

Where To Go With OOP From Here?

Post-Graduate Research

1. programming for free (Open Source)
2. demanding professors
3. long hours and very little sleep
4. intellectual reward (name on research papers, speaker at conferences, appearances on Nightline, Turing Award, etc.)
5. interesting research fads (quantum computing?)

Professional Software Engineering

1. programming for money (Closed Source)
2. demanding managers
3. long hours and very little sleep
4. financial reward (high salary and promise of stock options)
5. latest marketing hype (e-Commerce?)

Either way, there are still many things to learn about OO design and programming, in both C++ and Java. We have just covered the basics, and it often takes a couple of years experience to become really proficient with C++. More importantly, the best programs are written by people who really understand the problem domain and can develop abstract models of elements of the problem space and translate those abstract models into working software. Good independent problem solving skills are what are needed most. **You also can't wait until the last minute to start working on a hard problem!!**

So, the challenges in software engineering are to become an expert on nuances of various operating systems, programming languages, software tools (compilers, debuggers, memory leak detectors), and modeling aids, and then applying all of that knowledge simultaneously as an aid to general problem solving in some application domain.

You can learn a lot by studying the way experienced people write software, and learn from their programming style and code organization techniques. This usually means spending time reading and understanding other people's software and asking lots of questions. Lucky for you, there is a lot of well-written open source code available on the Internet that you can go read.

Keep a critical frame of mind about technology, and try to keep things simple, instead of making them more complex.

Programming-in-the-Small versus Programming-in-the-Large

1. Large software projects are complex things to manage, small projects are often more successful
 - working in teams with others is a real challenge
 - small teams usually make the most progress
 - good communication among developers is critical
 - use of version control software is critical to co-ordinate code changes
 - sales & marketing people often drive the software requirements

 3. Software quality is extremely important
 - too much software today is buggy (open source helps address this, since more people, in theory, review the code)
 - there is real pressure to ship software as early as possible often resulting in poor quality code
 - companies often ship “alpha” (aka. very low quality) or “beta” (aka. low quality) code,
 - and let their customers test it. Some companies charge for this privilege and call it a “Developer Network”
 - documentation is quite often poorly written, and buggier than the code, but it is just as important, if not more

 4. Ego-less programming is important
 - it is often hard to divorce your ego from the software you write
 - the software should be written with the needs of the end-user in mind (which is not necessarily what *you* want)

 5. Recommended Reading
 - Debugging the Development Process and Writing Solid Code*
 - both by Steve Maquire
 - The Dynamics of Software Development*
 - by Jim McCarthy
 - Design Patterns: Elements of Reusable Object-Oriented Software*
 - by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides
 - Object-Oriented Software Construction, 2nd Edition*
 - by Bertrand Meyer
 - The Mythical Man-Month (Anniversary Edition)*
 - by Fred Brooks
-

Comprehensive Final Exam Review Material

Thinking in Java and *C++ Primer* chapter reading as assigned during the term

18 sets of lecture notes handed out in class

5 quizzes

3 Java Programs, 2 C++ Programs

Key OO topics covered:

- properties of objects: identity and state
- encapsulation of data and methods using classes
- separation of interface and implementation
- categories of operations (constructors, destructors, accessors, mutators, operators, iterators)
- object construction, object destruction, copy constructor, default constructor, initializer lists, “this” pointer
- the role of scope and location in determining object construction/destruction rules
- static object allocation versus stack (automatic) object allocation versus heap allocation
- object-oriented design, recognizing is-a, has-a, and usage relationships
- naming conflicts and the use of namespaces
- private, protected, public access control, access declarations
- static variables and static methods
- interface inheritance (subtyping) versus implementation inheritance (subclassing)

Specific C++ topics:

- friend functions and friend classes
- inheritance using private/public
- polymorphism (ad hoc, subtype, parametric), abstract base classes and pure virtual functions, templates
- operator overloading, friend operators versus class member operators
- template methods, template classes, and collection classes defined as templates (e.g., Stack, List, etc.)
- smart pointers and reference counting garbage collections

Final Exam Review Material (continued)

Specific Java topics:

- packages and name spaces
- classes, class modifiers (final)
- scope access rules (public, private, protected, and “default” package access)
- primitive data types
- reference data types
- static class variables and methods
- the Object class, object creation, destruction by garbage collection, finalize method
- arrays, array operations, vector class
- subclassing and inheritance
- overriding methods
- interfaces
- simple input/output streams
- the Applet class and simple applets
- the Socket class and use of sockets
- the Thread class and Runnable interface
- using threads and synchronized methods
- wait and notify
- exception handling

You should understand key similarities and differences between C++ and Java Object Models

- packages, classes, scopes,
- Java interfaces versus abstract base classes w/ pure virtual functions in C++
- member scope access rules
- object creation/destruction, object lifetimes
- C++ pointers (and smart pointers) versus Java reference types
- contrast polymorphism features (operator overloading, templates, subtyping)
- common object base class in Java versus templates in C++ (i.e., polymorphic container types, such as vector)
- proper subtype vs subclass vs has-a (usage) relationships
