Using the BlueJ IDE in a Data Structures Course

James H. Paterson (j.paterson@bell.ac.uk)
John Haddow (j.haddow@bell.ac.uk) ,
Miriam Birch (m.birch@bell.ac.uk)
Alex Monaghan (a.monaghan@bell.ac.uk)

School of Science & Technology, Bell College, Hamilton ML3 0JB, UK, +44 1698 283100

The BlueJ IDE is an excellent tool for teaching programming using an objects-first approach. Learners are able to create, inspect and interact with instances of classes without the requirement to initiate a process with a main method.

The ability to inspect 'live' objects can provide insights in specific areas of programming beyond the basic concepts. We have made use of BlueJ in an introductory course on data structures for students who have previously learned object oriented programming using BlueJ. The course deals with stacks, queues, lists, binary trees and graphs.

Why BlueJ?

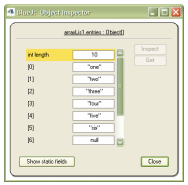
Descriptions, diagrams, animations and simulations can all play a part in understanding the way in which these structures store and provide access to their data. However, interacting directly with actual objects which have been instantiated from code which the student has written can make the data structure much more tangible.

The capability in BlueJ to **inspect** an object and to **navigate object references** to inspect associated objects allows a direct **visual exploration** of the internal representation of a data structure.

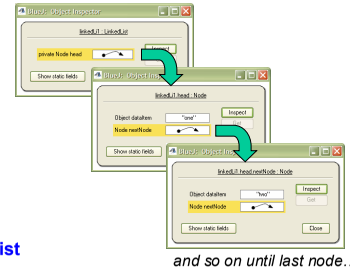
Lists

BlueJ allows the student to write a **linked list** class, create and populate an instance of the class, and visually inspect the nodes of the list. Object references can be followed from node to node to traverse the list.

This experiment provides strong contrast with a visualization of the storage in an **array list**. The point can be emphasized that data structures with the same interface can have completely different internal representations.



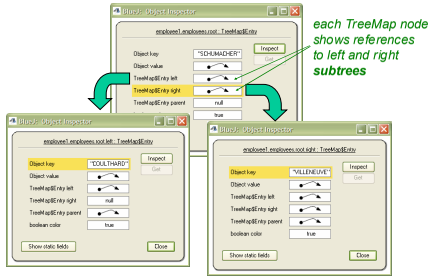
inspecting an ArrayList



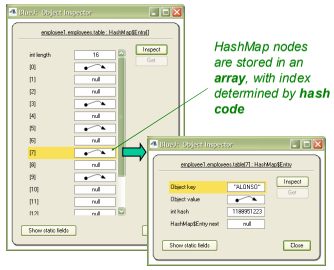
inspecting a LinkedList with the same data

Maps

Maps store data as *<key,value>* pairs. The figures show an example which uses the Java Collections Framework map classes. The same data are stored in each case, but the map can be instantiated as either a **HashMap** or a **TreeMap** by changing one line of code. The internal representations are completely different.



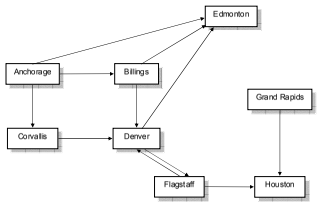
inspecting nodes of a TreeMap



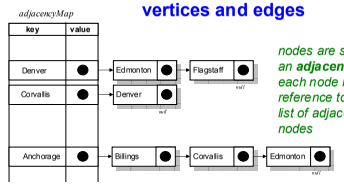
inspecting a HashMap with the same data

Graphs

A graph data structure is represented diagrammatically in terms of vertices joined by edges. However, there are a number of ways in which graphs can be represented internally, including adjacency lists and adjacency matrices. An adjacency list representation consists of a list of vertices, each of which is associated with a list of edges originating from that vertex. BlueJ allows visual exploration of such a structure, which can then be related back to the diagrammatic representation. The example is based on material from reference 1.

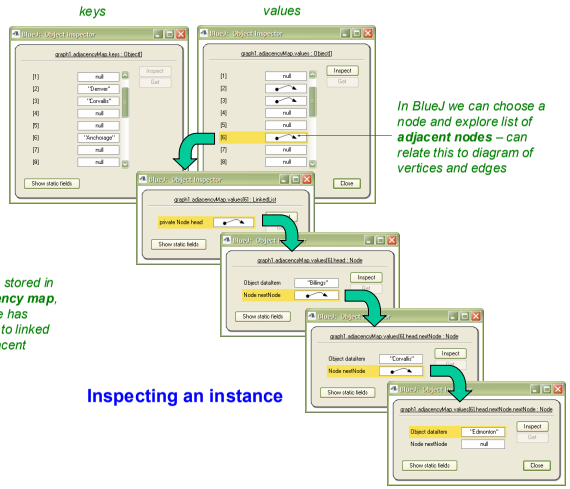


vertices and edges



nodes are stored in an **adjacency map**, each node has reference to linked list of adjacent nodes

diagram of internal representation



Inspecting an instance

Conclusion

It has been noted that it is important in taking an "objects-first" approach to pedagogy using BlueJ that the traditional programming topics such as data structures and algorithms are not neglected [2]. We believe that there is in fact great potential for tools such as BlueJ to enhance learning of these concepts.

References

1. Collins, W., *Data Structures and the Java Collections Framework*, McGraw-Hill, 2005.
2. Kölling, M., Quig, B., Patterson, A. and Rosenberg, J., *The BlueJ system and its pedagogy*, Journal of Computer Science Education, Special Issue on Learning and Teaching Object Technology, Vol 13, No 4, 2003.

http://hamilton.bell.ac.uk/datastructures