Fostering Interoperability in Java-Based Computer Algebra Software

Heinz Kredel, University of Mannheim

FINA at AINA 2012, FIT Fukuoka

Overview

- Introduction
- Interfaces and classes
  - Apache Commons Math
  - JLinAlg
  - Java Algebra System
- Comparison
  - Proposal
- Conclusions

Introduction

- API design of Java libraries for symbolic and numeric computations
- requirements
  - separately compiled library
  - generic and object oriented
  - statically type safe
  - usable in parallel and distributed environments
- possible because JVM run-time with automatic garbage collection
- generic libraries : use data types and algorithms from other groups

Interoperability levels

- System level
  - OpenMath XML interfaces for monolithic systems (Maple, Mathematica, etc.)
- Scripting level
  - Sage a Python implementation of Magma
  - use C/C++ libraries of other CAS from Python
  - Singular, Pari, Gap, Kant, ...
- Library level
  - here Java libraries :
  - JAS, Apache commons Math, JLinAlg

Overview

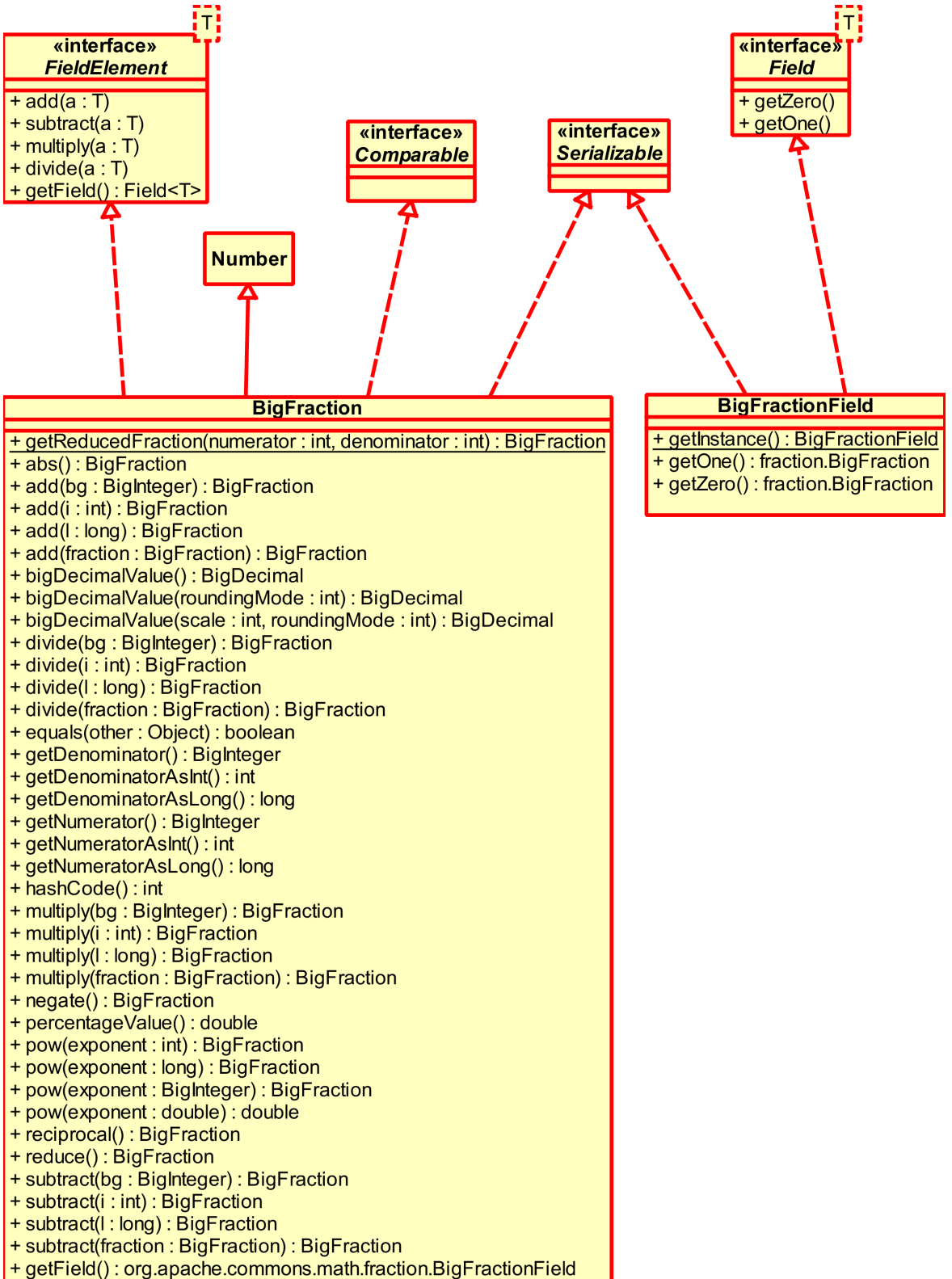
- Introduction
- Interfaces and classes
  - Apache Commons Math
  - JLinAlg
  - Java Algebra System
- Comparison
  - Proposal
- Conclusions

Interfaces and classes

- each library consists of a set of interfaces and implementing classes tailored to its focus
- here focus on rings and ring elements since common and central for interoperation
- common characteristics :
  - elements of algebraic structures
  - factories to create specific instances
  - agree on 3 of the library requirements
  - thread-safety requirement seems accepted
  - transportable objects (Serializable) not generally accepted

Apache Commons Math (1)

- focus on linear algebra
- central data type : fields for vector spaces
- interfaces : `Field` and `FieldElement`
- minimal set of methods for field elements
  - `add()`, `subtract()`, `multiply()` and `divide()`
- and for field factories
  - `getZero()` and `getOne()`
- type parameter <T> is not restricted


T
«interface»
FieldElement
+ add(a : T)
+ subtract(a : T)
+ multiply(a : T)
+ divide(a : T)
+ getField() : Field<T>
«interface»
Comparable
«interface»
Serializable
T
«interface»
Field
+ getZero()
+ getOne()
Number
BigFraction
+ getReducedFraction(numerator : int, denominator : int) : BigFraction
+ abs() : BigFraction
+ add(bg : BigInteger) : BigFraction
+ add(i : int) : BigFraction
+ add(l : long) : BigFraction
+ add(fraction : BigFraction) : BigFraction
+ bigDecimalValue() : BigDecimal
+ bigDecimalValue(roundingMode : int) : BigDecimal
+ bigDecimalValue(scale : int, roundingMode : int) : BigDecimal
+ divide(bg : BigInteger) : BigFraction
+ divide(i : int) : BigFraction
+ divide(l : long) : BigFraction
+ divide(fraction : BigFraction) : BigFraction
+ equals(other : Object) : boolean
+ getDenominator() : BigInteger
+ getDenominatorAsInt() : int
+ getDenominatorAsLong() : long
+ getNumerator() : BigInteger
+ getNumeratorAsInt() : int
+ getNumeratorAsLong() : long
+ hashCode() : int
+ multiply(bg : BigInteger) : BigFraction
+ multiply(i : int) : BigFraction
+ multiply(l : long) : BigFraction
+ multiply(fraction : BigFraction) : BigFraction
+ negate() : BigFraction
+ percentageValue() : double
+ pow(exponent : int) : BigFraction
+ pow(exponent : long) : BigFraction
+ pow(exponent : BigInteger) : BigFraction
+ pow(exponent : double) : double
+ reciprocal() : BigFraction
+ reduce() : BigFraction
+ subtract(bg : BigInteger) : BigFraction
+ subtract(i : int) : BigFraction
+ subtract(l : long) : BigFraction
+ subtract(fraction : BigFraction) : BigFraction
+ getField() : org.apache.commons.math.fraction.BigFractionField
BigFractionField
+ getInstance() : BigFractionField
+ getOne() : fraction.BigFraction
+ getZero() : fraction.BigFraction

Apache Commons Math (2)

- implementing classes, for example rational numbers
  - `BigFraction` and `BigFractionField`
- implement additionaly
  - `Serializable and Comparable`
- and extend the class `Number`
  - mandate conversion methods like `intValue()`
- interface methods four times overloaded
  - for the class itself, for `BigInteger`
  - and for the primitive types `int` and `long`

Apache Commons Math (3)

- overloaded methods not reflected in the interface
- `negate(), abs(), pow()` not defined in the interface
- conversion methods `bigDecimal(),` could also go to an interface
- methods related to rational numbers `getDenominator()` and `getNumerator()`

JLinAlg (1)

- focus on linear algebra
- central data type : modules over rings
- interfaces : `IRingElement` and `IRingElementFactory`
- methods for ring elements
  - `add(), subtract(), multiply(), divide(), inverse(), negate(), abs()`
  - `isZero(), isOne()`
  - `lt(), gt(), le(), ge()`
  - `norm(), apply()`


RE
«interface»
IRingElement
+ isZero() : boolean
+ add(other : RE)
+ subtract(val : RE)
+ negate()
+ isOne() : boolean
+ multiply(other : RE)
+ equals(obj : Object) : boolean
+ compareTo(o : RE) : int
+ apply(fun : MonadicOperator<RE>)
+ abs()
+ norm()
+ lt(val : RE) : boolean
+ gt(val : RE) : boolean
+ le(val : RE) : boolean
+ ge(val : RE) : boolean
+ invert()
+ divide(val : RE)
+ getFactory()
RE
«interface»
IRingElementFactory
+ getArray(size : int) : RE[]
+ getArray(rows : int, columns : int) : RE[][]
+ one()
+ zero()
+ m_one()
+ get(o : Object)
+ get(i : int)
+ get(d : long)
+ get(d : double)
+ gaussianRandomValue(random : Random)
+ randomValue(random : Random)
+ randomValue(min : RE, max : RE)
+ gaussianRandomValue()
+ randomValue()
+ randomValue(random : Random, min : RE, max : RE)
+ convert(from : Vector<?extends IRingElement>) : Vector
+ convert(from : Matrix<?extends IRingElement>) : Matrix
RE
RingElement
+ isZero() : boolean
+ subtract(val : RE)
+ isOne() : boolean
+ equals(obj : Object) : boolean
+ apply(fun : MonadicOperator<RE>)
+ norm()
+ lt(val : RE) : boolean
+ gt(val : RE) : boolean
+ le(val : RE) : boolean
+ ge(val : RE) : boolean
+ divide(SuppressWarnings : @)
+ invert()
RE
RingElementFactory
+ getArray(size : int) : RE[]
+ getArray(rows : int, columns : int) : RE[][]
+ one()
+ zero()
+ m_one()
+ get(o : Object)
+ get(i : int)
+ get(d : double)
+ gaussianRandomValue(random : Random)
+ randomValue(random : Random)
+ randomValue(min : RE, max : RE)
+ randomValue(random : Random, min : RE, max : RE)
+ convert(from : Matrix<?extendsIRingElement>) : Matrix
+ convert(from : Vector<?extendsIRingElement>) : Vector

JlinAlg (2)

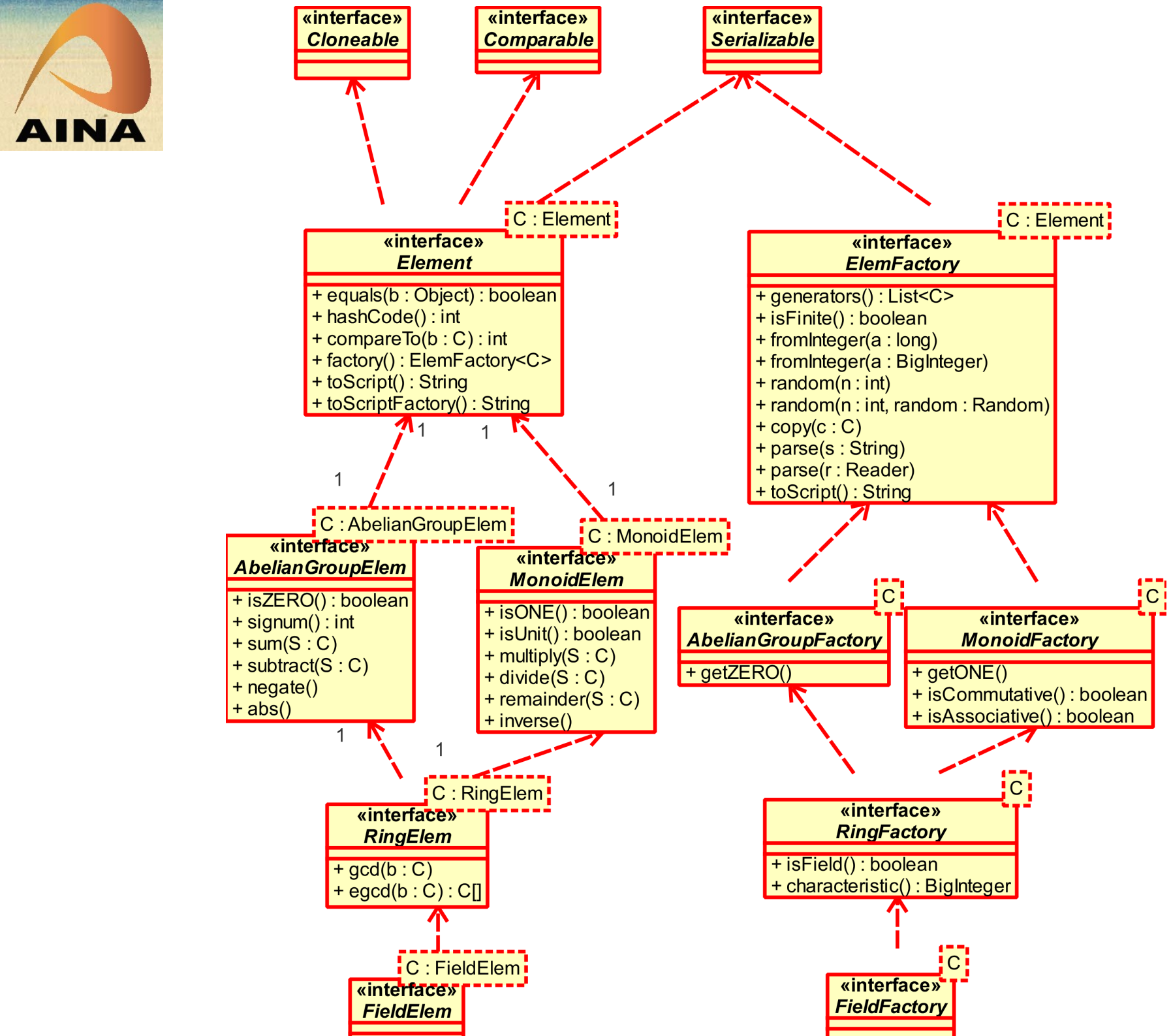
- and for ring factories
  - `zero()` and `one(), m_one()`
  - `randomValue(), gaussianRandomValue()`
  - conversion methods from other types : `get()`
  - construct arrays : `getArray()`
  - convert between vectors and matrices
- type parameter <RE> is restricted to `IRingElement`

JLinAlg (3)

- abstract classes `RingElement`, `RingElementFactory`
- implement `subtract()` in terms of `negate()` and `add()`
- implementations `divide()` and `inverse()` throw exceptions if not overwritten
- `get()` is implemented using conversion with String representations

Java Algebra System, JAS (1)

- focus on (non-linear) algebra
- central data type : polynomials over rings
- interfaces : `RingElem` and `RingFactory`
  - composed from `AbelianGroupElem` and `MonoidElem`
  - both in turn composed from `Element`
- `Element`
  - extends `Clonable, Comparable, Serializable`
  - defines `factory(), toScript()`


«interface»
Cloneable
«interface»
Comparable
«interface»
Serializable
C : Element
«interface»
Element
+ equals(b : Object) : boolean
+ hashCode() : int
+ compareTo(b : C) : int
+ factory() : ElemFactory<C>
+ toScript() : String
+ toScriptFactory() : String
C : Element
«interface»
ElemFactory
+ generators() : List<C>
+ isFinite() : boolean
+ fromInteger(a : long)
+ fromInteger(a : BigInteger)
+ random(n : int)
+ random(n : int, random : Random)
+ copy(c : C)
+ parse(s : String)
+ parse(r : Reader)
+ toScript() : String
1
1
1
1
C : AbelianGroupElem
«interface»
AbelianGroupElem
+ isZERO() : boolean
+ signum() : int
+ sum(S : C)
+ subtract(S : C)
+ negate()
+ abs()
C : MonoidElem
«interface»
MonoidElem
+ isONE() : boolean
+ isUnit() : boolean
+ multiply(S : C)
+ divide(S : C)
+ remainder(S : C)
+ inverse()
C
«interface»
AbelianGroupFactory
+ getZERO()
C
«interface»
MonoidFactory
+ getONE()
+ isCommutative() : boolean
+ isAssociative() : boolean
1
1
C : RingElem
«interface»
RingElem
+ gcd(b : C)
+ egcd(b : C) : C[]
C
«interface»
RingFactory
+ isField() : boolean
+ characteristic() : BigInteger
C : FieldElem
«interface»
FieldElem
C
«interface»
FieldFactory

JAS (2)

- `AbelianGroupElem`
  - `sum(), subtract(), negate(), abs()`
  - `isZERO(), signum()`
- `MonoidElem`
  - `multiply(), divide(), inverse(), remainder()`
  - `isONE(), isUnit()`
- `RingElem` adds
  - `gcd(), egcd()`
- `FieldElem` no further methods

JAS (3)

- `ElementFactory` defines
  - conversion : `fromInteger(), parse()`
  - construction : `random(), generators()`
  - predicate : `isFinite()`
- `AbelianGroupFactory` defines
  - `getZERO()`
- `MonoidFactory` defines
  - `getONE()`
  - `isCommuntative(), isAssociative()`

JAS (4)

- `RingFactory` defines
  - `isField()`
  - `characteristic()`
- `FieldFactory` no further methods
- type parameter <C> is restricted to respective interface

Overview

- Introduction
- Interfaces and classes
  - Apache Commons Math
  - JLinAlg
  - Java Algebra System
- Comparison
  - Proposal
- Conclusions

Comparison (1)

- all provide generic algebraic objects and algorithms for computation with them
- implemented using Java 5 type parameters
- basic design similar
  - split between elements and factories
  - factories to create elements
  - agreement on 3 of the library requirements
  - thread-safety requirement seems accepted
  - Serializable not generally accepted
- comprehensive : JAS > JLinAlg > AC Math

Comparison (2)

- different goals :
  - ACMath : linear algebra over commutative fields of characteristic 0, numeric computations with rounding errors
  - JLinAlg : linear algebra over fields of arbitrary characteristic, also numeric objects
  - JAS : more general algebraic structures like commutative and non-commutative (non-linear) algebras, arbitrary characteristic, mostly exactly represented objects, few numeric objects

Comparison (3)

- trade-offs
  - many methods in interfaces →
    - more implementations required
  - to few methods in interfaces →
    - many case distinctions in usage
    - generic design limited or impossible
  - thread-safety
    - design immutable objects
    - or maintain method synchronization
  - transport, distributed computing
    - maintain object serialization
- extra unit tests required and to be maintained

Comparison (4)

- Note : `add()` versus `sum()`
  - mutable in Java collections framework
  - need immutable for parallel usage
  - problem of confusion, so different names
- JAS started with a smaller set of defined methods in the interfaces
- current set of methods proven to be required in implementation of large parts of (polynomial) algebras / rings

Comparison (5)

- need to distinguish :
  - finite and infinite fields of finite characteristic
  - `isFinite()` and `characteristic()`
- required in generic algorithms :
  - `isCommutative()` and `isAssociative()`
  - `isField()`
- conversion methods :
  - `fromInteger(), parse()`
  - eventually more general `valueOf()`

Comparison (6)

- for distributed algorithms :
  - need `Serializable`
- for interoperation with Java collections :
  - `Comparable`
  - `Clonable`
- interoperation using adapter classes :
  - needs two adaptors for each pair of libraries
  - does not scale well to more libraries
  - run-time overhead using delegation

Proposal

- use revised interfaces from JAS as basis
  - check flat versus structured interfaces
  - burden to implement more methods and tests
    - only three predicates besides arithmetic
  - check where to place scripting methods, not useful in ACMath
    - toScript() in Element
  - will need some time
- make them available under Apache Commons Math and Apache licence

State of the cooperation

- contact with ACMath via mailing list
- offered proposal and explained questions
- ACMath now preparing for release 3.0
- then think about the interfaces
- no response from JLinAlg developers

Conclusions

- studied three interfaces
- not so different in concepts
- different number of methods
- different emphasis of interfaces vs. (abstract) classes
- will need some time to sort issues out
- defined a useful subset of methods for interoperation in a future standard

Thank you for your attention

Questions ?

Comments ?

http://krum.rz.uni-mannheim.de/jas/

http://jscl-meditor.sourceforge.net/

Acknowledgements

thanks to: Raphael Jolly, Apache Commons Math developers, JlinAlg developers, Thomas Becker, Werner K. Seiler, Axel Kramer, Dongming Wang, Thomas Sturm, Hans-Günther Kruse, Markus Aleksy

thanks to the referees