Project 9: High-Level Programming

Background

This project introduces Jack - a simple, Java-like, object-based programming language. Before building a Jack compiler in projects 10-11, it makes sense to become familiar with the language itself. That's the main purpose of this project. And... there is also the thrill of writing a little computer game in a cool little language.

Objective

Adopt or invent an application idea like a simple computer game or some other interactive program. Examples include Tetris, Snake, Hangman, Space Invaders, Sokoban, Pong, etc. Your job is to implement this application in Jack. You don't have to create a complete application. For example, you can create a basic version, or part of, some simple game or cool interaction.

The Jack OS

Writing Jack programs requires working with the Jack OS, just like writing Java programs requires working with the Java class library. The Jack OS can be viewed as a collection of software services that extend the basic language's capabilities and close gaps between it and the underlying hardware. Here is the Jack OS API.

The Nand2tetris Software Suite includes two Jack OS implementations. One OS implementation was written in Jack, and was then translated using a Jack compiler into the set of 8 class files Math.vm, Screen.vm, Output.vm, Keyboard.vm, Memory.vm, String.vm, Array.vm, and Sys.vm. These files are stored in the nand2tetris/tools/os folder in your computer. The second OS implementation was written in Java, and is an integral part of the supplied VM emulator, also available in nand2tetris/tools. Below we explain how to use each OS version.

Writing and Editing Jack Programs

A Jack application is a collection of one or more .jack files. You can write and edit these files in any text editor that handles plain text files. For example, if you wish to use Notepad++, you can follow these steps:

- Install the latest version of Notepad++ on your computer;
- Copy these Jack syntax conventions to the file \notepad++\userDefineLang.xml on your computer.

This action will customize Notepad++ for writing Jack programs on your computer. Of course you are welcome to use any other text editor that suites you. If you wish to customize the editor for Jack programming (not an essential tweak, really), you will have to figure out yourself how to do it.

It is perhaps worthy to note that we don't want to make a big deal out of Jack programming. We use Jack mainly as a means for learning how to develop simple compilers and operating systems, as we'll do in projects 10, 11, and 12. Therefore, there is no need to spend too much time honing your Jack programming skills.

Resources

The relevant reading for this project is book chapter 9. You will need three tools: a text editor for writing your Jack programs, the supplied JackCompiler for compiling them, and the supplied VM emulator for executing and testing them. This list of Jack OS error codes may come handy during your program development process.

Bitmap editor: If you wish to develop an app that has graphical elements, consider using this powerful Bitmap Editor, created by Eric Umble. Here is a demo of a Dino Adventure game that Eric developed with this editor.

Game ideas: This Youtube channel contains tips about programming "retro" computer games. You can consult it for some ideas, as well as technical tips.
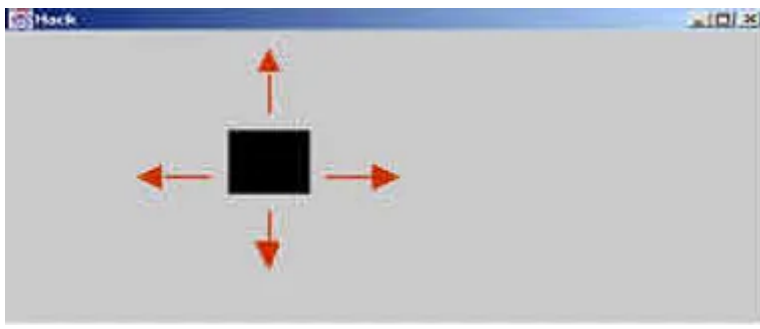
Submission note

If you take this course in Coursera, you have to submit a zip file, according to the submission instructions listed in the Coursera website. Note that in addition to the compiled VM files, your submission must include a folder named "source", containing all the source Jack files of your application. Your project reviewers will judge both the user experience as well as the quality of your Jack code. Make sure that your Jack code is readable and documented.

Sample Jack Program

Square Dance: this simple interactive program allows the user to move a graphical square around the screen:

When the program starts running, a square of size 30 by 30 pixels pops up at the top-left corner of the screen. The program then listens, and responds, to the following key-pressing events:

- **right-arrow** moves the square to the right;

- **left-arrow** moves the square to the left;

- **up-arrow** moves the square up;

- **down-arrow** moves the square down;

- **x** increments the square's size by 2 pixels;

The user can also control the square's size.

- **z** decrements the square size by 2 pixels;

The program's handling of the keyboard and the screen illustrates programming practices that are applicable to developing interactive Jack applications in general. Therefore, it is recommended to explore the program's code, which is given below.

- **q** quits the program.

Animation speed: can be controlled by changing the delay constant in the moveSquare method in the SquareGame.jack class source code.

The source code of this Jack program is available in your nand2tetris/projects/09/Square folder.

Compiling and Running a Jack Application

1. Create a folder, say MyApp.
2. Write your application - a set of one or more .jack files, and store them in the MyApp folder.
3. If you wish to use the compiled OS version (set of 8 .vm files), copy these .vm files from the nand2tetris/tools/os folder into your MyApp folder. If you wish to use the OS version which is built into the VM emulator, this step is not necessary.
4. Use the supplied Jack compiler to compile the MyApp folder. The compiler will translate all the .jack classes found in the given folder into corresponding .vm files. If the compiler complains about errors, debug the relevant class file and re-compile MyApp until there are no errors.
5. To test your program, load the MyApp folder into the supplied VM emulator. Then run and test the program. Any problems? debug a recompile.

Which OS version to use is up to you. The built-in version is faster. The VM emulator does not care which OS version is used, for the following reason: Suppose you've loaded a folder into the VM emulator, and proceeded to execute it. Whenever the emulator detects a call for some OSclass.function, it checks if this function is available in the loaded code base; if so, it executes this function's VM code; otherwise, it reverts to using the built-in implementation of this OS function. Therefore, you are welcome to use all, or some, or none of the compiled OS class files, as you see fit. This flexibility will be quite a boon when we'll develop and unit-test the Jack OS in project 12.

Some more tips

- To get an idea for your app, watch some existing Nand to Tetris apps. The "Cool Stuff" section of this website features some impressive apps, and you can find many more in Youtube.
- Play with some of the Jack programs supplied in nand2Tetris/projects/09 and in nand2Tetris/projects/11, and review their code.
- Plan your app carefully: Design all the graphical objects and user interaction on paper. Then think about classes, methods, an API, and a testing strategy. Then start implementing and testing.
- Documentation: Use your judgment and follow standard practices.
- Optimization: To boost animation speed, use the bitmap editor supplied in nand2tetris/projects/09.
- Important: Have fun!