An Inherent Conflict in Using IDEs in Computer Language Courses

Ronald I. Frank, DPS (Computing)
Associate Professor, CS & IS Departments
Pace University
860 Bedford Rd.
Pleasantville, NY 10570
Phone: (914) 773-3444
Email: rfrank@Pace.edu

Abstract

The computer language or programming part of the curriculum requires hands-on problem solving program development. In industry, the tool of choice for program development is an IDE (Integrated Development Environment). We would like to use this tool in teaching. Learning to use an IDE is as large a task as learning programming in courses which use it, so we have to limit its uses to appropriate levels for each course. However, there still is an inherent conflict when using an IDE in teaching due to conflicting requirements placed upon an IDE by early program development courses versus the requirements of advanced courses and industrial program development processes.

Early courses require students to learn the structure and function of the language and language system in solving basic problems. An IDE is merely a way to: 1) ease the typing burden, 2) organize files, 3) aid syntax error detection, 4) easily manage searching language documentation, and in later work, 5) control debugging. On the other hand, industrial IDE use requires it to do as much as possible automatically for the developer. This involves automating the steps just listed and even automatically generating code using code templates.

Automatic code generation using code templates destroys the student's motivation to learn code structure and function in early courses. It solves most of the problems for the student thus depriving students of having to learn language and language system basics. The resolution of this conflict is to use the beneficial functions of an IDE but not automatic code generation. In first courses, we should not use automatic code generation.

We show how to get around code generation in a sample Java IDE (JCreator). Other IDEs are circumvented in a similar way. This process also has benefits for grading and assignment management in a programming course.

Keywords: Integrated Development Environment, IDE, Java Education, Code Generation.

Introduction

The computer language or programming part of the curriculum requires hands on problem solving. The industrial tool of choice for software development is an IDE (Integrated Development Environment). Ideally, a computer language course introduces an IDE, and regularly uses it. This eases the student's burden in the class and better prepares the student for advanced work and jobs in industry.

However, IDEs are designed for industrial use. They are designed to ease the developer's burden., in part by automatic code generation using code templates. The developer initially indicates the type of project to be developed and the IDE uses templates to generate as much skeleton code as possible, to get the developer started, while saving programmer time and effort.

This creates an inherent conflict in classroom use. We would like to use some of the features of the IDE but code generation is not one of them. For the student, the IDE should be merely a way to ease the typing burden, organize files, aid syntax error detection, and later to control debugging.. Code generation, at least in early courses, would deprive the student of the opportunity to understand the structure and syntax of a problem's solution. In fact, early lessons are aimed specifically at teaching the structure and syntax of standard problem solutions in the given language and system, and forcing the student to figure out how to formulate them for practice problems.

The instructor's solution to this dilemma, we suggest, is to recognize the conflict, understand a particular IDE's template usage, and then to learn how to circumvent it. Students are then directed to use the IDE in the modified manner. This gives them the effort savings offered by an IDE without cheating them of the language learning experience.

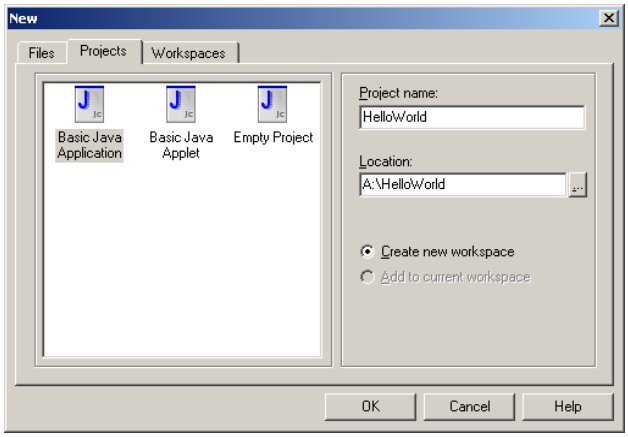
In later courses, after students have mastered the basics, they can be taught to use the more efficient code generation templates.

We present this argument in the context of a typical low cost ($35) IDE for the Java Language (JCreator Pro - academic version) and show how to avoid the use of code templates for a typical first year course in Java. We included a discussion of the advantage of this circumvention for the instructor's classroom management of the program grading process.

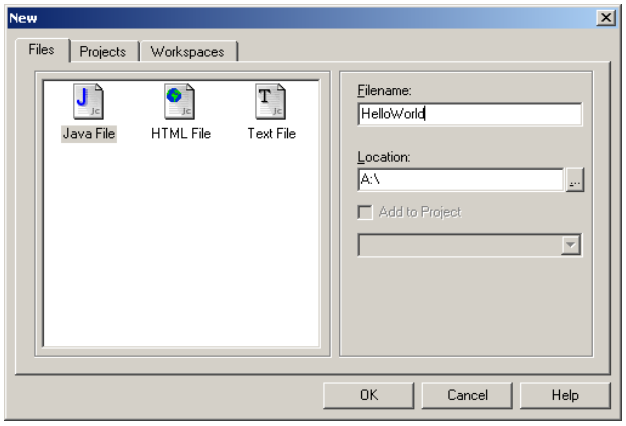
1. Problem Statement

(In the Context of JCreator Pro)

The suggested industrial method for starting program development is to establish a "project", even for the simplest one file programs. The following screen is the initial IDE response to choosing FILE NEW on the menu bar. It defaults to a Java Application Project format, which will automatically embed template code into the initial code text file. This is also true of the freeware version of JCreator, and other IDEs [1, 2, 4, 5, 6].



Alternatively, the programmer could simply start by opening a blank Java (text) file.



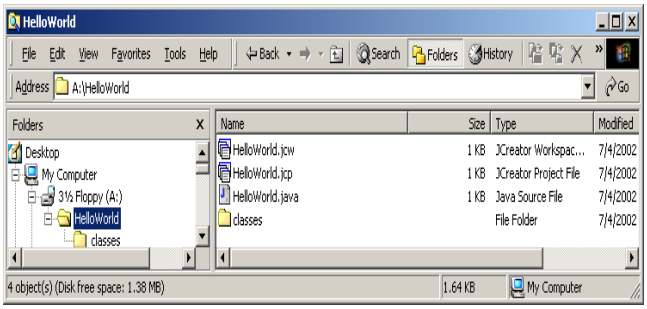
The difference is profound. The following figure is the result of declaring a project. The code details are not important. What is important is that all of this code is automatically generated from a template. Starting with just a Jave File presents a blank page to the student!

We would like to keep classroom discussions of the IDE to a minimum and concentrate instead on the language and language system. Fully learning an IDE is as large a task as multiple programming courses.

```
/*
 * @(#)HellowWorld.java 1.0 02/07/04
 *
 * You can modify the template of this file in the
 * directory ..\JCreator\Templates\Template_1\Project_Name.java
 *
 * You can also create your own project template by making a new
 * folder in the directory ..\JCreator\Template\. Use the other
 * templates as examples.
 *
 */
package myprojects.hellowworld;

import java.awt.*;
import java.awt.event.*;

class HellowWorld extends Frame {

   public HellowWorld() {
      addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent e) {
            dispose();
            System.exit(0);
         }
      });
   }

   public static void main(String args[]) {
      System.out.println("Starting HellowWorld...");
      HellowWorld mainFrame = new HellowWorld();
      mainFrame.setSize(400, 400);
      mainFrame.setTitle("HellowWorld");
      mainFrame.setVisible(true);
   }
}
```

This template code is a great help - if the student already understands windows, packages, java programming, objects (especially output streams), and system functions. All are in the generated code. The code syntax coloration is very useful. (It does not show in a black & white reproduction).

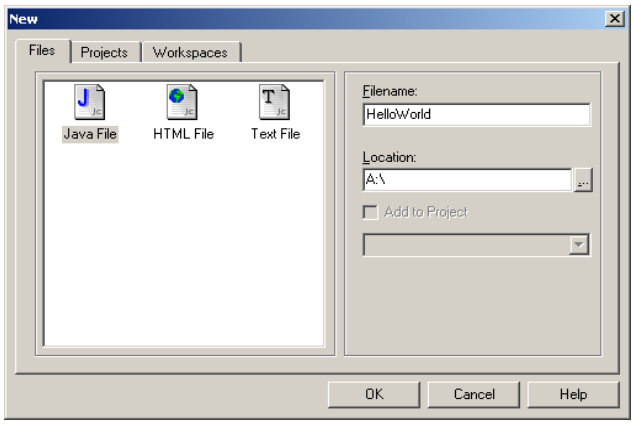
The next screen-shot is the structure of the directories and files created by the IDE. Again, this is a great help if you already know what they are all about, which a beginning student usually does not know. (This directory structure could have been assigned to the hard drive).



If all we are trying to do is a Hello World, it is better to simplify this process and not use automatic code generation. We show how to simplify this in the examples that follow. This applies to JCreator Pro but holds also for the free JCreator LE. A similar process can be used for other IDEs [1, 2, 4, 5, 6].

2. Examples

On the JCreator menu bar choose File & New:



This creates a simple empty Java file in the assigned directory. (The directory could instead have been chosen to be on the hard drive). The student must now type the problem code into that totally empty file. Here is a sample student input. It requires only that the student has started to learn some basic Java syntax and semantics:

```
/*
Student Name
Assignment #1
1.1
Welcome.java
*/

//Welcome.java: This application program prints Welcome to Java!
public class HelloWorld
{
   public static void main(String[] args)
   {
      System.out.println("Welcome to Java!");
   }
}
```

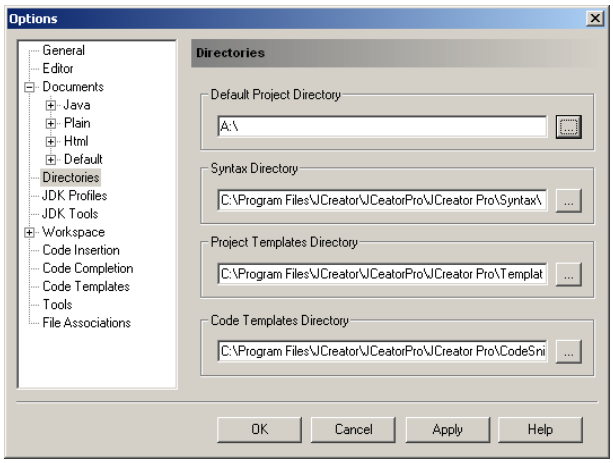
Before we proceed to the discussion of compilation, we must divert our attention to the underlying default settings which all IDEs use. In JCreator there are five major default choices:

1. compilation of application code
2. compilation of applet code
3. running applications
4. running applets
5. running the debugger

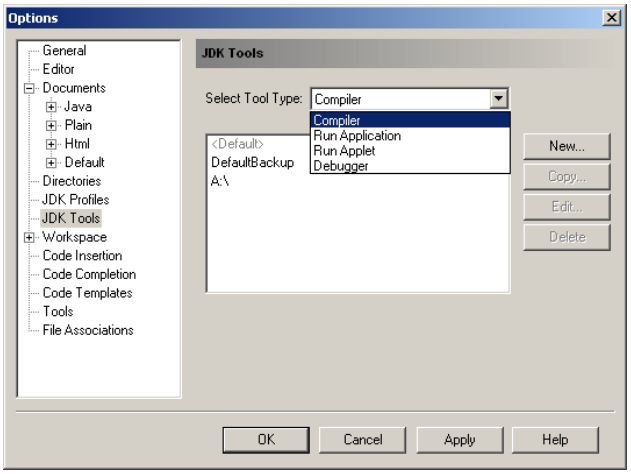
These defaults simplify the advanced user's use of the IDE. The first four generate code that the beginner is not ready to use. The fifth requires an existing project.

The defaults 1 and 3 must be set for this Hello World application for either the template use case or the simple case. The JCreator system comes with defaults set for template use that use Java Windows, which are an advanced topic in many texts. We make a few changes to the defaults once at the beginning of the course and reap simplifying benefits all course long.

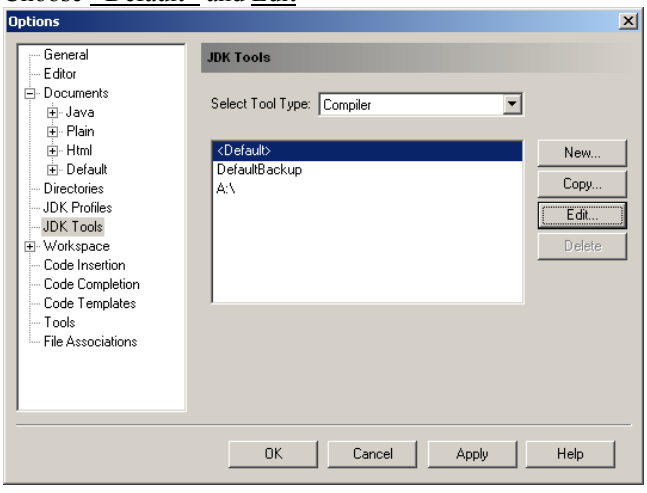
Even though there is no "project", a default path is used. It could be on the hard drive. On the menu bar choose Configure & Options (Choose Directories & change only the top choice) [the ellipsis button allows you to search your computer's directory paths]



We have to set the compilation defaults, which differ in the template or non template cases. The instructor provides this help once at the start of the course. On the menu bar choose Configure then Options (Choose JDK Tools & Compiler)

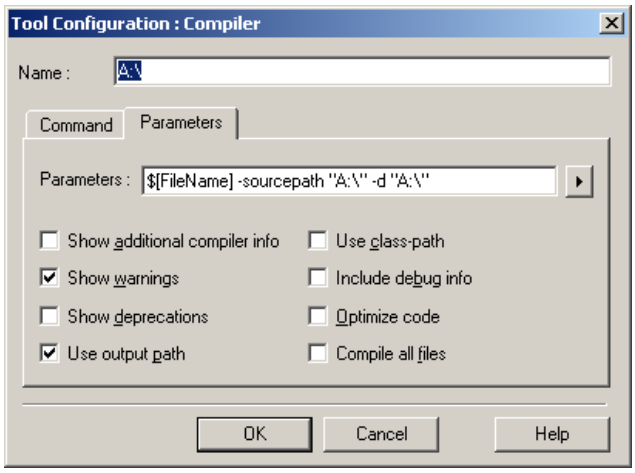


Choose <Default> and Edit



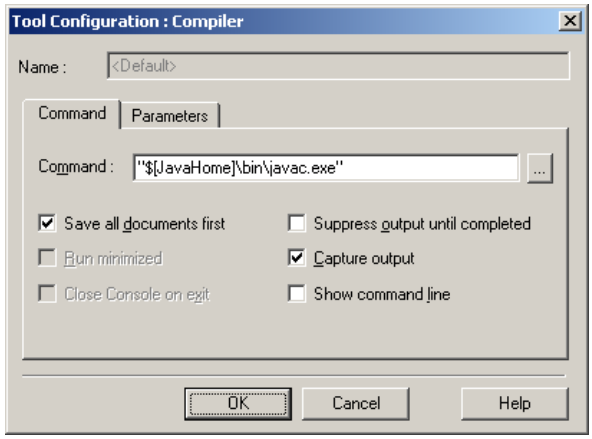
[The DefaultBackup and A:\ are copies I put in to remind me of the <Default> factory settings for hard drive or my settings for A:\ use.]

Change Name & Parameters. The Command tab does not change because it invokes the compiler in all cases.

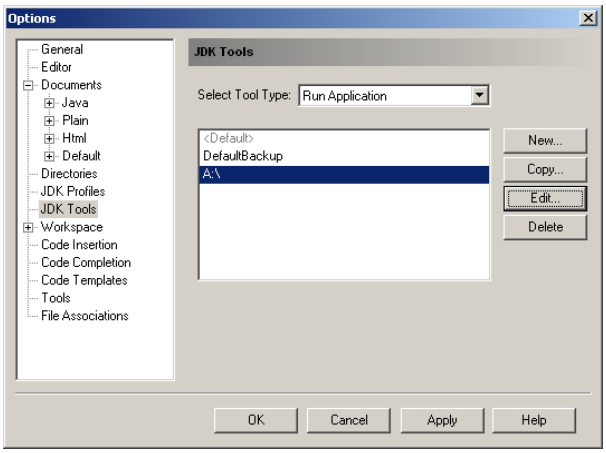


These are the settings for using the floppy as a source and sink for the compilation. With these few changes we can now create a simple Java source file and compile it. We could use either the hard drive or the floppy. The right arrow-head button gives you help in choosing parameters.

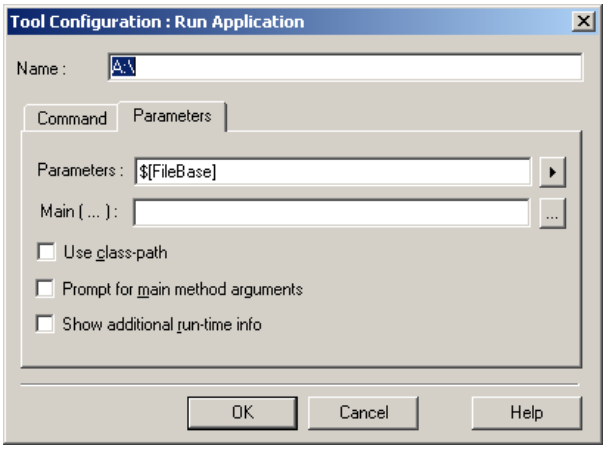
We don't touch the command choice which invokes the same compiler in all cases.



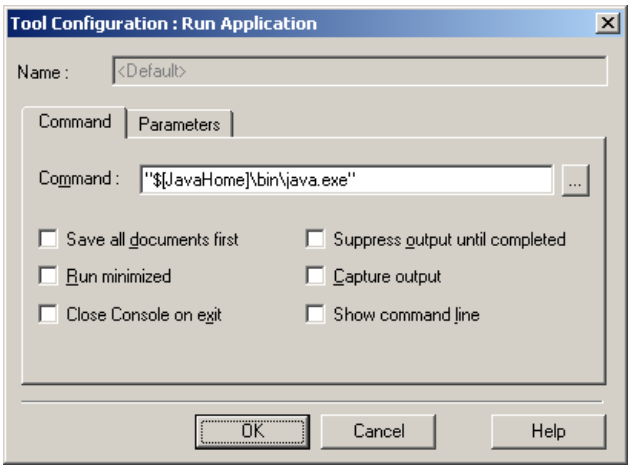
On the menu bar choose Configure then Options (Choose JDK Tools & Run Application).



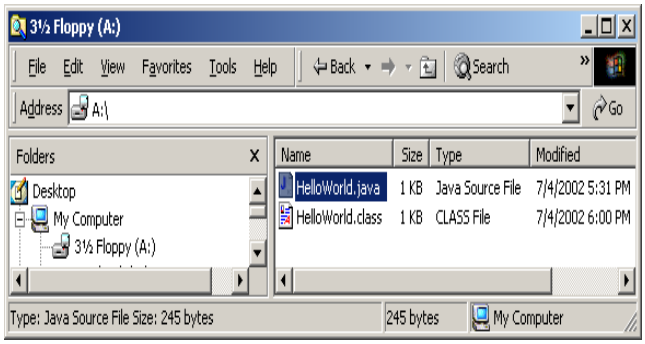
Then edit for running from A:\.



Again, we don't change the command which invokes the runtime environment. We specifically don't want to capture output (in the IDE window) since we want it to go to a "console window".



The compilation results in the following directory contents (we used A:\ instead of the hard drive). The .java file is the simple source and the .class file is the compiler output. This is as simple as it could be.



The output on the console window looks like this [in Windows 2000] (collapsed for printing):

3. Discussion of the Advantage of the Use of this Method for the Student

Once the student learns the proper default settings, compilations and executions are run from the floppy or the hard drive. The student gets no template code before its time (in the curriculum). Also, the task of learning to use the IDE is cut down to a few settings.

There are some instructors and students that prefer, in the Java context, to use the Sun DOS prompt environment for development. There is one overt reason given for this preference: the student is explicitly aware of the use of the compiler, "Javac" and of the runtime environment invocation , "Java". There is, in my opinion and experience, a stronger covert reason: fully learning any given IDE is a job as big as learning Java itself! A hidden sub text reason to this is that such a large effort is a probable waste because any given employer will probably choose some other IDE.

My answer to the overt reason is that paying attention to the few default settings shown above brings home the Javac and Java environments. My answer to the covert reason and sub text is that learning any IDE is an incremental process that should actually be spread over many courses. If the student goes on to courses involving Java Beans, Enterprise Beans, "fancy" HTML, XML, or multi-person development projects, all of which benefit from IDE features, the student learns just-in-time only those features that are needed that course.

Even when using a project for debugging a similar circumvention works.

4. Discussion of the Advantage of the Use of this Method for the Instructor

With this circumvention method, students are free to develop solutions on lab machines or their personal machines as they wish. They will have to make simple changes to the default settings for use of the floppy versus the hard drive. However, the instructor is now free to request that the students hand in assigned problems one to a diskette (or email file) with no extraneous code.

The instructor can standardize on a grading procedure that assumes a single standard set of defaults in the IDE. The submitted code would then contain only student code.

5. Conclusion: Problem and Solution

By defining a standard set of simplifying defaults in the IDE, the instructor can ensure that the students do the coding (modulo inter-student copying) and thinking about all of the syntax. Learning the IDE is minimized. Also the instructor simplifies grading assignments by standardizing the student hand-ins and standardizing running procedures for grading.

6. References (Java IDEs)

[1] Forte : Sun Co. Forte Environment.
http://forte.sun.com/ffj/documentation/relnote40.html

[2] JBuilder: Borland & Enterprise Studio:
http://www.borland.com/estudiojava/pdf/estj4_datasheet.pdf

[3] JCreator IDE: http://www.JCreator.com/ Click "Features".

[4] Kawa (discontinued 10/31/2001)
http://search.atomz.com/search/
?sp-a=sp1001395b&sp-p=any&sp-q=kawa

[5] Visual Café (discontinued 2/2003)
New product:
http://www.webgain.com/products/webgain_studio/
feature_matrix.html

[6] Visual J#.Net : Microsoft Visual Studio.Net.
http://msdn.microsoft.com/vjsharp/

==========

[7] Frank, R. I., "JCreator 'Just-In-Time' Tips". Pace Technical Report #181 Sept. 2002. This is an 83 page screen-shot-based set of tips for V2.5 for projects, applets, applications, etc. Also available from the author at rfrank@pace.edu as a PDF file ~2.8M.