

---

# 08 - Object-Oriented Libraries and Extensions

# Motivation

---

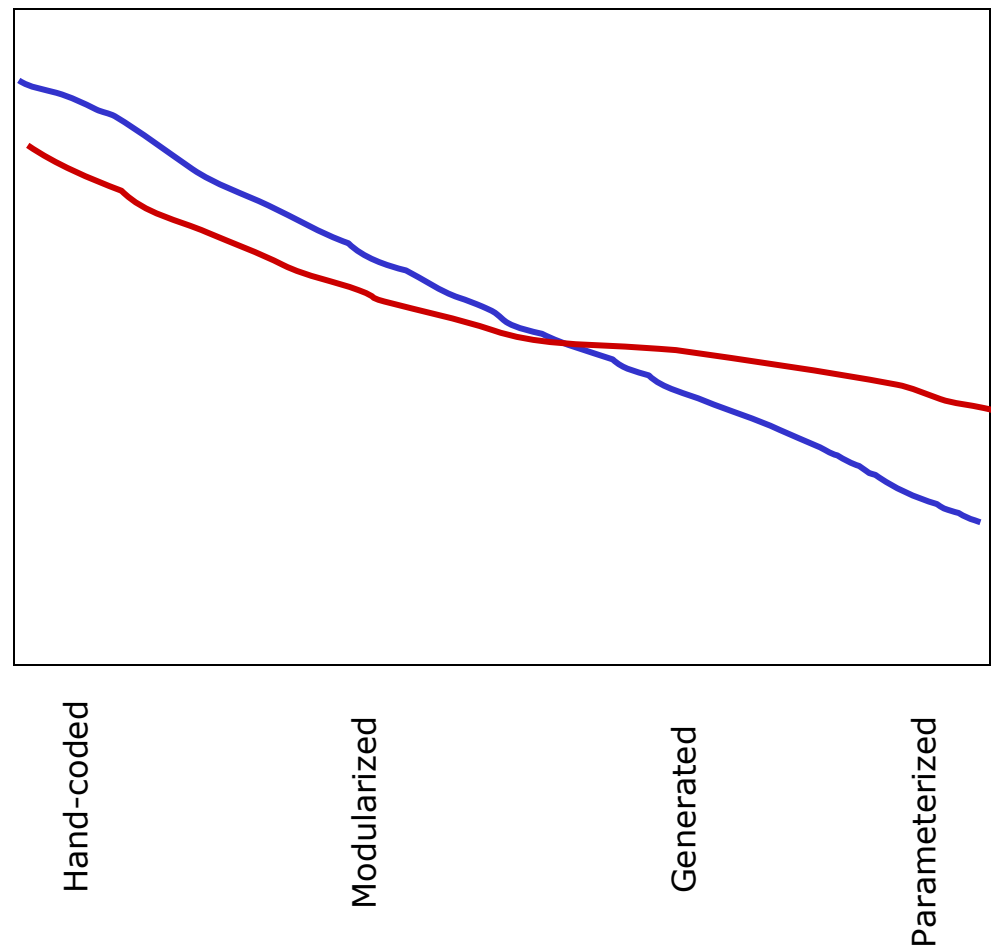
- When you write several systems, you notice that much of their code is similar.
- Stop reinventing the wheel, try to reuse code!
- How do you organize code reuse?  
History:
  - Copy & Paste
  - Collect useful files
  - Formal mechanisms to enable reuse across several people

# 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

---

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

---

- 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.
  - Usage is usually determined by API
  - Control-flow has to be managed by the programmer
  - Toolkits emphasize code reuse.
  - Offer isolated functionality
- Toolkits are in practice often referred to as libraries.
- Toolkits mostly fall in two categories:
  - interfaces to external systems
  - self-contained libraries of functionality

# Examples of Toolkits

---

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

# Independent Functionality

---

- Flow of control does not leave the toolkit before it returns to your application.
- Self-contained libraries
  - 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();
```

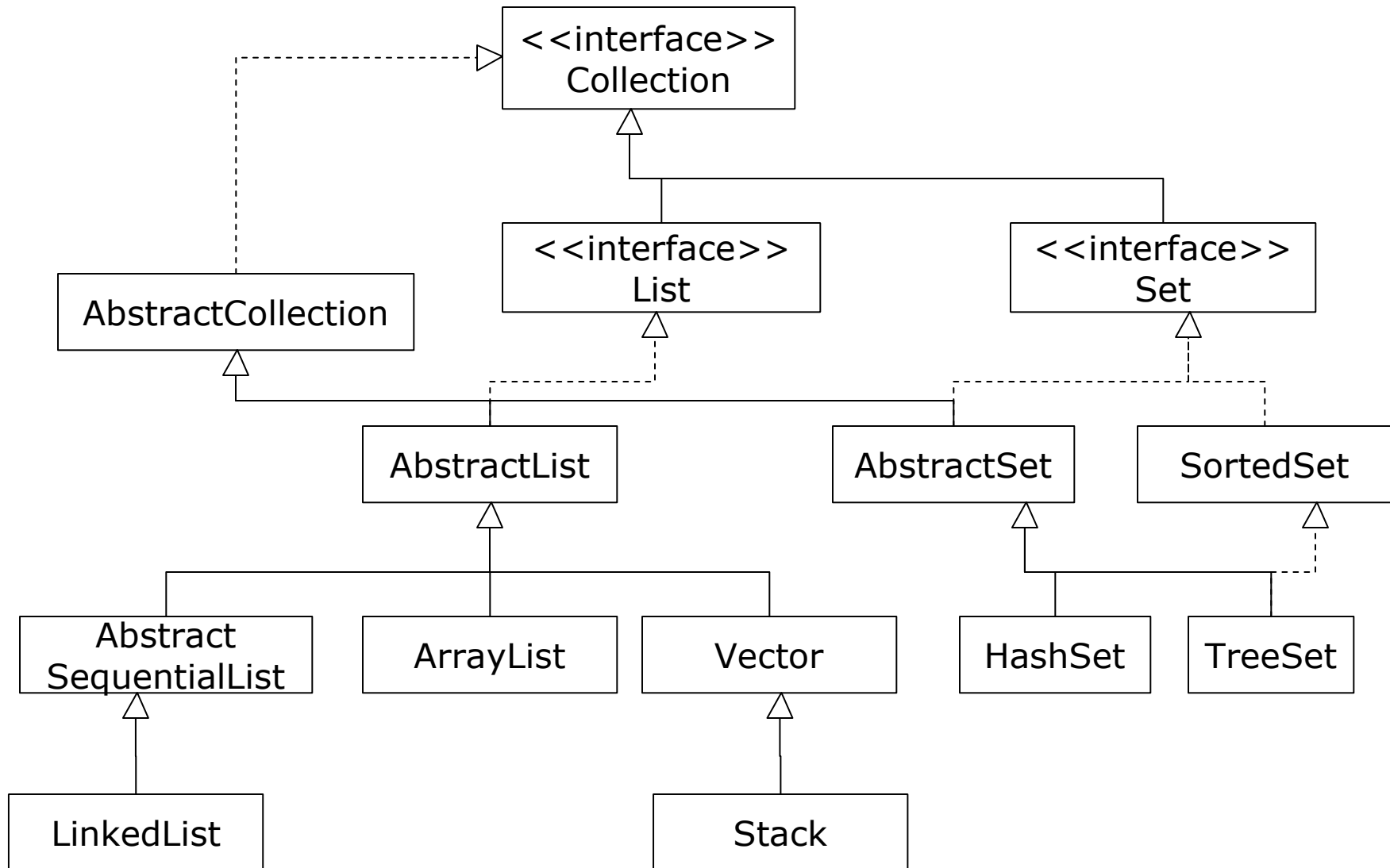
# General Collection Types

---

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

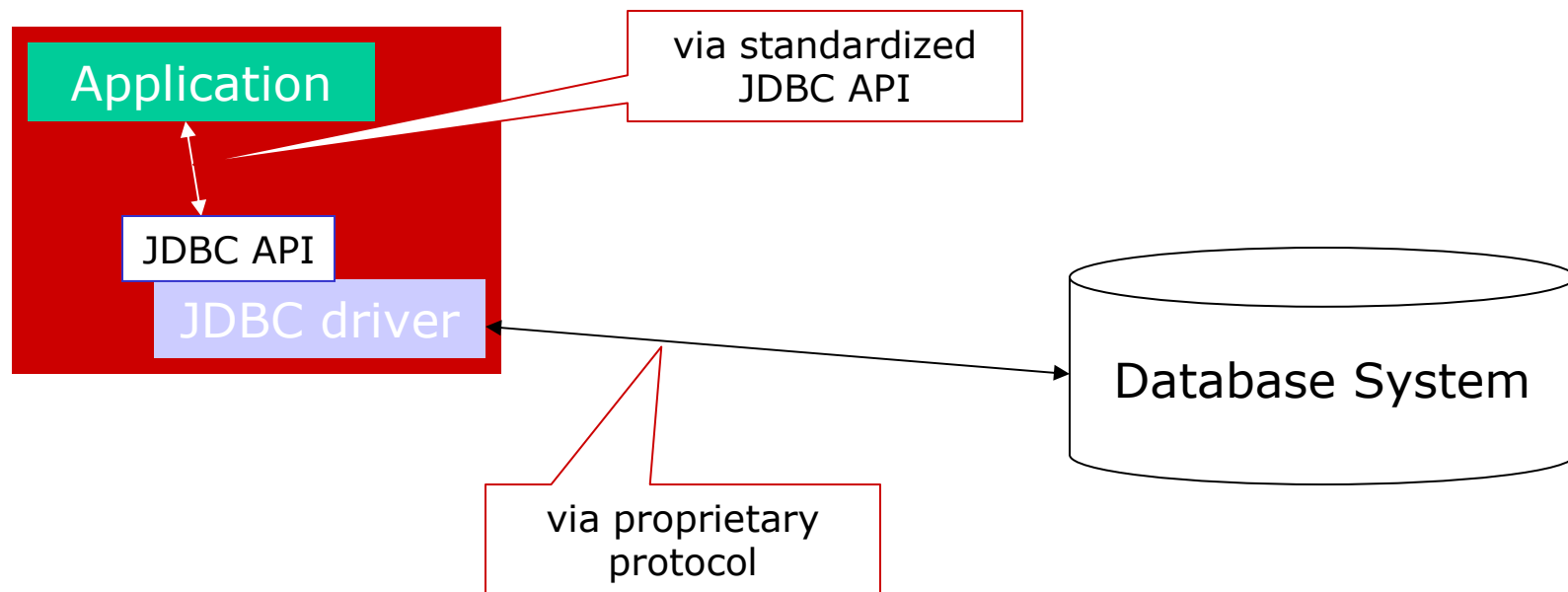
# Java Collection Hierarchy

---



# External System Access

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



# Frameworks

---

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

---

- Frameworks interact with the application through well-defined connection points:
  - 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.
  - 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.
  - sub-classing: application sub-classes generic classes from the framework. The instances then behave in an application specific way.
- Therefore, frameworks are often difficult to understand at first.
- More in Software Architectures lecture.

# Examples of Frameworks

---

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

# Components

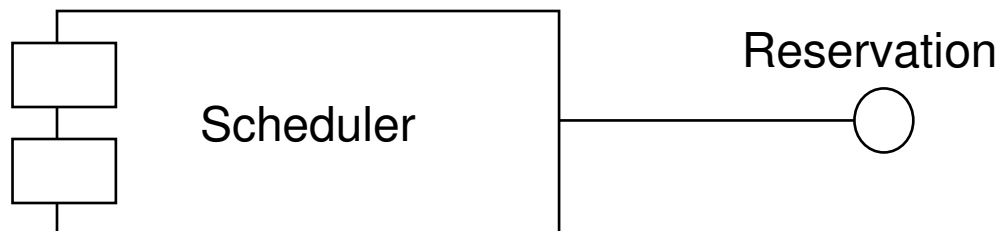
---

- Definition:
  - “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]
- Components do not have many outside dependencies
  - ideally they have none
  - the framework to run in is often the only dependency

# Component-based Architecture

---

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



# Component Models

---

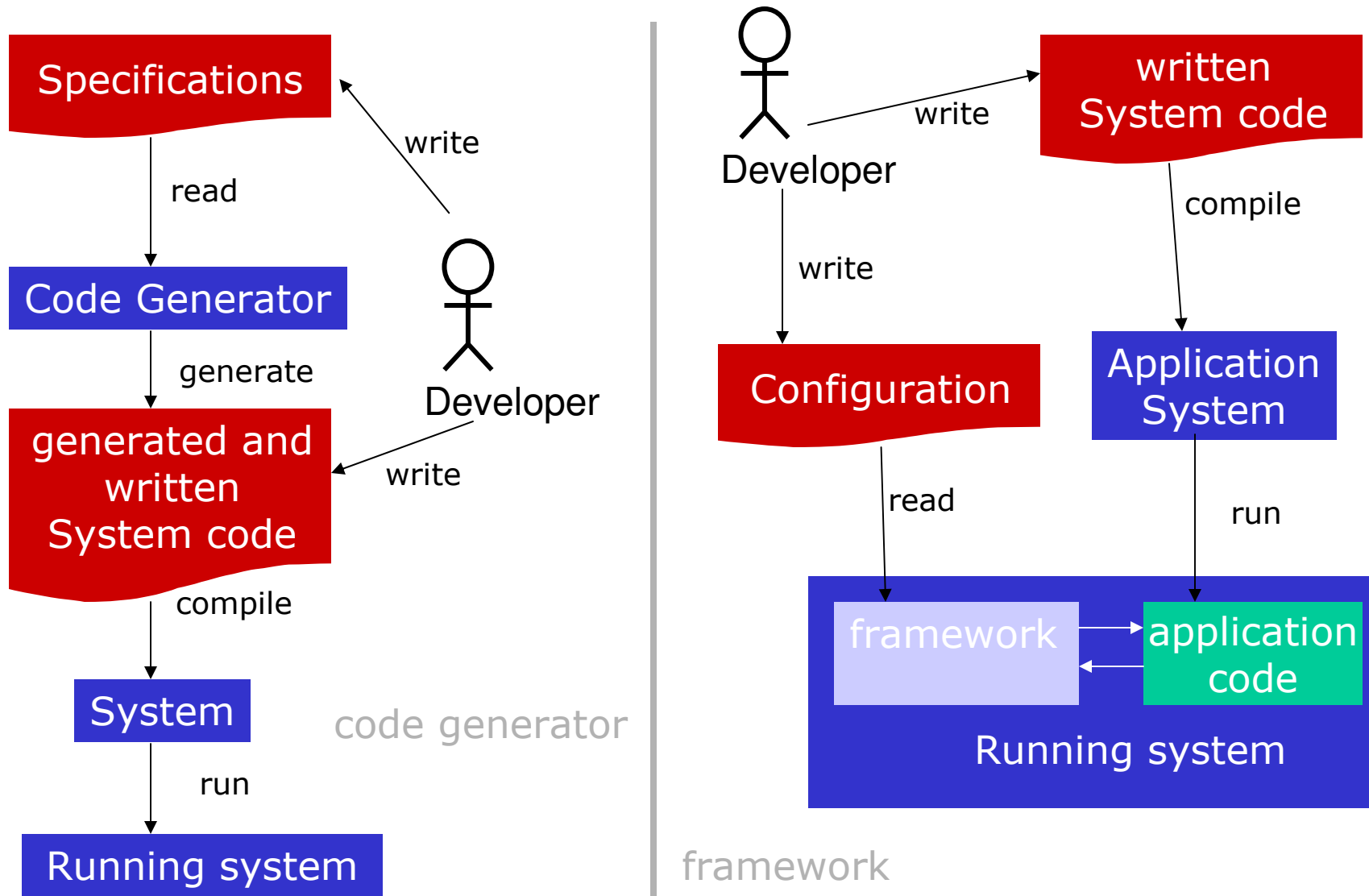
- Component models commonly specify:
  - Functional boundaries that allow decoupling of collaborating object implementations
  - A standard for an application server framework
  - Configuration and deployment standards
- Examples of component models:
  - Microsoft's .NET
  - Java EJB
  - CORBA Component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

# Code Generators

---

- 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.
- Code generators...
  - can create parts of systems or even a whole system.
  - can quickly mass-produce code that follows a certain pattern. They do the much of the boring work for you.
  - cannot generate anything that does not follow those patterns.
- Also see [2]

# Code Generators vs. Frameworks



# Examples Code Generators

---

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

# Discussion on Using Extensions

---

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

\* we are still waiting for a counter-example.

---

# Writing Extensions

# Writing Extensions

---

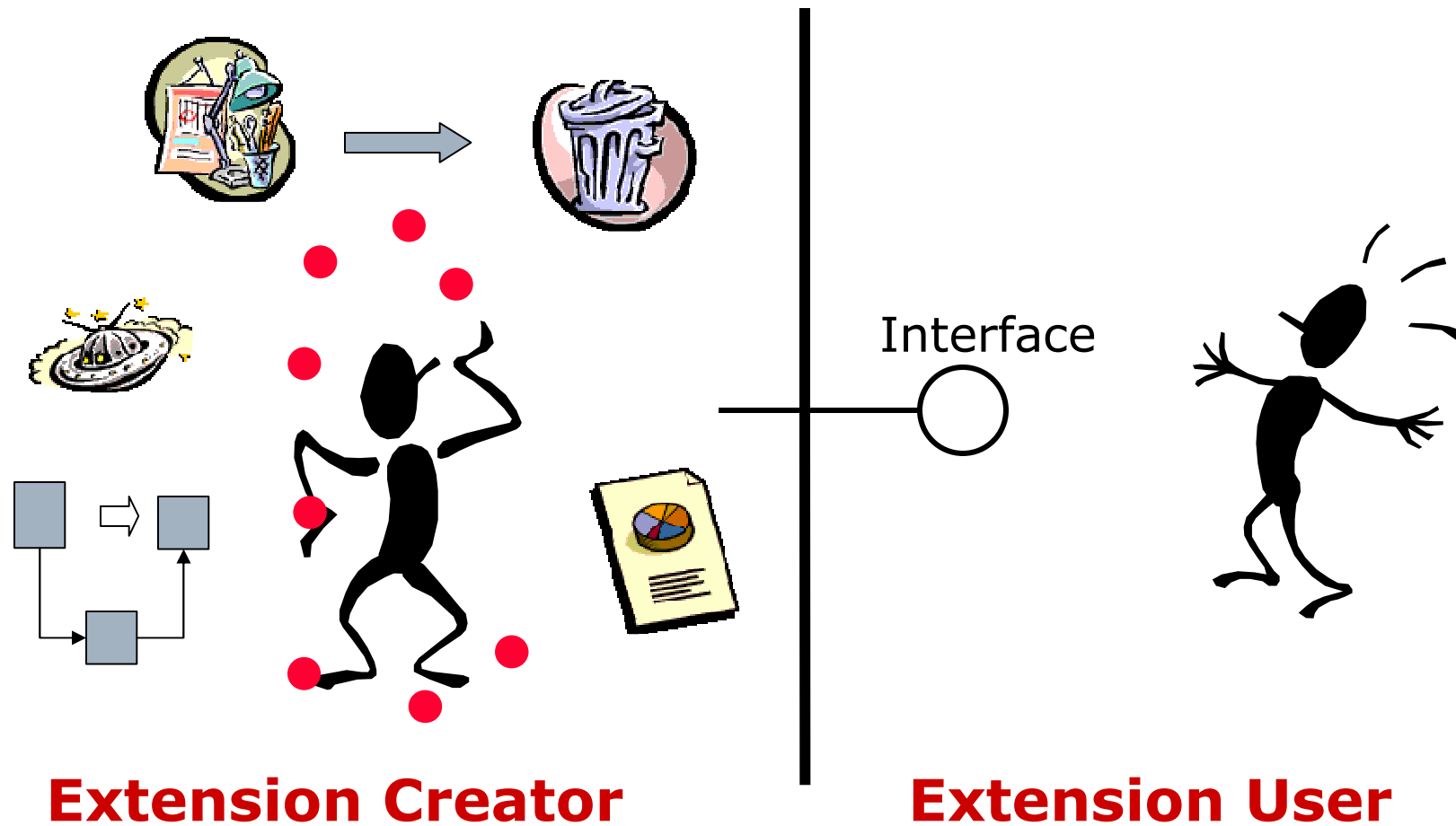
- “When you're building a library, it's not enough to just accumulate good components.”  
[Bertrand Meyer]
- Extensions are built like software in general:
  - well-defined development processes
  - careful analysis
- Extra aspects because extension are reusable:
  - interfacing with client programmers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

# Interfaces

---

- Client programmers of your extension want to incorporate it into their application.
- They do not care how the internal implementation works, they just care about what the parts visible to the outside world are.
- Therefore, you need to define your interfaces:
  - Set of classes that allows *use* of your extension
  - Documented on class-level
  - 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

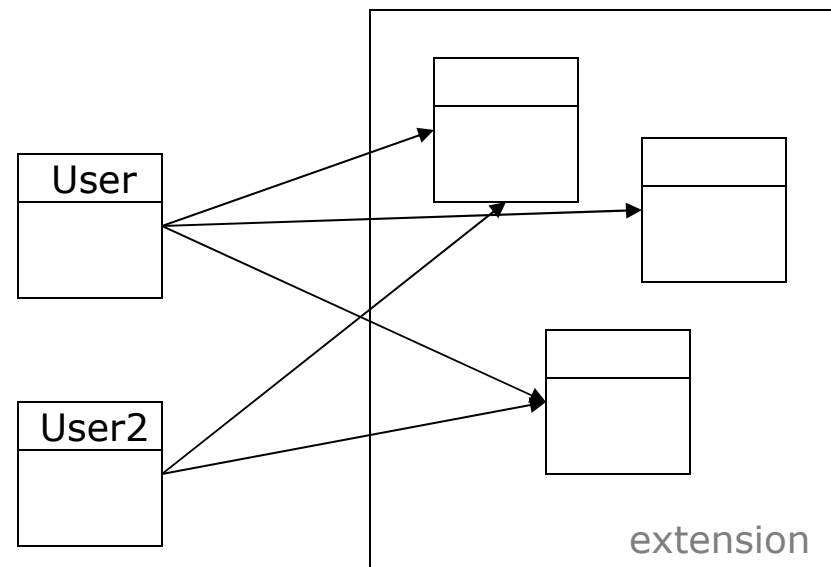
---

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

# Extension Usage

---

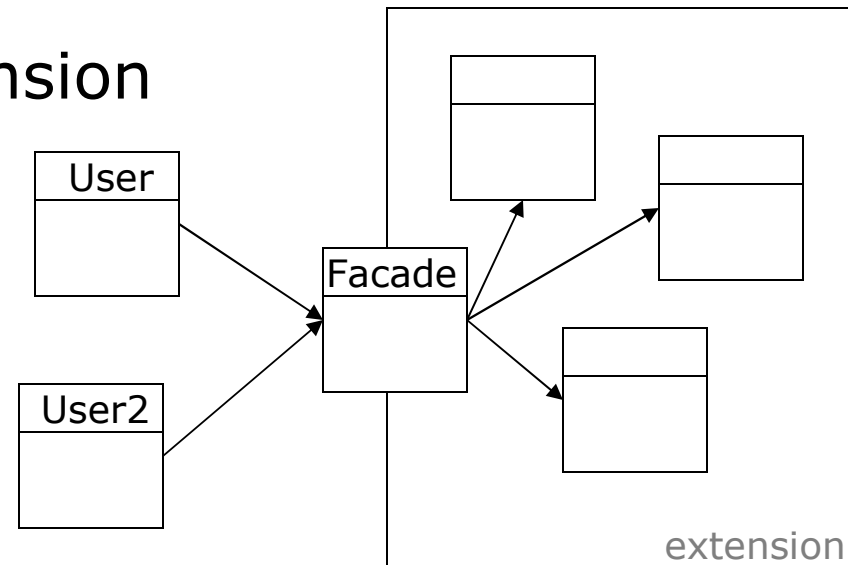
- Usage of any class from an extension can lead to problems:
  - everybody depends on everything
  - extension developers don't know what they may change
  - application developers don't know what will remain stable
- Result: extension becomes part of the application



# Facade Pattern

---

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





# References

---

## □ Articles:

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

## □ Books:

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

# References (2)

---

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