

Class 35: Introspection and Reflection

Held: Wednesday, 27 April 2011

Summary: We consider the basics of introspection in the Java programming language and some reasons we might want to use introspection.

Related Pages:

- EBoard.
- Reading: Trail: The Reflection API, available at ..

Notes:

- Reading for Friday: “gc wiki”.
- Due to prereg and College committee work, I need folks to sign up for appointments. Times at <http://www.cs.grinnell.edu/~rebelsky/Schedule/signup.2011S.txt>. Signup via email.
- CS Picnic: Friday, May 6, 2011.
- Don’t forget to vote on a t-shirt.
- EC for today’s Integrating Digital Collections into the Liberal Arts College Curriculum, 4pm JRC 209. [academic]
- EC for Poetry Convo Thursday. [academic]
- EC for Thursday’s Bites to Bits.
- EC for Friday’s CS table.
- EC for Psychologist Candidate Talks, coming soon. [academic]

Overview:

- Introspection basics.
- Application: Factory Methods.
- Application: Dynamic Loading.
- Application: Serializing.
- Application: Profiling.

Introspection Basics

- Introspection lets you write code that looks into existing objects and classes, permitting you to use such things differently.
- In some ways, introspection is like Scheme’s ability to treat the same structure as either a list or a bit of code.
- Basic introspection operations in Java:
 - Given an object, determine its class.
 - The class of an object is a value in class `Class`.

- Given the name of a class, get its class object.
- Given its class, determine its methods, fields, and constructors.
 - Each of these things is also an object.
- Given a method or constructor object, call it.
- Other introspection operations in Java:
 - Build a proxy
 - Modify the class loader
 - ...
- Why have introspection?
 - Easy way to have factory methods.
 - Many many more.

Example One: Factory Methods

- Sometimes you want to be able to build a lot of objects of the same type, and don't necessarily know that type in advance.
- By grabbing an object's class, and then grabbing the constructor from that class, you can easily make new objects.

Example Two: Dynamic Loading

- You can even load classes whose names you did not know when you wrote the program.
- How? Use `Class.forName(string)`.
- This strategy is often used for plug-in based programs.
 - That is, the program can read the name of the plug-in from the directory structure or from an initialization file, and then load the appropriate plug-ins using reflection.
- You can use this strategy for some forms of live patching.

Example Three: Serializing

- Problem: You want to dump an object to a file for reading later.
- How do you do it?
- What if you have to do the same thing for dozens of classes?
- What if you don't have access to the `.java` file?

Example Four: Profiling

- Problem: You want to make a count of how many times each method is called in any object.
- How do you do it?
- What if you have to do the same thing for dozens of classes?
- What if you don't have access to the `.java` file?