Compiles JSP pages using Weblogic's JSP compiler, <i>weblogic.jspc</i> . (For non-Weblogic JSP compiles, see the JspC task.)

Deployment Tasks

Task Name	Description
ServerDeploy	Task to run a "hot" deployment tool for vendor-specific J2EE server.

Documentation Tasks

Task Name	Description
Javadoc/Javadoc2	Generates code documentation using the <i>javadoc</i> tool. The Javadoc2 task is deprecated; use the Javadoc task instead.

EJB Tasks

Task Name	Description
EJB Tasks	(See the documentation describing the EJB tasks.)

Execution Tasks

Task Name	Description
Ant	Runs Ant on a supplied buildfile, optionally passing properties (with possibly new values). This task can be used to build sub-projects.
AntCall	Runs another target within the same buildfile, optionally passing properties (with possibly new values).
Apply/ExecOn	Executes a system command. When the <i>os</i> attribute is specified, the command is only executed when Ant is run on one of the specified operating systems.
Dependset	This task compares a set of source files with a set of target files. If any of the source files is newer than any of the target files, all the target files are removed.
Exec	Executes a system command. When the <i>os</i> attribute is specified, the command is

	only executed when Ant is run on one of the specified operating systems.
Java	Executes a Java class within the running (Ant) VM, or in another VM if the <code>fork</code> attribute is specified.
Parallel	A container task that can contain other Ant tasks. Each nested task specified within the <code><parallel></code> tag will be executed in its own thread.
Sequential	A container task that can contain other Ant tasks. The nested tasks are simply executed in sequence. Its primary use is to support the sequential execution of a subset of tasks within the <code><parallel></code> tag.
Sleep	A task for suspending execution for a specified period of time. Useful when a build or deployment process requires an interval between tasks.
Subant	Calls a given target for all defined sub-builds. This is an extension of ant for bulk project execution.
Waitfor	Blocks execution until a set of specified conditions become true. This task is intended to be used with the Parallel task to synchronize a set of processes.

File Tasks

Task Name	Description
Attrib	Changes the permissions and/or attributes of a file or all files inside the specified directories. Currently, it has effect only under Windows.
Checksum	Generates a checksum for a file or set of files. This task can also be used to perform checksum verifications.
Chgrp	Changes the group ownership of a file or all files inside the specified directories. Currently, it has effect only under Unix.
Chmod	Changes the permissions of a file or all files inside the specified directories. Currently, it has effect only under Unix. The permissions are also UNIX style, like the arguments for the <code>chmod</code> command.
Chown	Changes the owner of a file or all files inside the specified directories. Currently, it has effect only under Unix.
Concat	Concatenates multiple files into a single one or to Ant's logging system.

Copy	Copies a file or Fileset to a new file or directory.
Copydir	<i>Deprecated.</i> Use the Copy task instead.
Copyfile	<i>Deprecated.</i> Use the Copy task instead.
Delete	Deletes either a single file, all files and sub-directories in a specified directory, or a set of files specified by one or more FileSets .
Deltree	<i>Deprecated.</i> Use the Delete task instead.
Filter	Sets a token filter for this project, or reads multiple token filters from a specified file and sets these as filters. Token filters are used by all tasks that perform file-copying operations.
FixCRLF	Modifies a file to add or remove tabs, carriage returns, linefeeds, and EOF characters.
Get	Gets a file from a URL.
Mkdir	Creates a directory. Non-existent parent directories are created, when necessary.
Move	Moves a file to a new file or directory, or a set(s) of file(s) to a new directory.
Patch	Applies a "diff" file to originals.
Rename	<i>Deprecated.</i> Use the Move task instead.
RenameExtensions	<i>Deprecated.</i> Use the Move task with a glob mapper instead.
Replace	Replace is a directory-based task for replacing the occurrence of a given string with another string in selected file.
ReplaceRegExp	Directory-based task for replacing the occurrence of a given regular expression with a substitution pattern in a file or set of files.
Sync	Synchronize two directory trees.
Tempfile	Generates a name for a new temporary file and sets the specified property to that name.

Touch	Changes the modification time of a file and possibly creates it at the same time.
-----------------------	---

Java2 Extensions Tasks

Task Name	Description
Jarlib-available	Check whether an extension is present in a FileSet or an ExtensionSet. If the extension is present, the specified property is set.
Jarlib-display	Display the "Optional Package" and "Package Specification" information contained within the specified jars.
Jarlib-manifest	Task to generate a manifest that declares all the dependencies in manifest. The dependencies are determined by looking in the specified path and searching for Extension/"Optional Package" specifications in the manifests of the jars.
Jarlib-resolve	Try to locate a jar to satisfy an extension, and place the location of the jar into the specified property.

Logging Tasks

Task Name	Description
Record	Runs a listener that records the logging output of the build-process events to a file. Several recorders can exist at the same time. Each recorder is associated with a file.

Mail Tasks

Task Name	Description
Mail	A task to send SMTP email.
MimeMail	<i>Deprecated.</i> Use the Mail task instead.

Miscellaneous Tasks

Task Name	Description
Defaultexcludes	Modify the list of default exclude patterns from within your build file.

Echo	Echoes text to <code>System.out</code> or to a file.
Fail	Exits the current build by throwing a <code>BuildException</code> , optionally printing additional information.
GenKey	Generates a key in keystore.
HostInfo	Sets properties related to the provided host, or to the host the process is run on.
Input	Allows user interaction during the build process by displaying a message and reading a line of input from the console.
Script	Executes a script in a Apache BSF -supported language.
Sound	Plays a sound file at the end of the build, according to whether the build failed or succeeded.
Splash	Displays a splash screen.
Sql	Executes a series of SQL statements via JDBC to a database. Statements can either be read in from a text file using the <code>src</code> attribute, or from between the enclosing SQL tags.
Taskdef	Adds a task definition to the current project, such that this new task can be used in the current project.
TStamp	Sets the <code>DSTAMP</code> , <code>TSTAMP</code> , and <code>TODAY</code> properties in the current project, based on the current date and time.
Typedef	Adds a data-type definition to the current project, such that this new type can be used in the current project.
XmlValidate	Checks that XML files are valid (or only well-formed). This task uses the XML parser that is currently used by Ant by default, but any SAX1/2 parser can be specified, if needed.

Pre-process Tasks

Task Name	Description
ANTLR	Invokes the ANTLR Translator generator on a grammar file.

AntStructure	Generates a DTD for Ant buildfiles that contains information about all tasks currently known to Ant.
Import	Import another build file and potentially override targets in it with targets of your own.
Include	Include another build file.
JavaCC	Invokes the JavaCC compiler-compiler on a grammar file.
Javah	Generates JNI headers from a Java class.
JJDoc	Invokes the JJDoc documentation generator for the JavaCC compiler-compiler. JJDoc takes a JavaCC parser specification and produces documentation for the BNF grammar. It can operate in three modes, determined by command line options. This task only invokes JJDoc if the grammar file is newer than the generated BNF grammar documentation.
JJTree	Invokes the JJTree preprocessor for the JavaCC compiler-compiler. It inserts parse-tree building actions at various places in the JavaCC source that it generates. The output of JJTree is run through JavaCC to create the parser. This task only invokes JJTree if the grammar file is newer than the generated JavaCC file.
Macrodef	Define a new task as a macro built-up upon other tasks.
Native2Ascii	Converts files from native encodings to ASCII with escaped Unicode. A common usage is to convert source files maintained in a native operating system encoding to ASCII, prior to compilation.
Presetdef	Define a new task by instrumenting an existing task with default values for attributes or child elements.
Translate	Identifies keys in files, delimited by special tokens, and translates them with values read from resource bundles.
XSLT	Processes a set of documents via XSLT.

Property Tasks

Task Name	Description
Available	Sets a property if a specified file, directory, class in the classpath, or JVM system

	resource is available at runtime.
Basename	Sets a property to the last element of a specified path.
BuildNumber	Task that can be used to track build numbers.
Condition	Sets a property if a certain condition holds true; this is a generalization of Available and Uptodate .
Dirname	Sets a property to the value of the specified file up to, but not including, the last path element.
Echoproperties	Lists the current properties.
LoadFile	Loads a file into a property.
LoadProperties	Load a file's contents as Ant properties. This task is equivalent to using <code><property file="..."/></code> except that it supports nested <code><filterchain></code> elements, and it cannot be specified outside a target.
MakeURL	Creates a URL (list) from a file/fileset or path
PathConvert	Converts a nested path, path reference, filelist reference, or fileset reference to the form usable on a specified platform and/or to a list of items separated by the specified separator and stores the result in the specified property.
Property	Sets a property (by name and value), or set of properties (from a file or resource) in the project.
PropertyFile	Creates or modifies property files. Useful when wanting to make unattended modifications to configuration files for application servers and applications. Typically used for things such as automatically generating a build number and saving it to a build properties file, or doing date manipulation.
Uptodate	Sets a property if a given target file is newer than a set of source files.
Whichresource	Find a class or resource.
XmlProperty	Loads property values from a well-formed XML file.

Remote Tasks

Task Name	Description
FTP	Implements a basic FTP client that can send, receive, list, and delete files, and create directories.
Rexec	Task to automate a remote rexec session.
Scp	Copy files to or from a remote server using SSH.
setproxy	Sets Java's web proxy properties, so that tasks and code run in the same JVM can have through-the-firewall access to remote web sites.
Sshexec	Execute a command on a remote server using SSH.
Telnet	Task to automate a remote <i>telnet</i> session. This task uses nested <read> and <write> tags to indicate strings to wait for and specify text to send.

SCM Tasks

Task Name	Description
Cvs	Handles packages/modules retrieved from a CVS repository.
CvsChangeLog	Generates an XML report of the changes recorded in a CVS repository.
CVSPass	Adds entries to a .cvspass file. Adding entries to this file has the same affect as a <i>cvs login</i> command.
CvsTagDiff	Generates an XML-formatted report file of the changes between two tags or dates recorded in a CVS repository.
ClearCase	Tasks to perform the ClearCase cleartool <i>checkin</i> , <i>checkout</i> , <i>uncheckout</i> , <i>update</i> , <i>lock</i> , <i>unlock</i> , <i>mklbtype</i> , <i>rmtypes</i> , <i>mklabel</i> , <i>mkattr</i> , <i>mkdir</i> , <i>mkelem</i> , and <i>mkbl</i> commands.
Continuus/Synergy	Tasks to perform the Continuus <i>ccmcheckin</i> , <i>ccmcheckout</i> , <i>ccmcheckintask</i> , <i>ccmreconfigure</i> , and <i>ccmcreateTask</i> commands.
Microsoft Visual SourceSafe	Tasks to perform the Visual SourceSafe <i>vssget</i> , <i>vsslabel</i> , <i>vsshistory</i> , <i>vsscheckin</i> , <i>vsscheckout</i> , <i>vssadd</i> , <i>vssc</i>

	<i>p</i> , and <i>vsscreate</i> commands.
Pvcs	Allows the user extract the latest edition of the source code from a PVCS repository.
SourceOffSite	Tasks to perform the SourceOffSite <i>sosget</i> , <i>soslabel</i> , <i>soscheckin</i> , and <i>soscheckout</i> commands.

Testing Tasks

Task Name	Description
Junit	Runs tests from the Junit testing framework. This task has been tested with JUnit 3.0 up to JUnit 3.7; it won't work with versions prior to JUnit 3.0.
JunitReport	Merges the individual XML files generated by the Junit task and applies a stylesheet on the resulting merged document to provide a browsable report of the testcases results.

Appendix X – Pong, Breakout and MindSweeper

Pong

The first arcade version of Pong (not the first time it was tried on a CRT screen) was built by Al Acorn for Atari. The game started the quarters rolling for the company and started the entire game business as we know it today.

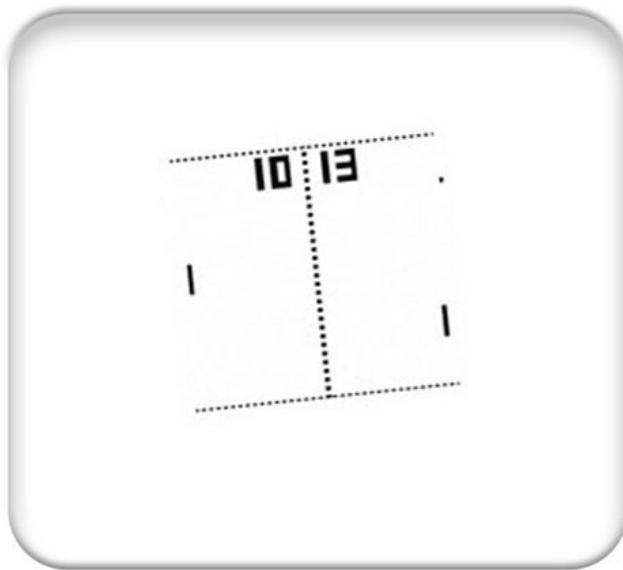


Figure 120 -The game PONG as it appears on a T-shirt

The game was a two player game where the only instructions were “Avoid missing ball for high score.”

Breakout

The story behind the original video game Breakout is one that combines hubris and the big double-cross. Nolan Bushnell, the founder of the game company Atari, came up with a variation of the game Pong (the video game that started the entire video game industry) where the player tried to clear from the top a row of bricks with a ball. You can go online and play a Flash version of the game at <http://smasher.the-game.us/>.

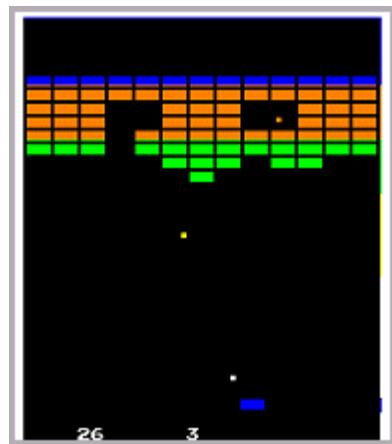


Figure 121 – Breakout

Nolan Bushnell wanted to keep the construction of a Breakout game cabinet circuitry²⁵ as cheap as possible so he challenged all his engineers to a bonus for whomever came up with the best (read cheapest) design, that is reduce the number of chips required. Steve Jobs (of Apple fame) was an Atari employee²⁶ at this time. Jobs talked his best friend Steve Wozniak (the inventor of the Apple computer) to try his hand. Jobs informed his friend Woz that the bonus money for reducing the original Breakout design was

\$750 when in fact the bonus was actually \$100 for each chip removed. Woz managed a great feat of engineering and reduced the design by 50 chips! What is more incredible is that he managed this in four days. Woz only got \$375 (half the fictitious bonus) while Jobs pocketed the rest.

The rest is of course history. The two Steve's went on to start up the company named Apple and Atari went on to make the worst²⁷ game ever – E.T.



Figure 122 - Steve Jobs and Steve Wozniak

MindSweeper

From: [http://en.wikipedia.org/wiki/Minesweeper_\(computer_game\)](http://en.wikipedia.org/wiki/Minesweeper_(computer_game))

Minesweeper is a single-player [computer game](#). The object of the game is to clear an abstract [minefield](#) without detonating a [mine](#). The game has been rewritten for nearly every [system platform](#) in use today. The most well-known version is [Minesweeper for the Windows platform](#), which comes bundled with later versions of the operating system.

²⁵ This was in time when video games were built using discrete logic circuits and not software (via a microprocessor).

²⁶ It was rumored that Jobs was spy for Bushnell since the Engineering team left Bushnell in the dark about what they were working on.

²⁷ It is rather difficult to make a game any dreadful...but many companies have come close.

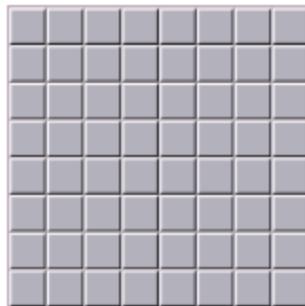


Figure 123 - Start of game

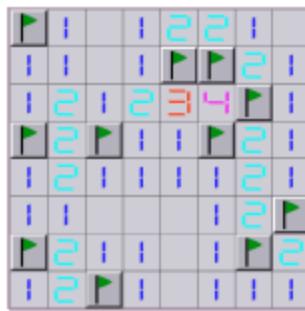


Figure 124 - Finished game

When the game is started, the player is presented by a grid of blank squares. The size of the grid is dependent on the skill level chosen by the player, with higher skill levels having larger grids. If the player clicks on a square without a mine, a digit is revealed in that square, the digit indicating the number of adjacent squares (typically, out of the possible 8) which contain mines. By using logic, players can in many instances use this information to deduce that certain other squares are mine-free (or mine-filled), and proceed to click on additional squares to clear them or mark them with flag graphics to indicate the presence of a mine.

The player can place a flag graphic on any square believed to contain a mine by right-clicking on the square. Right-clicking on a square that is flagged will change the flag graphic into a question mark to indicate that the square may or may not contain a mine. Right-clicking on a square marked with a question mark will set the square back to its original state. Squares marked with a flag cannot be cleared by left-clicking on them, though question marks can be cleared as easily as normal squares. The third question mark state is often deemed unnecessary and can be disabled so that right clicking on a flagged mine will set it back to its original state

right away so mines flagged in error can be corrected with one right-click instead of two.

In some versions of the game, middle-clicking (or clicking the left and right buttons at the same time) on a number having as many adjacent flags as the value of the number reveals all the unmarked squares neighboring the number; however, one forfeits the game should the flags be placed in error. This method is a very useful tool when trying to beat a high score. Some of those implementations also allow the player to move the mouse with the right mouse-button held down after marking mines; the player can then left-click on multiple numbered squares while dragging with the right mouse-button, in order to clear large areas in a short time. As an alternative to clicking both buttons at the same time players can also middle-click or shift-click on fully-flagged numbers.

Some implementations of minesweeper have a built in cheat option where the game will set up the board in favor of the player by never placing a mine on the first square clicked; some also change the board so the solution does not require guessing.

Appendix X – The Top 25 Platformers you should check out

From: <http://arcadesushi.com/top-25-best-2d-platformers/>

25. Actraiser
24. New Super Mario Bros.
23. Sonic Advance 3
22. Super Ghouls N Ghosts
21. Fez
20. Henry Hatsworth in the Puzzling Adventure
19. Wario Land: Super Mario Land 3
18. Rocket Knight Adventures
17. Ninja Gaiden
16. Ducktales
15. Rayman Origins
14. Earthwork Jim
13. Braid
12. Super Meat Boy
11. Cave Story
10. Super Mario Bros.
9. Kirby Super Star
8. Super Mario World
7. Mega Man X
6. Donkey Kong Country 2: Diddy's Kong Quest
5. Castlevania: Symphony of the Night
4. Super Metroid
3. Sonic 3 & Knuckles
2. Mega Man 2

March 30, 2014

[BUILD A 2D PLATFORMER - JAVA]

1. Super Mario Bros. 3