
08 - Object-Oriented Libraries and Extensions

Motivation

- o When you write several systems, you notice that much of their code is similar.
- o Stop reinventing the wheel, try to reuse code!
- o How do you organize code reuse?
History:
 - n Copy & Paste
 - n Collect useful files
 - n Formal mechanisms to enable reuse across several people

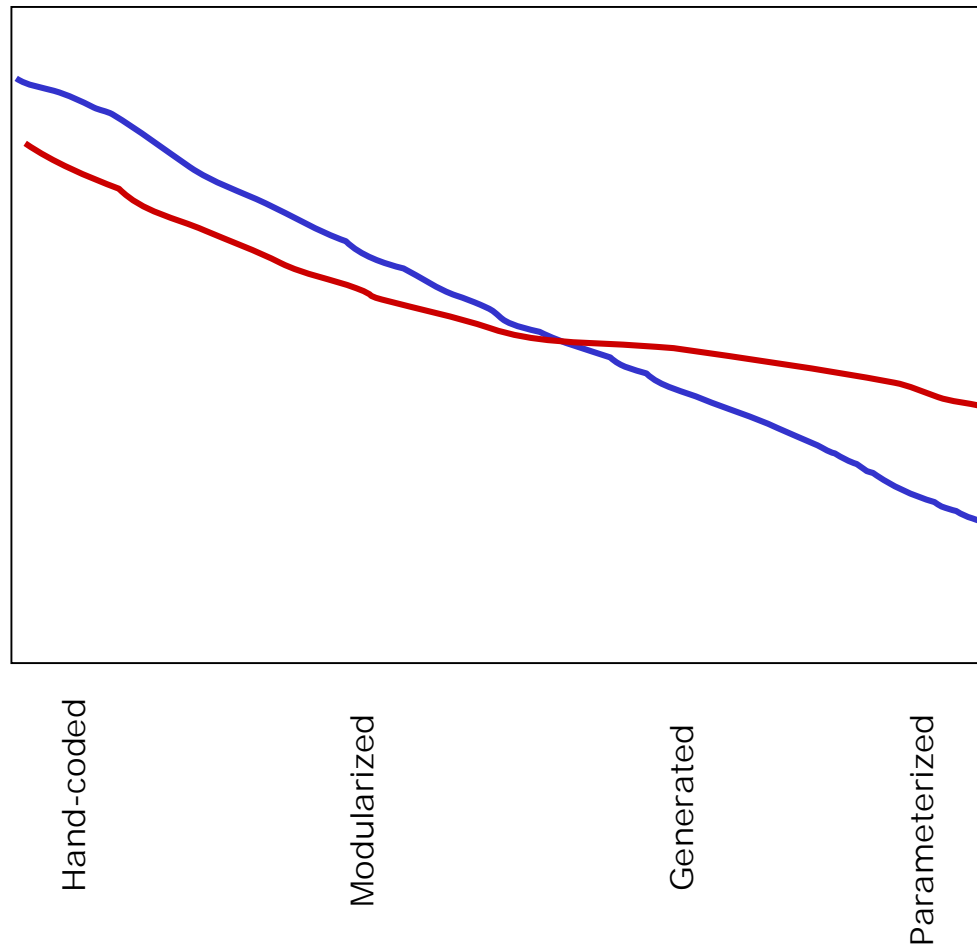
Spectrum of Software Development

- Hand-coded
- Modularized
- Parameterized
- Generated

Any of these can be used together: there are parameterized modules etc.

— meet requirements

— cost



Types of Extensions

- o Toolkits/Libraries
 - n collections of related but independent routines/classes
- o Frameworks
 - n collaborating classes that also implement workflow
- o Components
 - n system parts that can be aggregated to build whole system
- o Code generators
 - n write parts of your system's code

Any of these can be used together: A code generator can e.g. generate code to be run in a framework. A component usually runs in a framework.

Toolkits

- o Definition: A toolkit is a set of related and reusable classes designed to provide useful, general-purpose functionality. They are the object-oriented equivalent of subroutine libraries.
 - n Usage is usually determined by API
 - n Control-flow has to be managed by the programmer
 - n Toolkits emphasize code reuse.
 - n Offer isolated functionality
- o Toolkits are in practice often referred to as libraries.
- o Toolkits mostly fall in two categories:
 - n interfaces to external systems
 - n self-contained libraries of functionality

Examples of Toolkits

- o Examples
 - n Java Standard Edition libraries:
 - o XML, SQL,
 - o reflection,
 - o I/O, networking,
 - o collections,
 - o ...
 - n XML parsers for the DOM (Document Object Model)
 - n Accessing specific hardware:
 - o scanners
 - o measuring instruments
 - n Implementations of protocols
 - n Database access

Independent Functionality

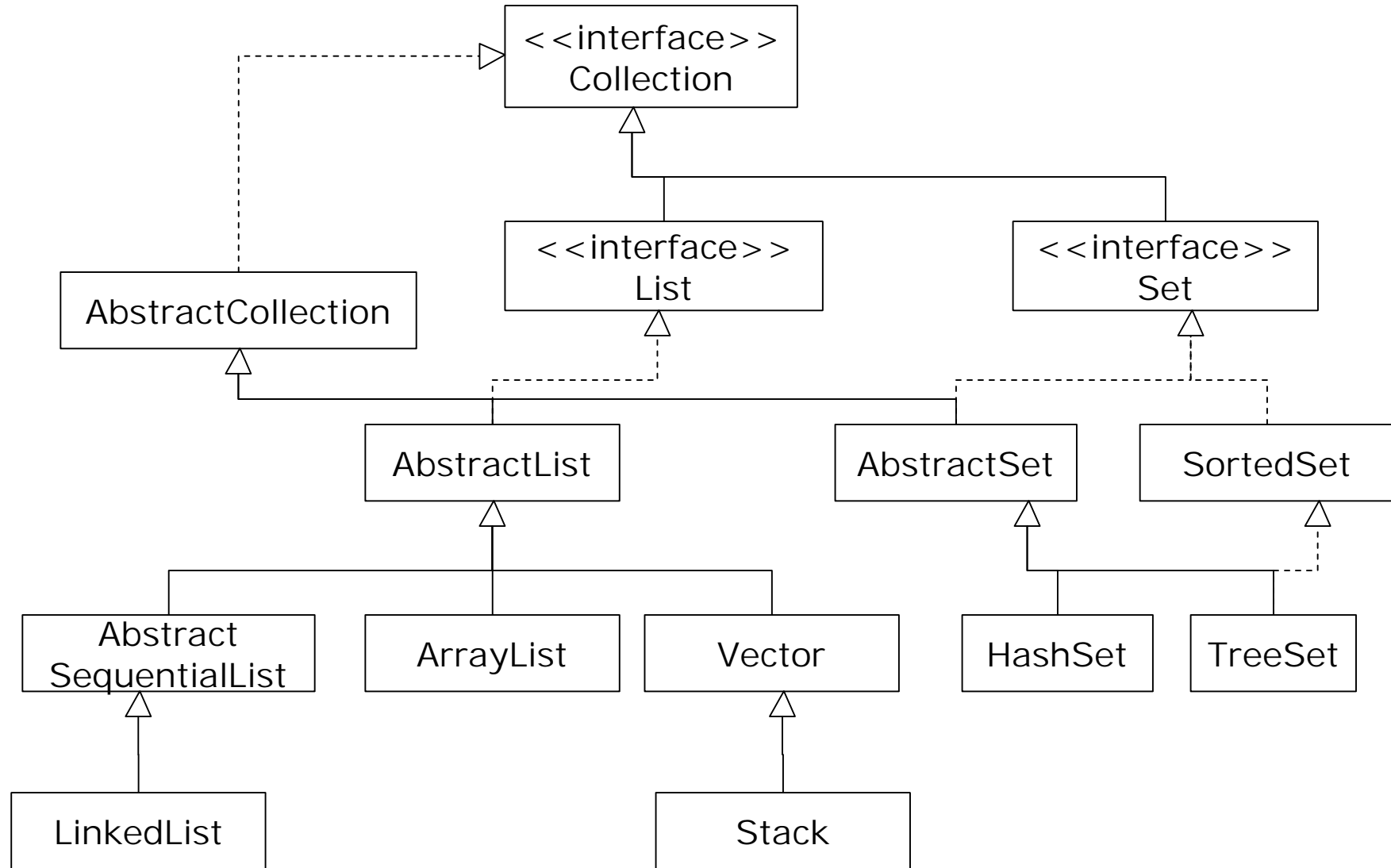
- o Flow of control does not leave the toolkit before it returns to your application.
- o Self-contained libraries
 - n Example: math functions in Jakarta commons-math

```
double[][] matrixData = { {1d,2d,3d}, {2d,5d,3d}};  
RealMatrix m = new RealMatrixImpl(matrixData);  
double[][] matrixData2 = { {1d,2d}, {2d,5d}, {1d,7d}};  
RealMatrix n = new RealMatrixImpl(matrixData2);  
  
RealMatrix p = m.multiply(n);  
RealMatrix pInverse = p.inverse();
```

General Collection Types

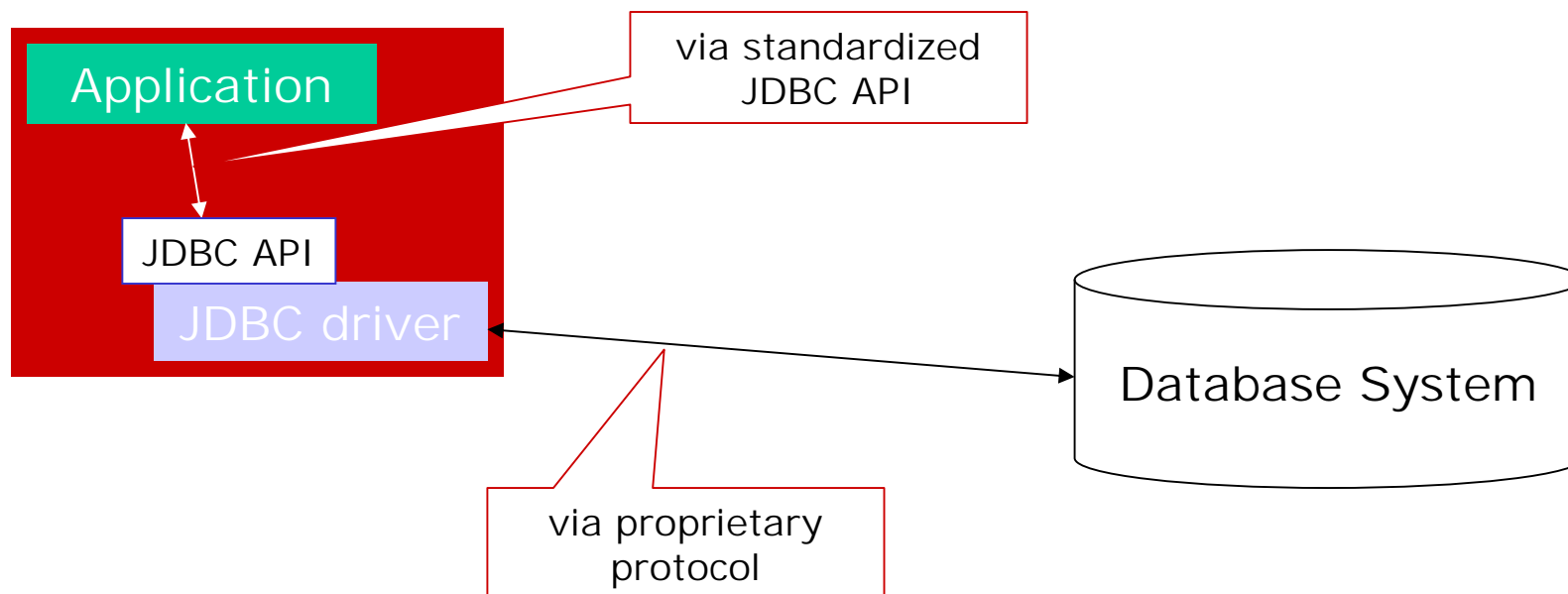
- o Collection
 - n anything that contains enumerable elements
 - n typical operations: add, iterate, contains, remove, size
- o List
 - n collection that contains indexed (i.e. numbered) elements
 - n additional typical operations: get(index), add(index)
- o Set
 - n collection that contains each element only once
- o Map
 - n not a collection
 - n contains pairs of elements: (key, value)
 - n operations: put(key, value), get(key), iterate (keys or values)

Java Collection Hierarchy



External System Access

- o Access to external system can work via toolkits.
 - n These toolkits often implement standardized APIs.
 - n Example: database access such as JDBC, ODBC, XMLDB
- o Flow of control usually leaves the toolkit before it returns to your application.



Frameworks

- o Definition: A framework is a set of cooperating classes that make up a reusable design for a specific class of software. These classes ...
 - n capture architectural and implementation artifacts that are invariant and
 - n defer the variant parts to application-specific logic,
 - n manage the control-flow,
 - n offer prefabricated parts as building blocks combined by design patterns.
- o Frameworks usually have reversed control-flow.

Design Patterns [...] capture the intent behind a design by identifying objects, their collaborations, and the distribution of responsibilities. [1]

Frameworks (2)

- o Frameworks interact with the application through well-defined connection points:
 - n events: application can register its interest in certain types of events with the framework. The framework then invokes the application's code for handling the event.
 - n hooks: application can inform the framework that it would like to run some of its code at a specific place in the control flow (hook-points). Again, the framework invokes the application.
 - n sub-classing: application sub-classes generic classes from the framework. The instances then behave in an application specific way.
- o Therefore, frameworks are often difficult to understand at first.
- o More in Software Architectures lecture.

Examples of Frameworks

- o GUI drawing frameworks:
 - n Java e.g.: Swing, SWT, JHotDraw
 - n Microsoft: MFC, ...
 - n Web Applications: Struts, Java Server Faces, Web Services, ...
- o XML parsers using the SAX (Simple API for XML)
- o Application servers: Run components

Components

- o Definition:
 - n “A component is a nontrivial, nearly independent, and replaceable part of a system that fulfills a clear function in the context of a well-defined architecture.
A component conforms to and provides the physical realization of a set of interfaces.”
[Philippe Krutchen, Rational Software]
- o Components do not have many outside dependencies
 - n ideally they have none
 - n the framework to run in is often the only dependency

Component-based Architecture

- o Assemble a whole software from components:
 - n Ideally, you should be able to put together your software from components with a bit of “glue code”.
 - n Often, you have to write very much “glue code”.
- o Application code interacts with components through well-defined interfaces
- o Components are often run inside an application server



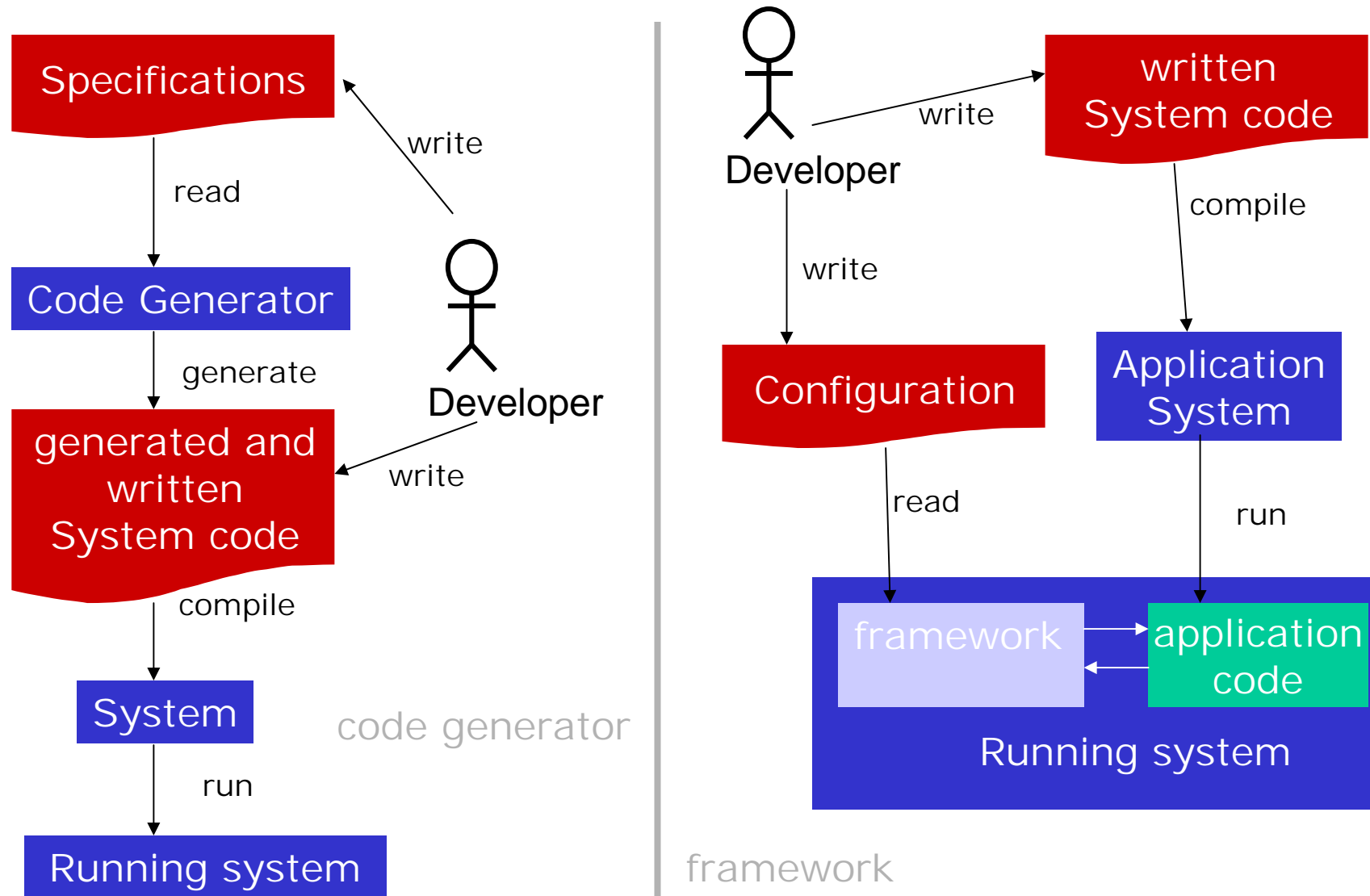
Component Models

- o Component models commonly specify:
 - n Functional boundaries that allow decoupling of collaborating object implementations
 - n A standard for an application server framework
 - n Configuration and deployment standards
- o Examples of component models:
 - n Microsoft's .NET
 - n Java EJB
 - n CORBA Component Model [3]

Code Generators

- o Definition: Code generators create source code for specialized systems. The code is tailored to the system's description, thus there is little configuration in the runtime system.
- o Code generators...
 - n can create parts of systems or even a whole system.
 - n can quickly mass-produce code that follows a certain pattern. They do the much of the boring work for you.
 - n cannot generate anything that does not follow those patterns.
- o Also see [2]

Code Generators vs. Frameworks



Examples Code Generators

- o XML loading/saving
 - n XMLBeans, JAXB
- o GUI modeling tools
 - n often built into IDEs (JBuilder, Visual Studio, NetBeans, ...)
- o (Pre-)Compiler
 - n IDL (CORBA)
- o Lexer/Parser Generator
 - n lex, yacc, bison, ANTLR, CUP
- o Systems generator
 - n CoCoMa model compiler

Discussion on Using Extensions

- o Use of extensions has to be considered in system design.
- o Using an existing extension is always* cheaper than recoding the functionality yourself.
 - n development
 - n maintenance
 - n training

* we are still waiting for a counter-example.



Writing Extensions

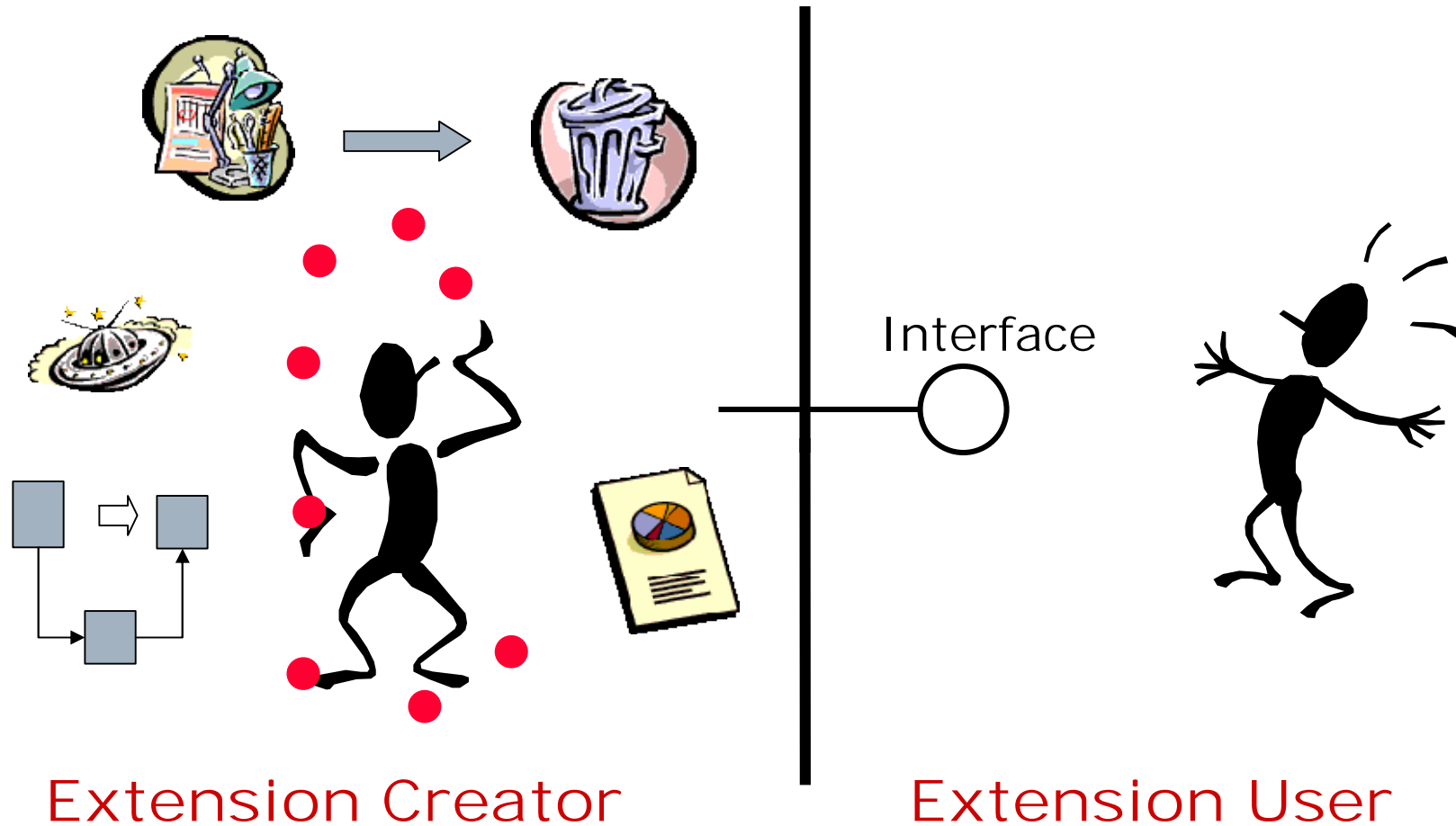
Writing Extensions

- o “When you're building a library, it's not enough to just accumulate good components.”
[Bertrand Meyer]
- o Extensions are built like software in general:
 - n well-defined development processes
 - n careful analysis
- o Extra aspects because extension are reusable:
 - n interfacing with client programmers
 - n consistency (ease of use)
 - n documentation

Interfaces

- o Client programmers of your extension want to incorporate it into their application.
- o They do not care how the internal implementation works, they just care about what the parts visible to the outside world are.
- o Therefore, you need to define your interfaces:
 - n Set of classes that allows use of your extension
 - n Documented on class-level
 - n There also must be higher level documentation so that programmers understand the general idea behind the extension

Interfaces (2)



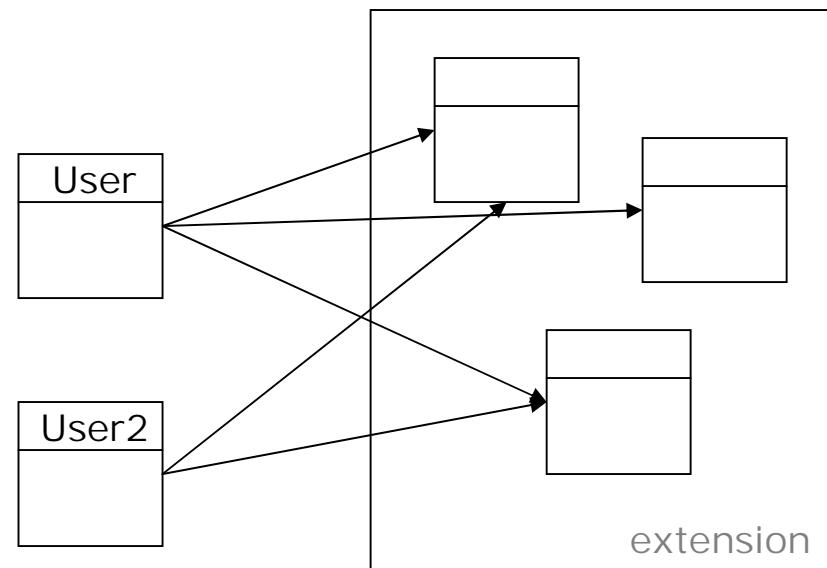
You have seen this before, right?
So guess how to implement it.

Design by Contract

- o A class (or extension) and its client have a contract with each other:
 - n client must guarantee preconditions before invocation of the extension
 - n extension must guarantee postconditions after its invocation
- o These conditions can be given in the code in some languages (e.g. Eiffel)
- o Contracts can be used...
 - n during development to understand what a piece of software does
 - n at runtime to detect errors
 - n for debugging to find the faulty piece of code

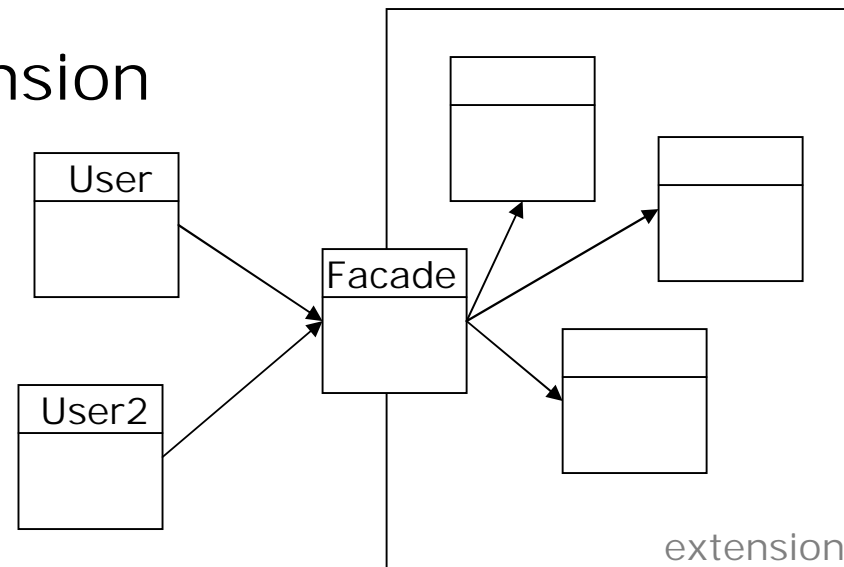
Extension Usage

- o Usage of any class from an extension can lead to problems:
 - n everybody depends on everything
 - n extension developers don't know what they may change
 - n application developers don't know what will remain stable
- o Result: extension becomes part of the application



Facade Pattern

- o Hide the complexity of your extension behind one (or a few) simple facade(s).
- o Facade is a design pattern.
- o Can help to ...
 - n implement design by contract
 - n define a useful interface
- o Makes using your extension much easier.



References

- o Articles:

- n [1] E. Gamma, R. Helm, R. Johnson, J. Vlissides. Design Patterns: Abstraction and Reuse of Object-Oriented Design. Proceedings of ECOOP '93, Kaiserslautern.
- n [2] M. Stevens. Automated Code Generation.
http://www.softwarereality.com/programming/code_generation.jsp
- n [3] D.C. Schmidt, S. Vinoski. The CORBA Component Model.
<http://www.cuj.com/documents/s=9039/cujexp0402vinoski/vinoski.htm>

- o Books:

- n E. Gamma, R. Helm, R. Johnson, J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley Longman (1998)

References (2)

- o Web:
 - n XMLBeans: <http://xmlbeans.apache.org/>
 - n JAXB: <http://java.sun.com/xml/jaxb/>
 - n CoCoMa System generator:
<http://www.sts.tu-harburg.de/~hw.sehring/cocoma/>
 - n Jakarta commons-math:
<http://jakarta.apache.org/commons/math/>