From Concrete to Abstract, the Power of Generalization

Christopher H. Nevison
Colgate University
Hamilton
NY, 13346
315-228-7589
chris@cs.colgate.edu

ABSTRACT
We describe an assignment for students in a software engineering class or advanced programming class with emphasis on design. In the assignment students are given a program that solves a maze, with a display that shows the steps toward the solution. The given program has three variations of an iterative search (implemented using the strategy pattern), depth-first search, breadth-first search, best-first search (the latter using a priority queue). The students are asked to disentangle the problem-specific aspects of this code from the search strategy so as to define an abstract problem solver that can be applied to any problem fitting the model of step-by-step searching of a solution space. This requires three steps: defining appropriate interfaces that specify the information about the problem needed by the abstract solver, defining the abstract solver to find a solution using these interfaces, and defining the maze problem so as to implement these interfaces. With the abstract solver created, students should also be able to implement other problems fitting this model, such as the word-ladder game or a search in a graph for a Hamiltonian circuit, so that the abstract solver can be applied to them. The assignment demonstrates the power of generalization using abstract classes and interfaces as a bridge between concrete problems and the solution algorithm. This assignment is intended for a course in software engineering or an advanced programming course with emphasis on design. Students should already be familiar with object-oriented programming, inheritance, abstract classes and interfaces.

Categories and Subject Descriptors
D.2.2 [**Software Engineering**]: Design Tools and Techniques – *object-oriented design methods.*

General Terms
Algorithms, Design.

Keywords
Object-oriented design, Pedagogy.

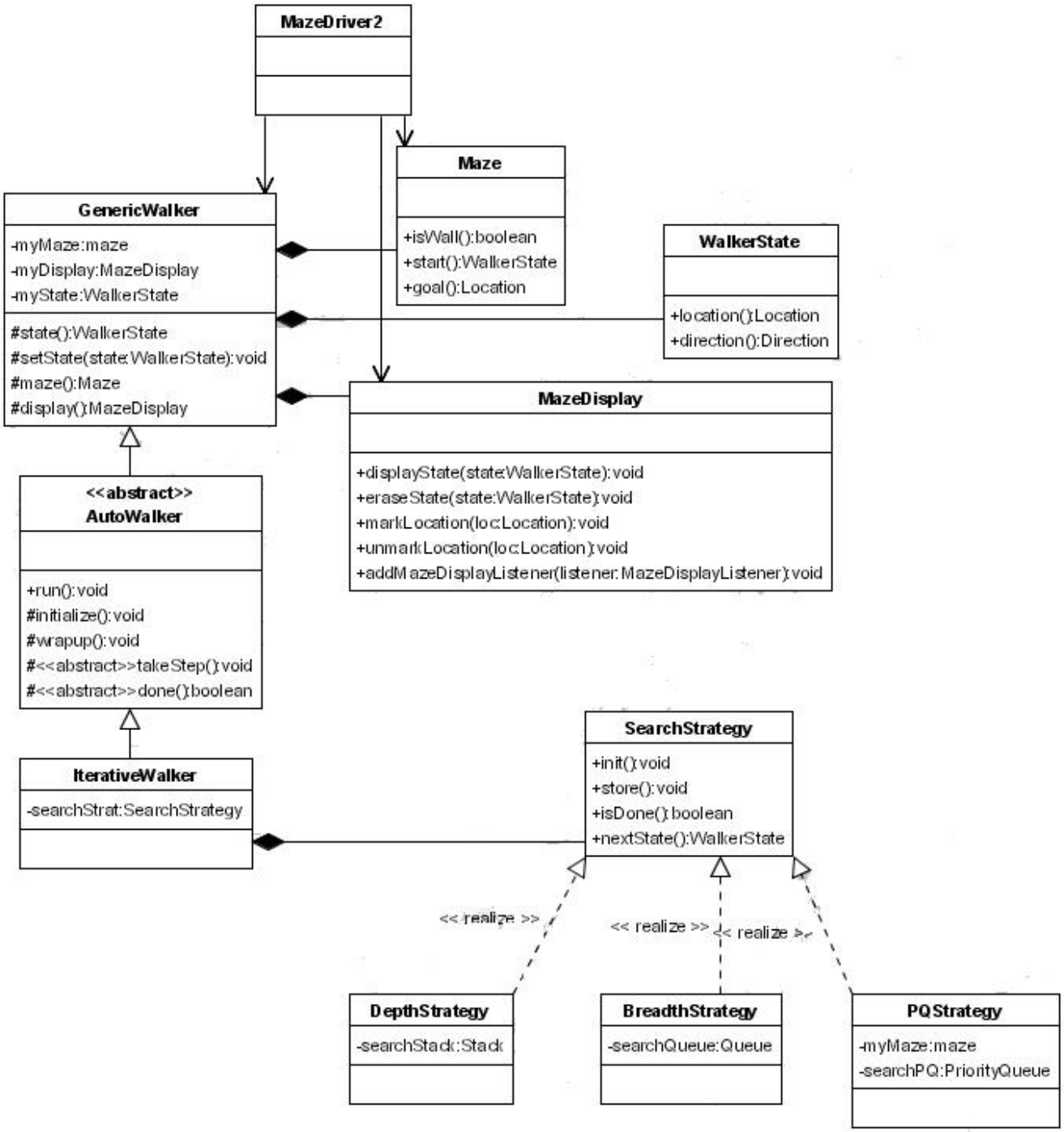
1. INTRODUCTION
We describe a "nifty assignment" that teaches students some ideas about object-oriented design. The purpose of the assignment is to teach students how one may generalize from a program that solves a particular problem to create a program that can be applied more generally. The given program has three variations that solve a maze using depth-first, breadth-first, or heuristic, best-first searches, respectively using a stack, queue, or priority queue to control the search strategy. The student is asked to separate out the problem-specific aspects of this program from the problem-solving algorithm to produce a program where a module for a different problem, which can also be solved using the same step-by-step exploration of the state space, can be plugged in. This assignment demonstrates how one can generalize from a solution to a specific problem to a more general algorithm and program, and how the tools of object-oriented design and programming can be used to accomplish this task. Samples of such problems are solving the word-ladder game, or finding a Hamiltonian circuit in a graph.

OOPSLA'04, Oct. 24–28, 2004, Vancouver, British Columbia, Canada.
ACM 1-58113-833-4/04/0010.

2. WHY IS THIS ASSIGNMENT NIFTY?
Students are motivated to find a solution that demonstrates their understanding of object-oriented design. The first part of the assignment is to create the design using UML diagrams. The second part of the assignment is to implement the design. Students get the satisfaction of creating a moderately large program that has the flexibility to be adapted to different problems. The program can demonstrate this with graphical output for the maze problem as well as appropriate output for the word-ladder or Hamiltonian circuit problems.

Since students are refactoring a given program to create the new more flexible design, they have the code implementing the algorithms given. Consequently, they do not get hung up in the implementation details of the algorithms as they might if they were creating this program from scratch. Instead they can focus on the design issues that are the main purpose of the assignment.

The problem is well-defined but has enough room for original ideas in the design. Student solutions create opportunities for discussing different issues that come up in the design. The program illustrates some design patterns including the strategy pattern which is used in the original program and adapted by the students and a variation of the bridge pattern for the students' designs, whereby an abstract problem description forms a bridge between the concrete problem description and the solution method.



3. TARGET AUDIENCE

The target for this assignment are students that have already had an introduction to object-oriented programming and are learning more advanced design and programming techniques, either in an advanced programming class or a software engineering class.

Students should understand object-oriented programming in Java. They should understand inheritance and the use of abstract classes and interfaces. The initial program given here is in Java, so the students are expected to implement their new design in Java as well; however, this same assignment could be converted to another object-oriented language.

4. IDEAS AND SKILLS TAUGHT

This assignment teaches students how they can work from the application of an algorithm to a particular problem in a program to a more general program that implements that algorithm in a way that can be applied to many different problems. This demonstrates the power of generalization from a single-purpose program to a more abstract and therefore more reusable program.

The assignment also demonstrates the use of design patterns including the strategy pattern [1], which is embedded in the given program and is modified by the students, and a variation on the bridge pattern [1]. The latter comes up when students develop an abstraction for a problem that bridges between the concrete problem (e.g. maze or word-ladder) and the solution program.

The design of this program teaches students about the specification of an interface and its implementation in Java as an abstract class or Java interface. This idea is essential to the generalization of the program.

5. THE ASSIGNMENT

The students are given a program and UML diagram of its design, as shown in figure 1. In the given program the use of the maze is intertwined with the search algorithm. For example, in the code generating the next state to be checked, the maze method isWall is used to verify whether this is a state that can be accessed. The details of the assignment as used in a software engineering course at Colgate University can be found at the following url:

http://cs.colgate.edu/faculty/nevison/nifty

5.1 Design of a General Program

The first part of the assignment is for students to refactor the given design so that the algorithmic parts are separate from the problem-specific parts, with an abstract problem specification bridging between them. Their design should include a UML diagram and a description of the classes used in their new program. I usually assign the problem for pairs of students.

5.2 Implementation

The second part of the assignment is to implement their new design in Java. Here they can use much of the code that they are given with the original program, so they tend not to get hung up in algorithmic details. However, they are challenged to be sure that their design really successfully separated problem-dependant aspects of the program from the algorithmic aspects.

5.3 Extension

The third part of the program asks the students to demonstrate the reusability of their new design by developing classes implementing a new problem that can be plugged into their program in place of the maze problem. The word-ladder game is one good example of this that students can easily understand and for which a simple display is adequate. An alternative is the problem of finding a Hamiltonian circuit in a graph. This allows them to be creative about a display, although they should be discouraged from spending too much time on the display module.

5.4 Student Solutions

This problem has enough room for variation so that students come up with different solutions. Some of the solutions from students in the software engineering course at Colgate are also at the url given above. The student solutions often raise interesting design issues that provide a good basis for discussions in class or lab.

5.5 Variations

I divided this assignment into two parts. The first part was the design described in section 5.1 and the second assignment was the implementation of the design described in section 5.2 and the extension described in section 5.3 In each case a pair of students worked through the whole sequence together. One could split this assignment into three parts, corresponding to the sections 51., 5.2, and 5.3. One could also have a different pair of students carry out the implementation described in 5.2, working from another pair of students' design. This would be a good test of the design.

5.6 Time Needed

I assigned pairs of students to develop a design for this problem as a lab assignment that they had one week to complete. In a second assignment students implemented their design, again working in pairs. In my class, students had two weeks for the second assignment. However, these students had taken their introductory programming course in C++ and were learning Java at they same time we were working on object-oriented program design. It might be possible to do this second assignment in one week.

6. DIFFICULTIES

The point of this assignment is design and seeing that design work, so you should not let students get hung up on language issues. Help students get over whatever difficulties they have with language issues and supply them with understanding of the language structures needed, such as abstract classes and interfaces.

Students may need help with design. However, resist the temptation to direct them toward your idea of what the design should be. Let them experiment and work toward their own design with as little guidance from you as you can get away with. Different student designs can provide opportunities for discussion of design ideas and tradeoffs.

I have not used this lab with the variation suggested above where each pair of students does the implementation of another pair's design. I think this would be a challenging but instructive exercise. Of course, the difficulties here would be that the implementing students would be dependent on the good, or poor, design document of another pair.

7. REFERENCES

[1] Gamma ,E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns* Addison-Wesley, Reading, MA, 1995