Course on SPIS Numerical core

(or SPIS/NUM, or "the solvers")

J.-F. Roussel, F. Rogier, D. Volpert, *ONERA / DESP*

Outline

- Object Oriented Programming concepts (OOP)
- SPIS/NUM architecture:
  - UML diagram (Universal Modelling Language)
  - JavadDoc browsing
  - Eclipse browsing
- Introduction to Java OO language
- A Java IDE: Eclipse
- SPIS/NUM top level objects and examples

Object Oriented Programming concepts (OOP) 1/4

- An **object** contains 3 types of members:
  - ***Fields*** (or data, variables…)
  - ***Constructor(s)*** (~ initialiser)
  - ***Methods*** (~ subroutines)
- It is "self-contained":
  - It is initialised by a constructor (mandatory to build an object)
  - Methods act mostly on object own fields (they often have no or few parameters)
- Example, Electric Field object:
  - It contains (fields):
    - Field values (a VolumeField in SPIS)
    - A Poisson solver (it is self-contained)

    they are defined in the constructor: `new PoissonSolver(fieldValues, poissonSolver);`
  - Methods:
    - solve() (no parameters needed)
    - getPotential()

ONERA

Object Oriented Programming concepts (OOP) 2/4

- A distinction must be done between:
  - The ***class*** ~ a type of object, a model (type in Fortran 90)
  - The ***instance(s)***: objects following that model (variables of that type Fortran 90)

- Example:
  - The class: a PIC (Particle-In-Cell) Volume Distribution (distribution represented by Monte-Carlo sampling)
  - Several instances possible:
    - ions = new PicVolDistrib(ionType, etc.);
    - Electrons = new PicVolDistrib(…);

OOP concepts: Heritage 3/4

- **Heritage** :
  - From a class, other classes can be derived
  - They have the parent-class members + others to be defined
  - Derived classes can have several interest: enrichment, specialisation

- Examples:
  - ParticleList has all particles similar (same type & weight) ← RichParticleList has different particles (involves extra tables for particles types and weights)

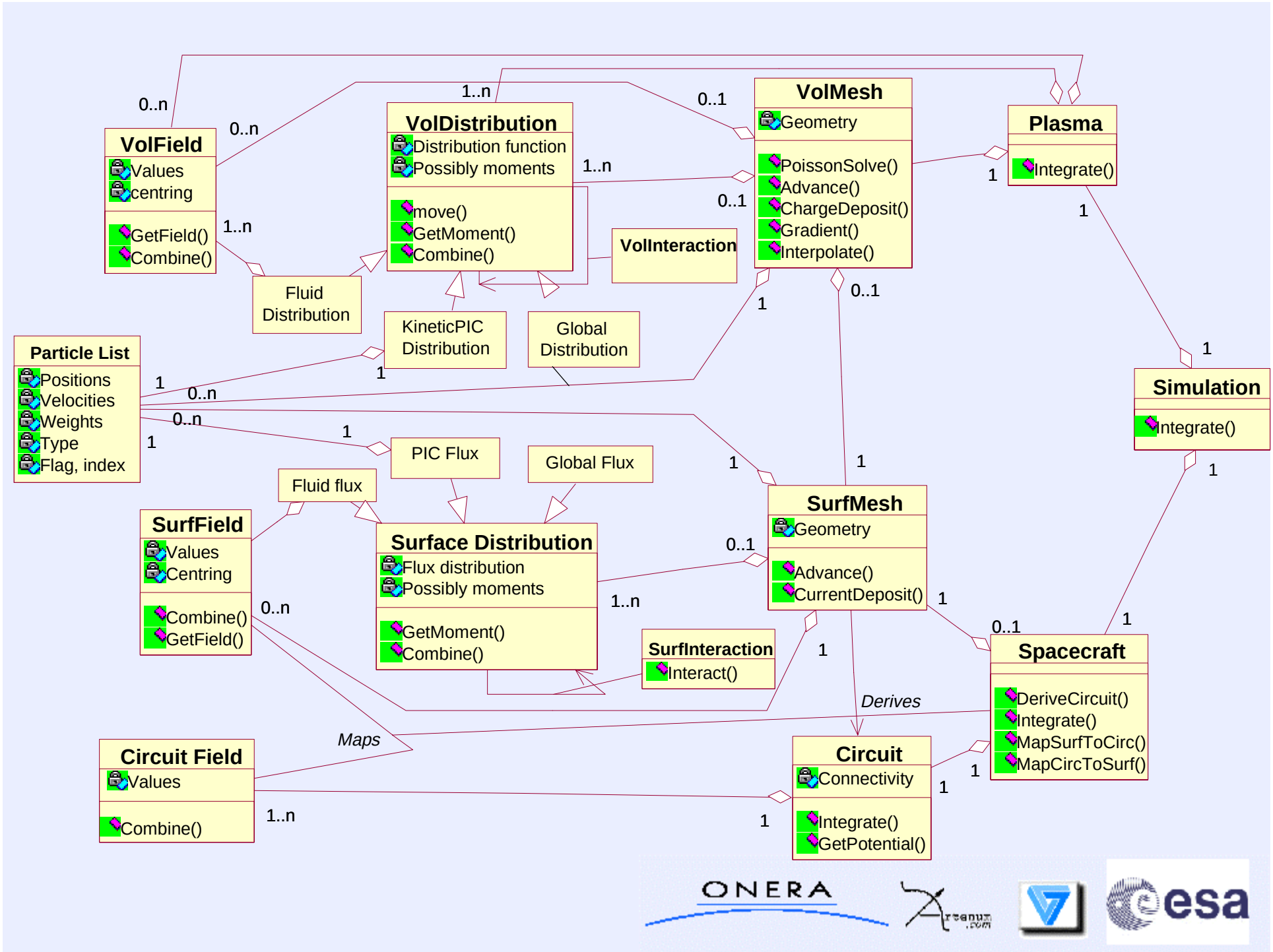
    Here the idea is enrichment (more data)
  - SurfaceField has two derived types:
    - ScalarSurfaceField
    - VectorSurfaceField

    Here, the idea is rather specialisation than enrichment

OOP concepts: genericity - polymorphism 4/4

- In previous examples:
  - Instances of parent-class ParticleList can exist
  - Instances of SurfaceField cannot exist, what would be the data stored, scalar or vector? (an object must be initialised with its data in the constructor)

  ⇒ SurfaceField is an abstract class, it cannot be instantiated, only ScalarSurfaceField or VectorSurfaceField can have instances (real existing objects)
- Other example:
  - VolumeDistribution class: object representing matter, but not defined enough to create a real object (cannot be instantiated)
  - Only derived classes can be instantiated: a PicVolumeDistribution, an AnalyticalVolumeDitribution (Maxwell-Boltzmann), or a FluidVolumeDistribution…
  - Fortran programmer think: “but why have these sub-classes (or sub-types) derived from a common parent class?”
  - Answer: **genericity / polymorphism**:
    - Parent class defines abstract methods, which must be implemented by its derived classes
    - Ex here: move(time), getMoment(order), etc…
    - Implementation is very different for PIC or fluid, but results are similar => a plasma can handle any type of distribution on the same footing whatever its sub-type


VolField
Values
centring
GetField()
Combine()
VolDistribution
Distribution function
Possibly moments
move()
GetMoment()
Combine()
VolMesh
Geometry
PoissonSolve()
Advance()
ChargeDeposit()
Gradient()
Interpolate()
Plasma
Integrate()
VolInteraction
Fluid Distribution
KineticPIC Distribution
Global Distribution
Particle List
Positions
Velocities
Weights
Type
Flag, index
Simulation
Integrate()
PIC Flux
Global Flux
Fluid flux
SurfField
Values
Centring
Combine()
GetField()
Surface Distribution
Flux distribution
Possibly moments
GetMoment()
Combine()
SurfMesh
Geometry
Advance()
CurrentDeposit()
SurfInteraction
Interact()
Spacecraft
DeriveCircuit()
Integrate()
MapSurfToCirc()
MapCircToSurf()
Derives
Maps
Circuit Field
Values
Combine()
Circuit
Connectivity
Integrate()
GetPotential()
0..n
1..n
0..1
1
ONERA
esa

JavaDoc browsing through SpisNum

- Start with SpisNum\doc\index.html
- Ex 1: SurfField
- Ex 2: EField
- Ex 3: PartList
- Ex 4: VolDistrib
- …

Java language in brief

- **OO** language:
  - Really Object Oriented language, neater than C++
  - Multiple heritage of C++ replaced interfaces
- Efficiency:
  - At execution: between 1 and 2 times slower than Fortran or C++
  - At development, much faster than C++ (debugging much more efficient)
- A few characteristics:
  - Garbage collector (no need to de-allocate)
  - Javadoc: very powerful html documentation generation
  - Compilation Just-In-Time (pre-compilation possible)
  - Variables passed by reference
  - Conventions:
    - **Upper/lower case usage: M**y**C**lass **m**y**I**nstance
    - private fields, public methods (OOP, not specific to java) => the way the class is coded can be modified provided the public interface is unchanged (the API (Application Program Interface) : the methods and their arguments)
  - Documentation (language + library API) - download: http://java.sun.com

Eclipse: a wonderful java IDE

- Java Integrated Development Environment (IDE):
  - Source editing
  - GUI debugging

  } Classical in Fortran

  - Syntax correction
  - Automatic completion
  - Hyper-text navigation
  - “Javadoc compatible”

  } One step beyond
- Assets:
  - Increased development efficiency
  - Efficiency gain to enter in an unknown code even larger (SPIS)
  - Very good quality product (few bugs, stability)
- Eclipse project:
  - Open source, community development
  - Now IBM is the major contributor (remains free open source)

SPIS/NUM top level objects and examples

- Examples demonstrating the concepts
- Review of top level objects
- Running the examples: watching objects at work in Eclipse
- Writing code in Eclipse (tomorrow ?)