JGA: JAVA generic algorithms

Team 13
Adlan Israilov
Mohit Bhonde
Raul Vejar

What are functors? Why are they relevant?

- Common problem: how to pass “behaviour” as a parameter to a function?
- Specific problem: JAVA does not have function pointers or delegates like other languages
- Solution: Function objects, or objects built only to encapsulate a function or behavior = “Functors”
- Advantages:
  - Can be passed as an argument without being executed
  - Can be persisted
  - Can be serialized
  - Can be instantiated at runtime with runtime information
- Idea originally from C++ STL (Standard Template Library)

JGA: Objectives of the project

- Main objectives
  - Implement a library for JAVA to support and define functors
  - Implement functors described in the STL C++ library
  - Extend functors to be more “natural” for JAVA programmers
- Secondary objectives
  - Allow composition of functors
  - Take advantage of tree structure of composed functors and metadata
  - Implement “JAVA style” STL libraries using functors
  - Demonstrate usefulness of functors in event-driven programs like GUI like Swing programs

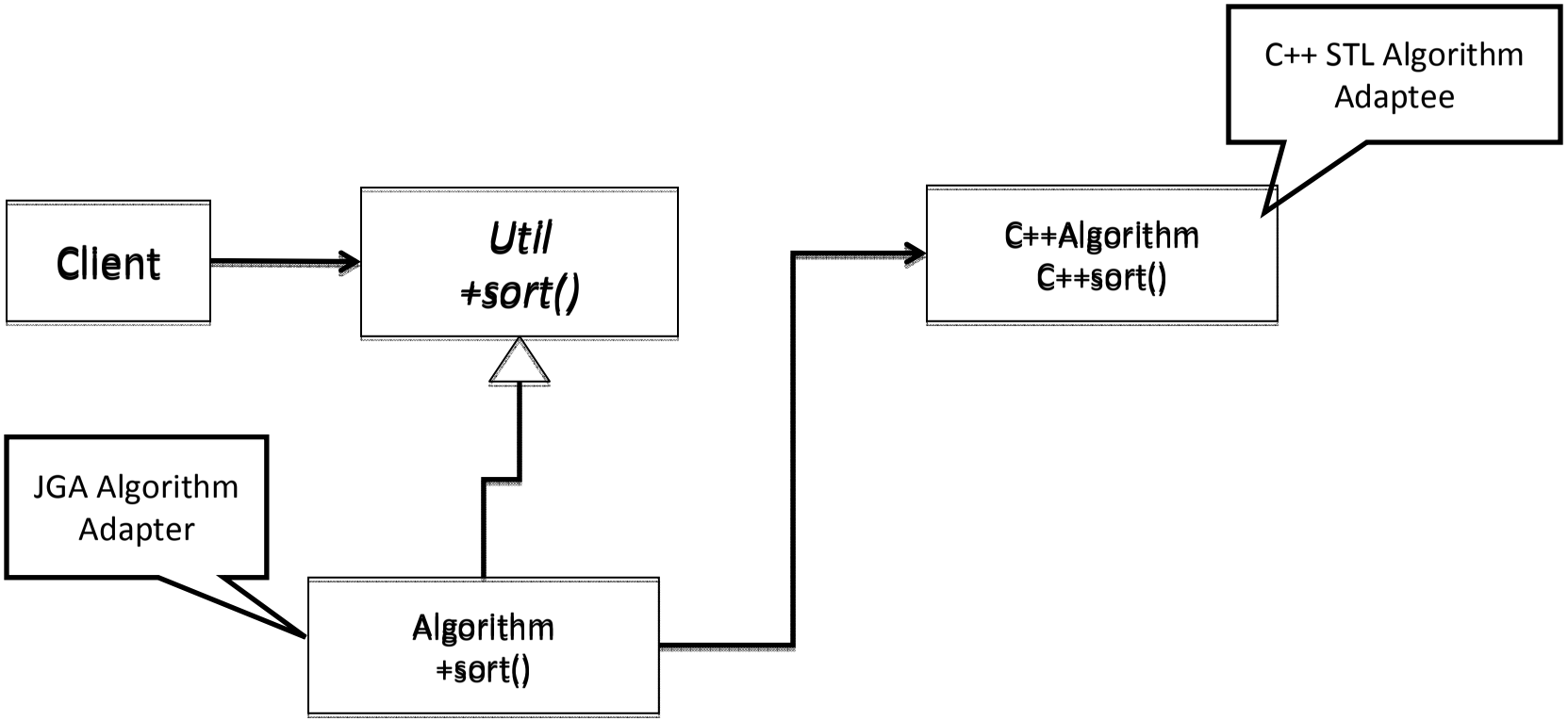
JGA: Project background

- Created 7 years ago
- Current version: 0.8 (released 2006-12-19)
- Compatible with JAVA 1.4 and 1.5
- Website: http://jga.sourceforge.net
- Distributed under two licenses: the CDDL and the LGPL
- Key quality attributes
  - Extensibility/Modifiability: Easy to extend, “template”
  - Usability: Easy to use for programmers

Adapter pattern

- Transforms similar functionality from one interface to another one
- Good when the implemented object is beyond your control
- The adapter class basically acts as a rapper to the different interface class by mapping all functionality to the new interface

The Adapter Pattern – Sort example



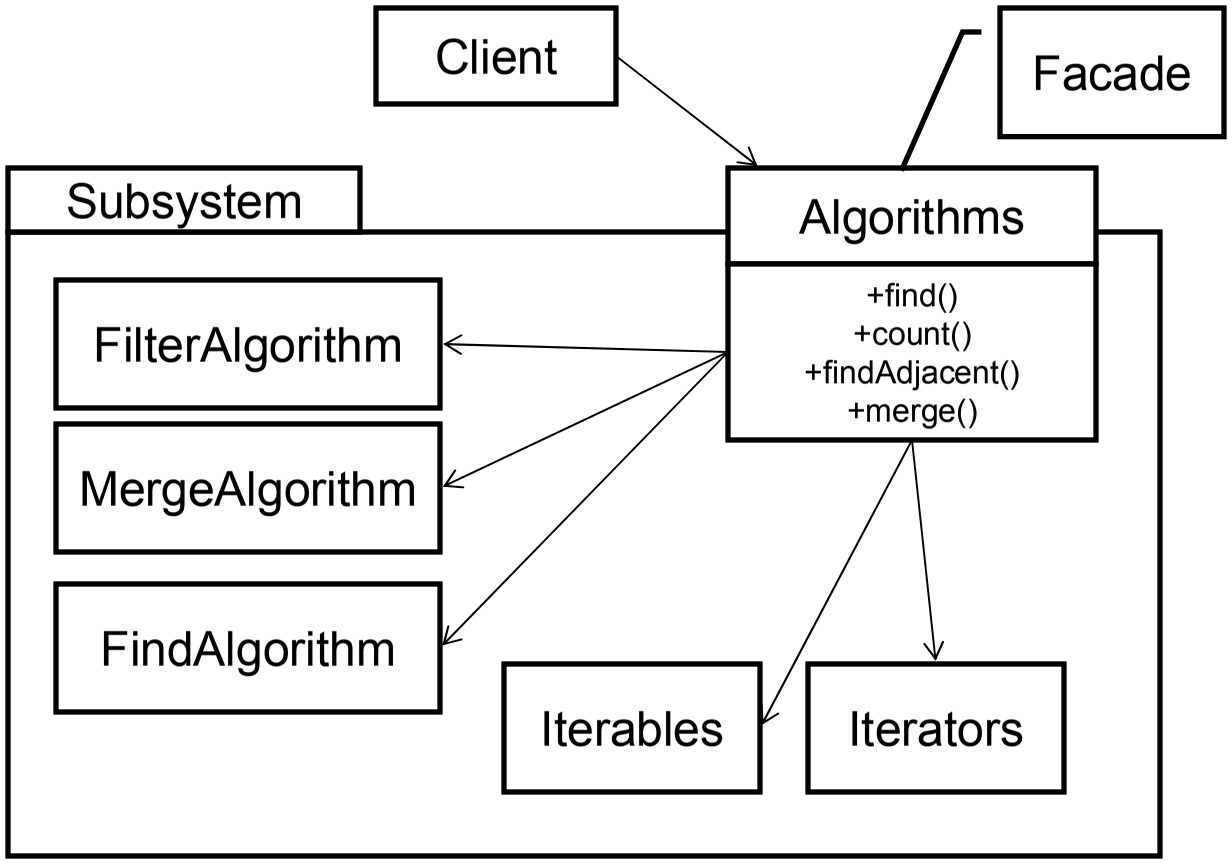
Adapter pattern in JGA

- Clients need java-like function calls, but JGA needs to adapt to C++ STL function calls which require function parameters
- Problem: C++ functions can make use of function pointers as parameters, java does not support function pointers
- Java makes use of Functors to adapt to C++ like calls => New interface
  - E.g. algorithms like sort, find, merge, etc
- Intermediate implementation in previous versions adapted calls to the STL C++ library to support gradual implementation but still full functionality
- http://jga.sourceforge.net/docs/javadoc/net/sf/jga/util/Algorithms.html

Façade Pattern

- Façade is a pattern used when we need to simplify the way how to use an existing complex system and we want to define our own interface.
- http://jga.sourceforge.net/docs/javadoc/net/sf/jga/util/Algorithms.html

Façade Pattern: UML diagram



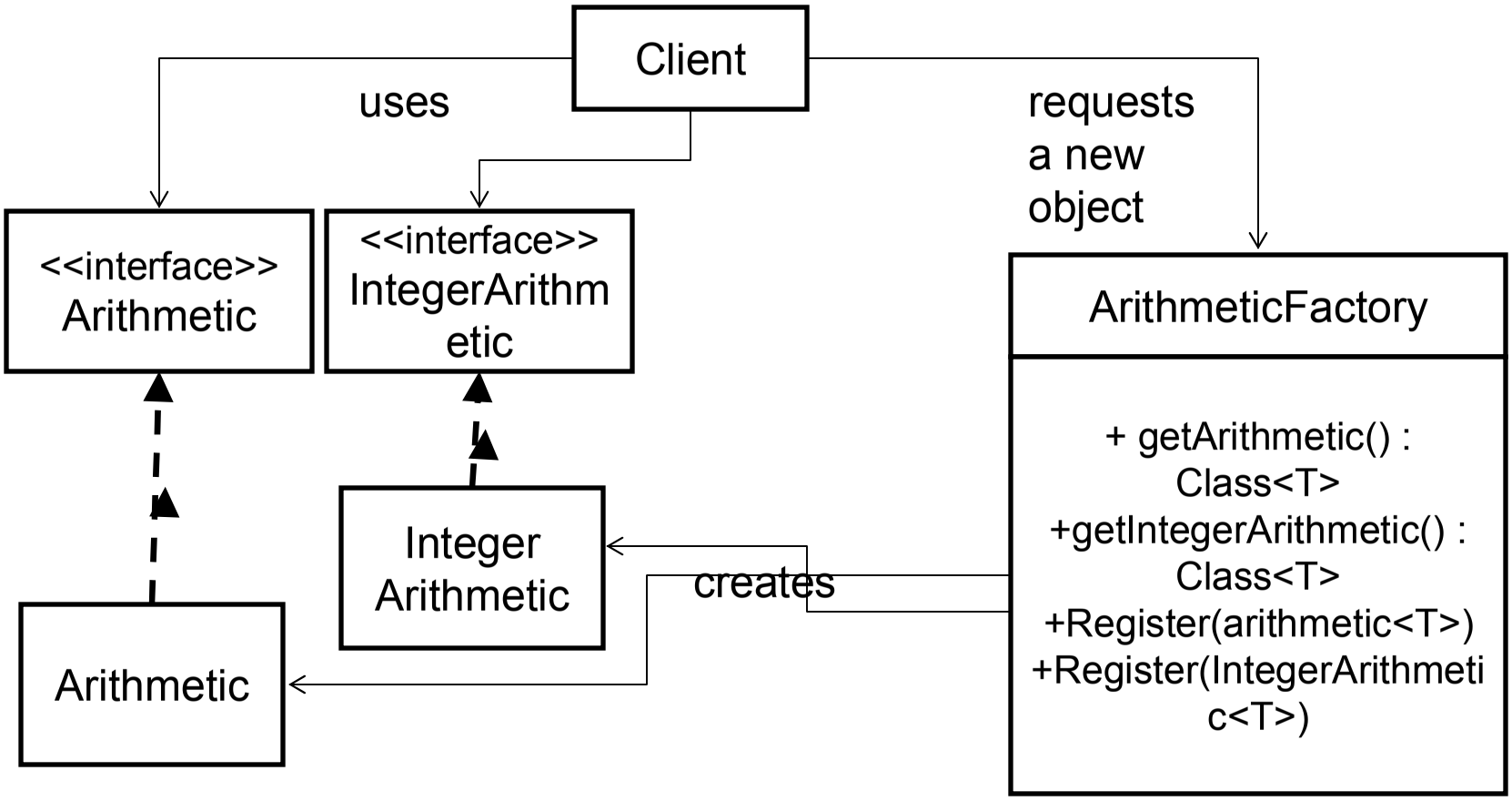
Façade Pattern: How it is used in JGA?

- In JGA the primary role of the Façade is to simplify access to the multiples classes making the java-version of the STL library
- JGA has a lot of functions, left alone there are more than 15 internal classes that support the STL implementation. The algorithms façade simplifies this and makes the library more usable for programmers.

Factory Pattern

- Defines an interface for creating an object, but let subclasses decide which class to instantiate. It lets class defer instantiation to subclasses.
- http://jga.sourceforge.net/docs/javadoc/net/sf/jga/fn/arithmetic/ArithmeticFactory.html
- http://jga.sourceforge.net/docs/javadoc/net/sf/jga/fn/arithmetic/Arithmetic.html
- http://jga.sourceforge.net/docs/javadoc/net/sf/jga/fn/arithmetic/IntegerArithmetic.html

Factory Pattern: UML diagram



Factory Pattern: How it is used in JGA?

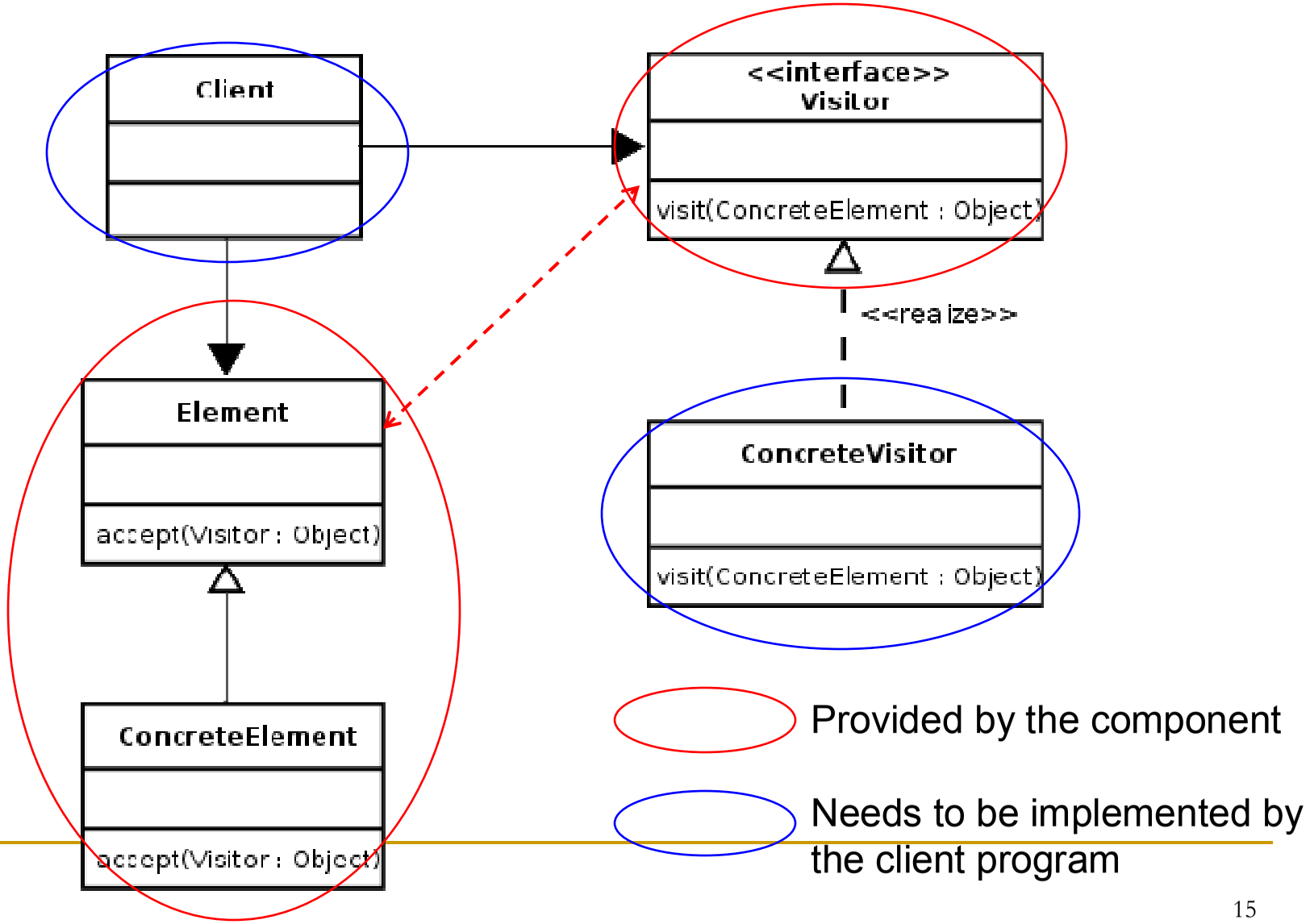
- Is responsible for generating objects (implementation) for two key numerical types using interfaces provided by the system (*Arithmetic and IntegerArithmetic)*.
- Focuses on modifiability and extensibility to facilitate adding new functors and new kind of calculation inside the library.
- Extensibility is supported by using the register function to add additional classes to the factory

Visitor Pattern: What is it?

- Used to add functionality to objects without changing their structure
- Similar to virtual functions, but:
  - Easier to implement
  - Can have internal state
  - Better support for design principles such as “open extension/close modifiable” and “separation of concerns”

From: http://en.wikipedia.org/wiki/Visitor_pattern

Visitor Pattern: What it looks like?



Visitor Pattern: How it is used in JGA?

- All functors implement acyclic visitor pattern: http://www.objectmentor.com/resources/articles/acv.pdf
- Uses multiple inheritance and dynamic cast to remove dependencies and improve extensibility: very important quality attribute for JGA
- Visitor is specially useful for hierarchic composed functors
- Examples
  - Functor
  - Binary Functor
  - Generator
  - Unary Functor
  - Etc (all others inherit from one of these basic functors)