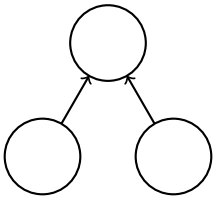
# Application of Object-Oriented Principles and Design Patterns in the architecture of a Monte-Carlo simulation of liquids



PLMC

February 17, 2015 - November 29, 2016

**Abstract**

A classical Monte-Carlo simulation of liquids has a simple and well-defined structure: 1. Intialisation, 2. Iterations of move trial, 3. Results calculation. However, some details may vary such as the geometry (e.g. Bulk / Slab), the thermodynamic ensemble (e.g. Canonical / Grand canonical / Isobaric) or the particles type and interactions (e.g. Hard or Soft Spheres / Apolar or Dipolar Spheres). This tension between constance and variability may be addressed by the Object-Oriented paradigm (e.g. in Fortran, no kidding). By following simple OOP principles, we notice the natural emergence of Design Patterns in the architecture of our program. Those patterns ensure that our simulation program will be robust yet flexible to a certain extent.

However, compared to a procedural programming approach, object-oriented programming may have more pitfalls. For instance, the implementation is likely to be hindered if the overall design is not well-defined in advance. Thus an effort must be made upstream.

The notion of *interface* will be fundamental. It will define the general boundaries of the program. And it will be the key to switch between local alternatives.

Notations:

- `Type[:, …, :]` is an array of `Type`. The special case $\texttt{Type[:, :]}_{\texttt{sym}}$ means it's a 2D symmetric array. Hence, only the (upper or lower) triangular values need to be stored.
- Among the arguments of a `ClassA` :: construct(…), ♢ object: `ClassB` and ♦ object: `ClassB` are shorthands for aggregation and composition respectively. They are mainly used when `ClassB` is defined outside the namespace of `ClassA`, cf. Import. Otherwise they merely simplify the diagram.

# Contents

# 1 Core

## 1.1 Common

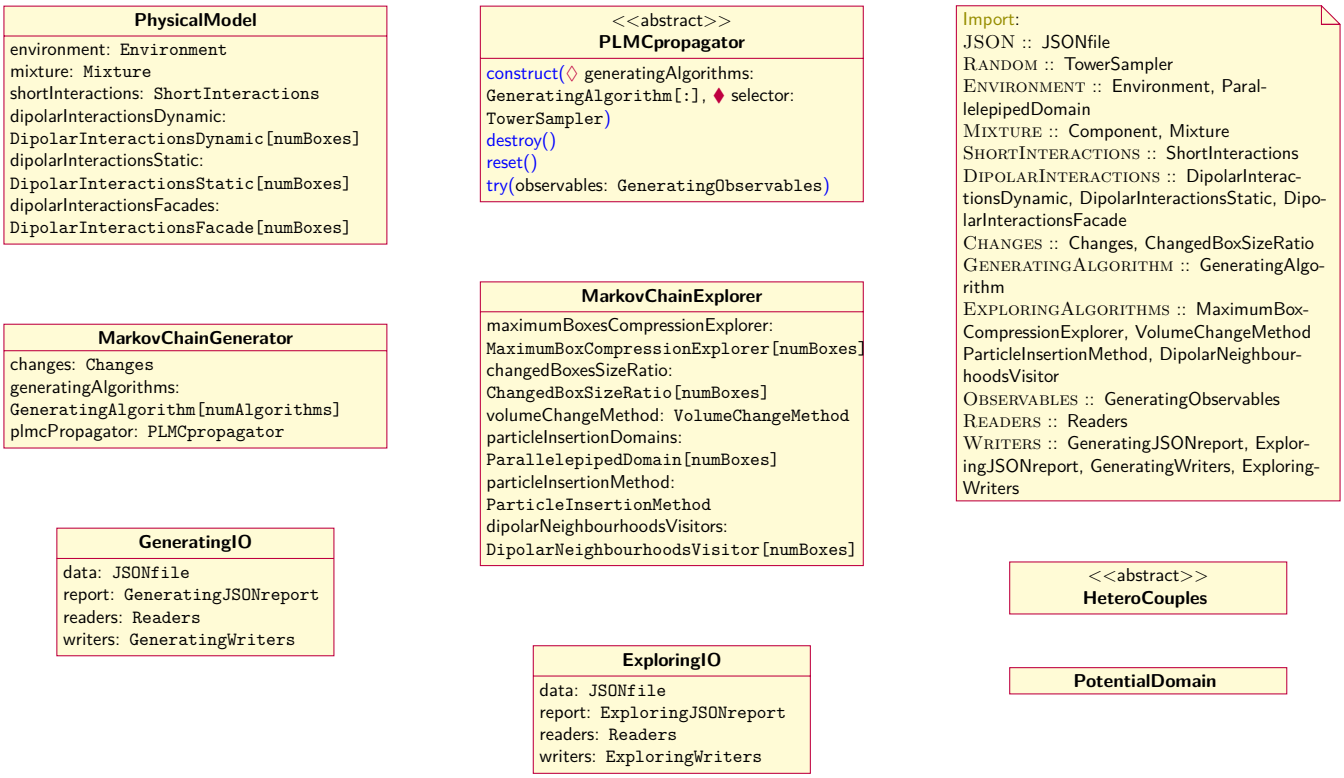
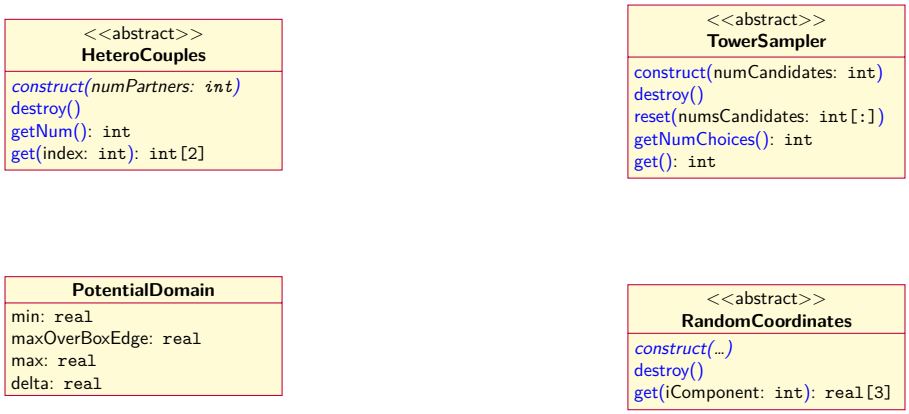



Figure 1: Common

## 1.2 Common utilities & Random




(a) Common: utilities

(b) Random

Figure 2: Common utilities & Random

# 2 Physical model

## 2.1 Environment

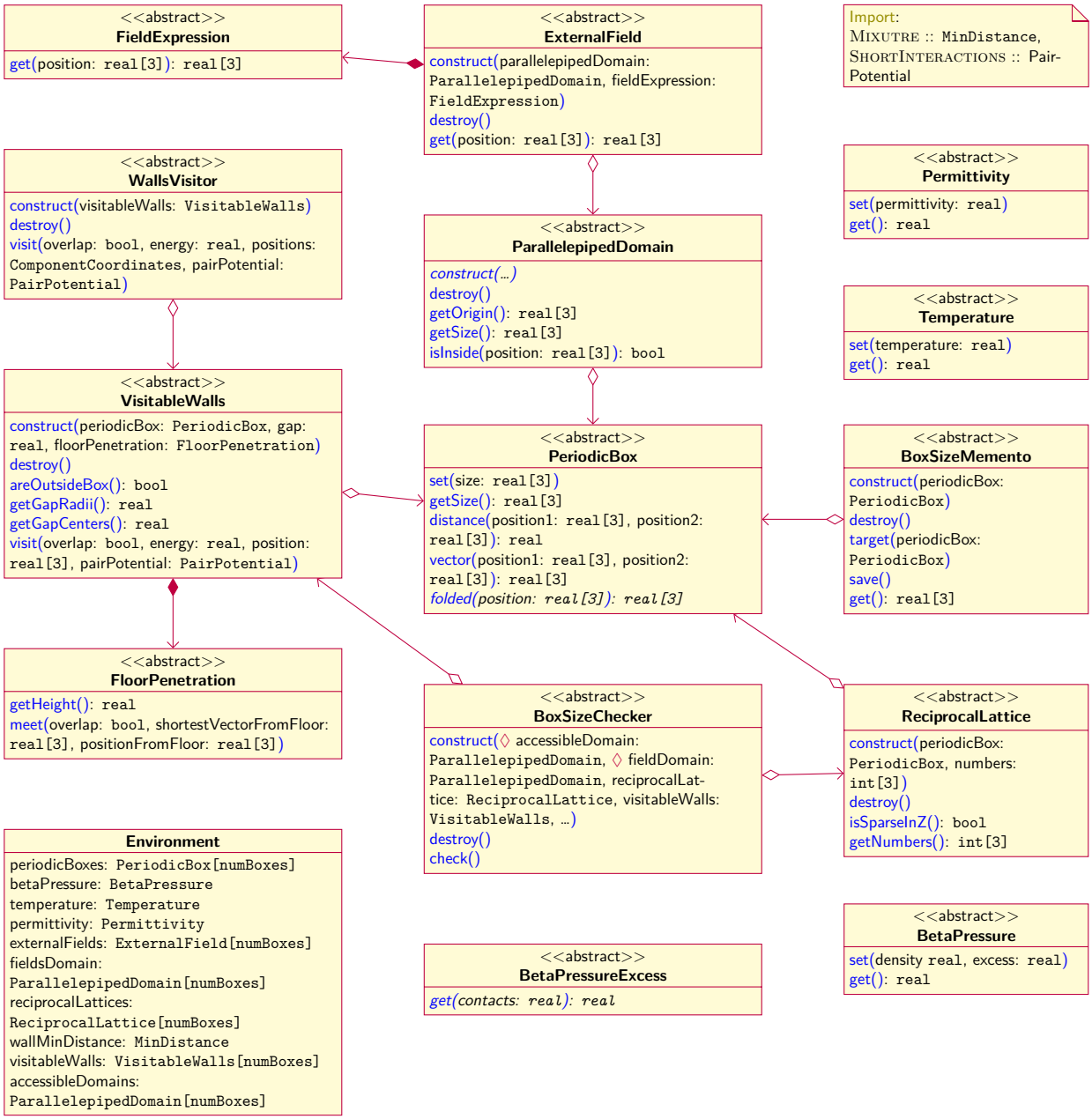



Figure 3: ENVIRONMENT

## 2.2 Particles

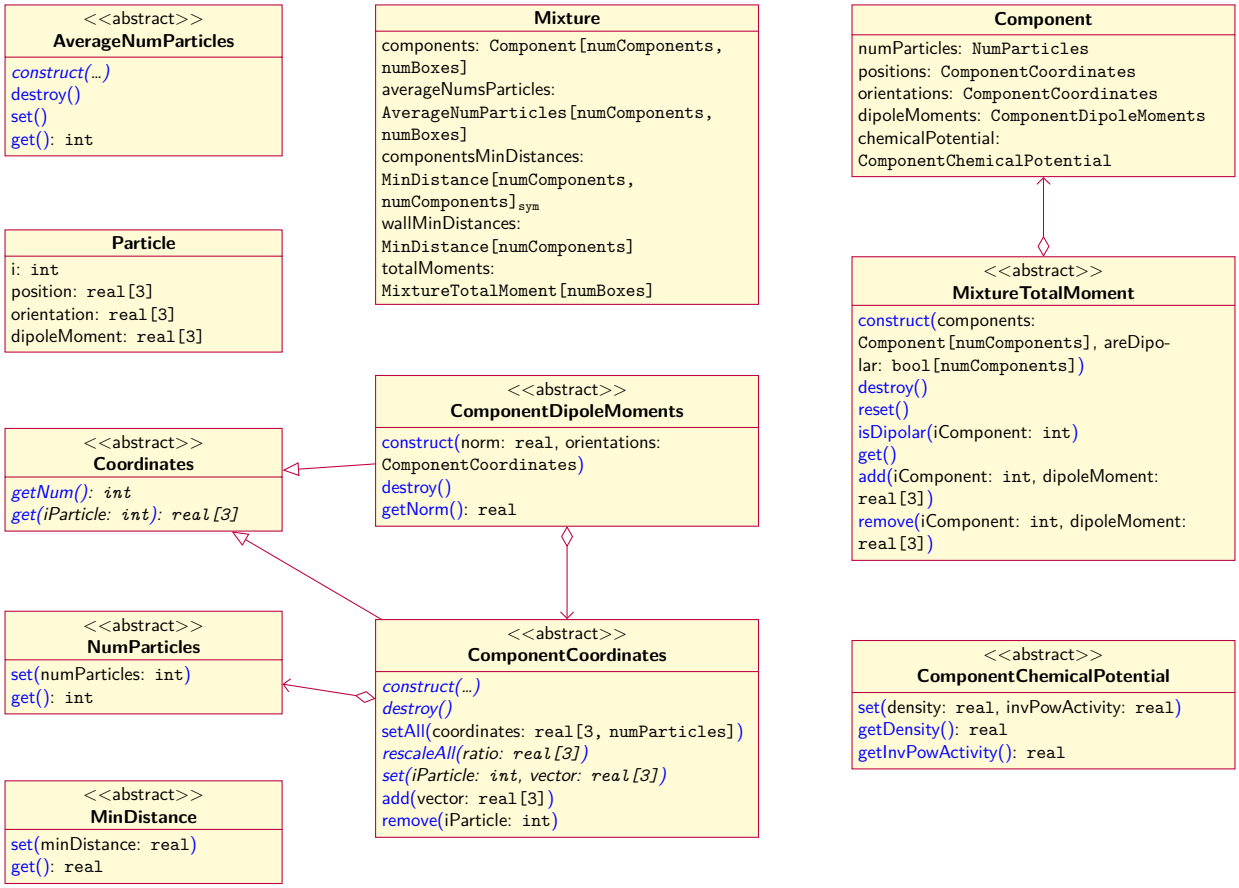



Figure 4: Mixture

## 2.3 Interactions

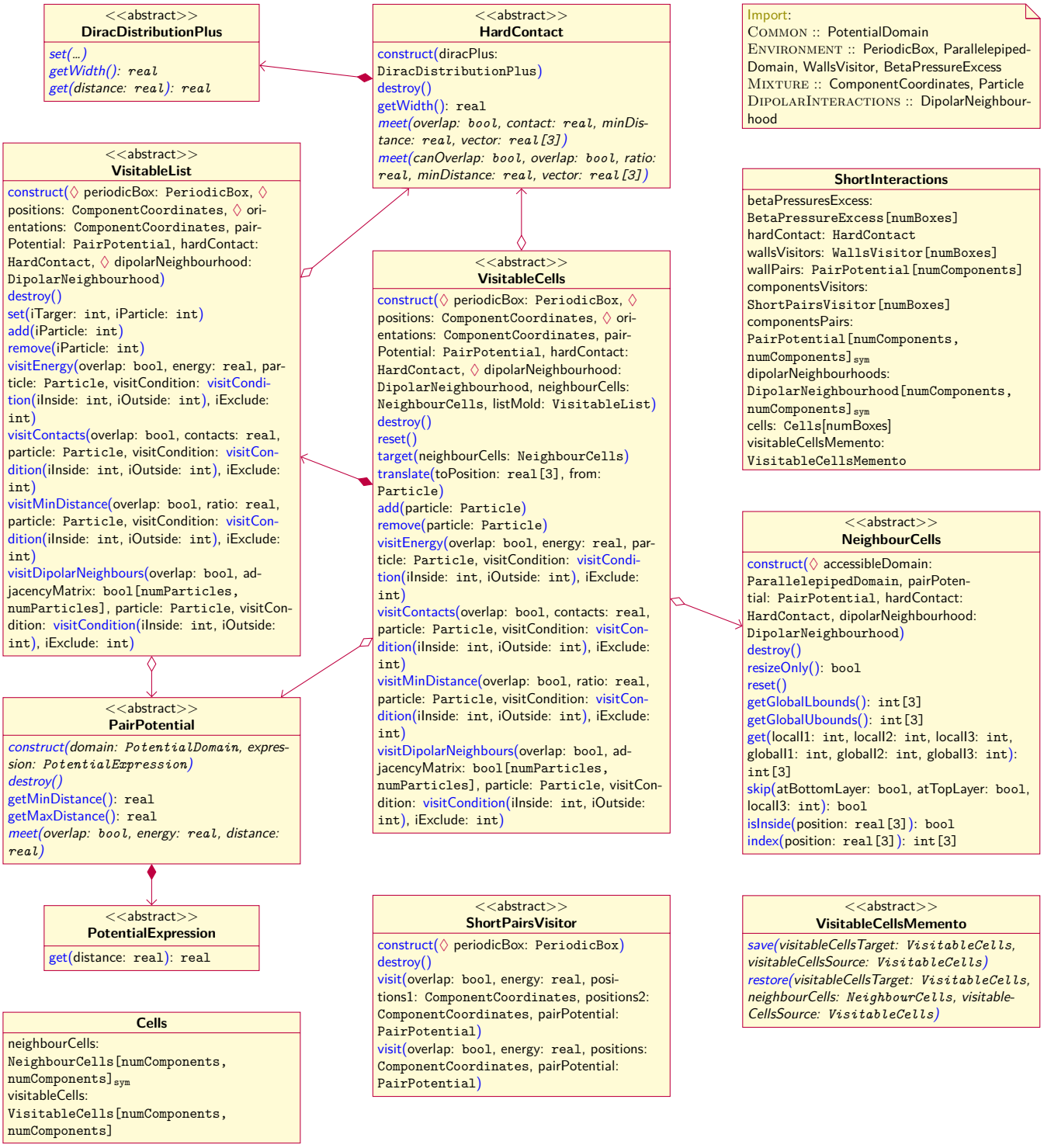



Figure 5: SHORT INTERACTIONS

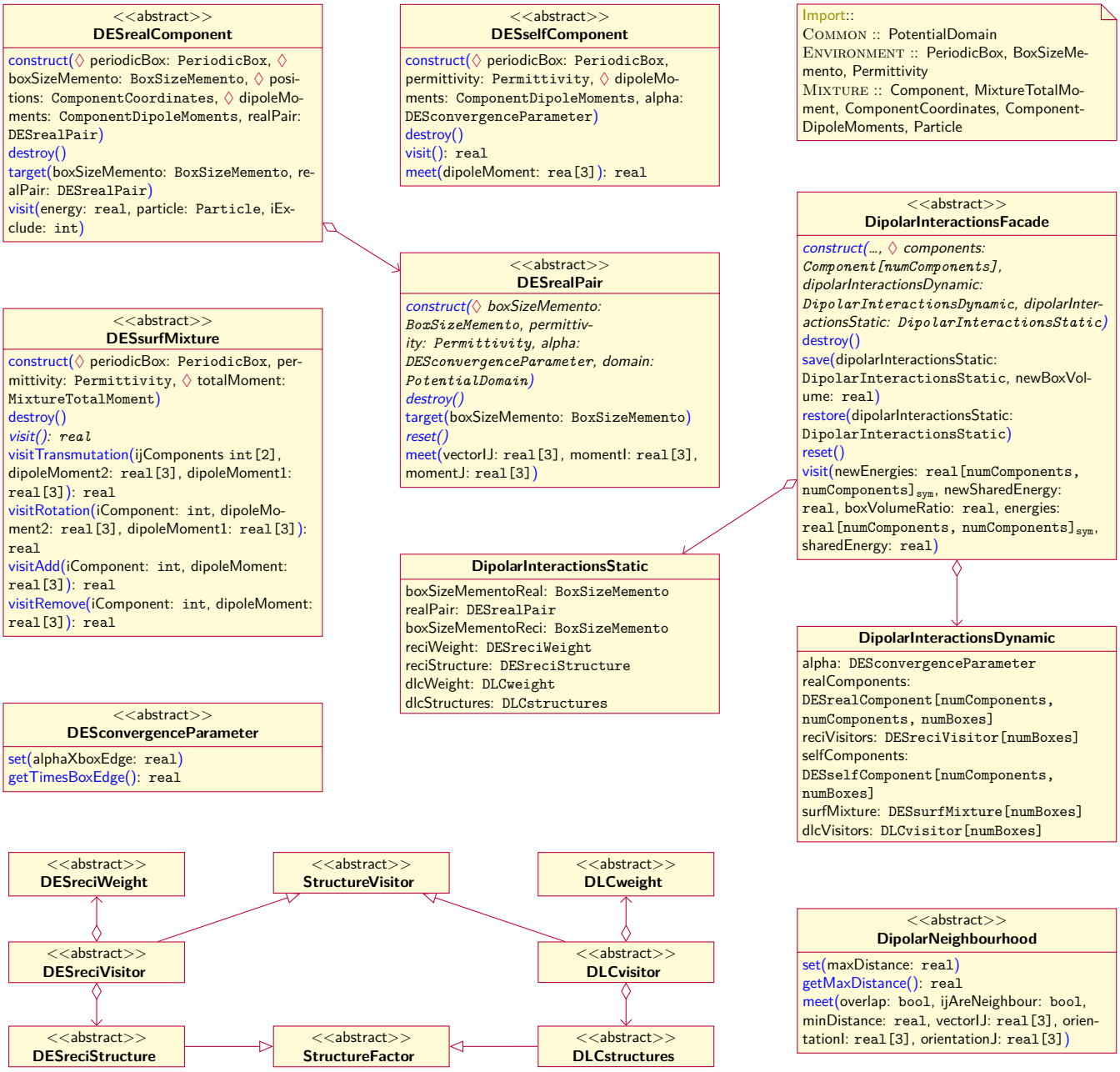



Figure 6: Dipolar Interactions

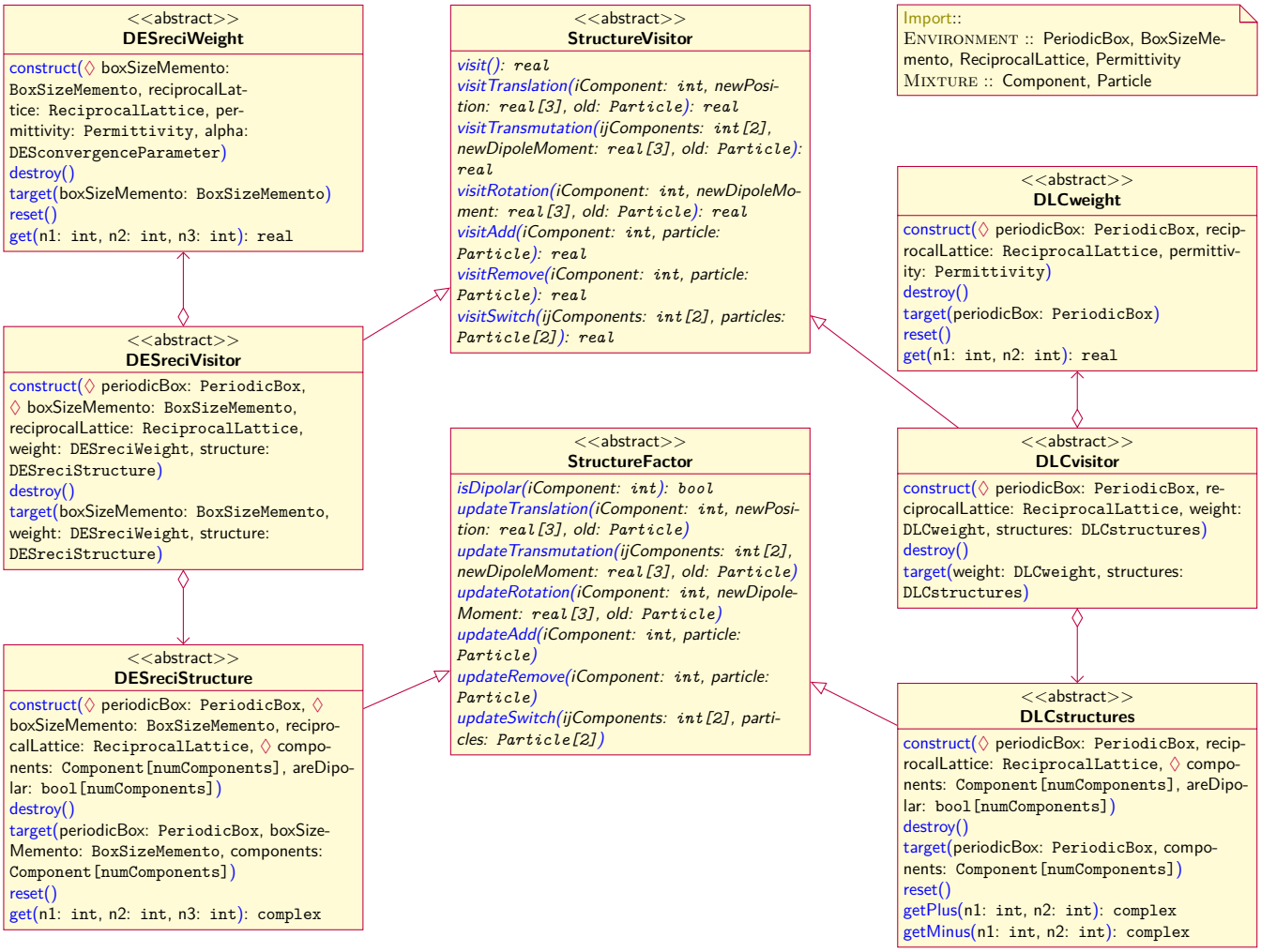



Figure 7: Dipolar Interactions: Reciprocal Space

# 3 Markov chain generator

## 3.1 Changes

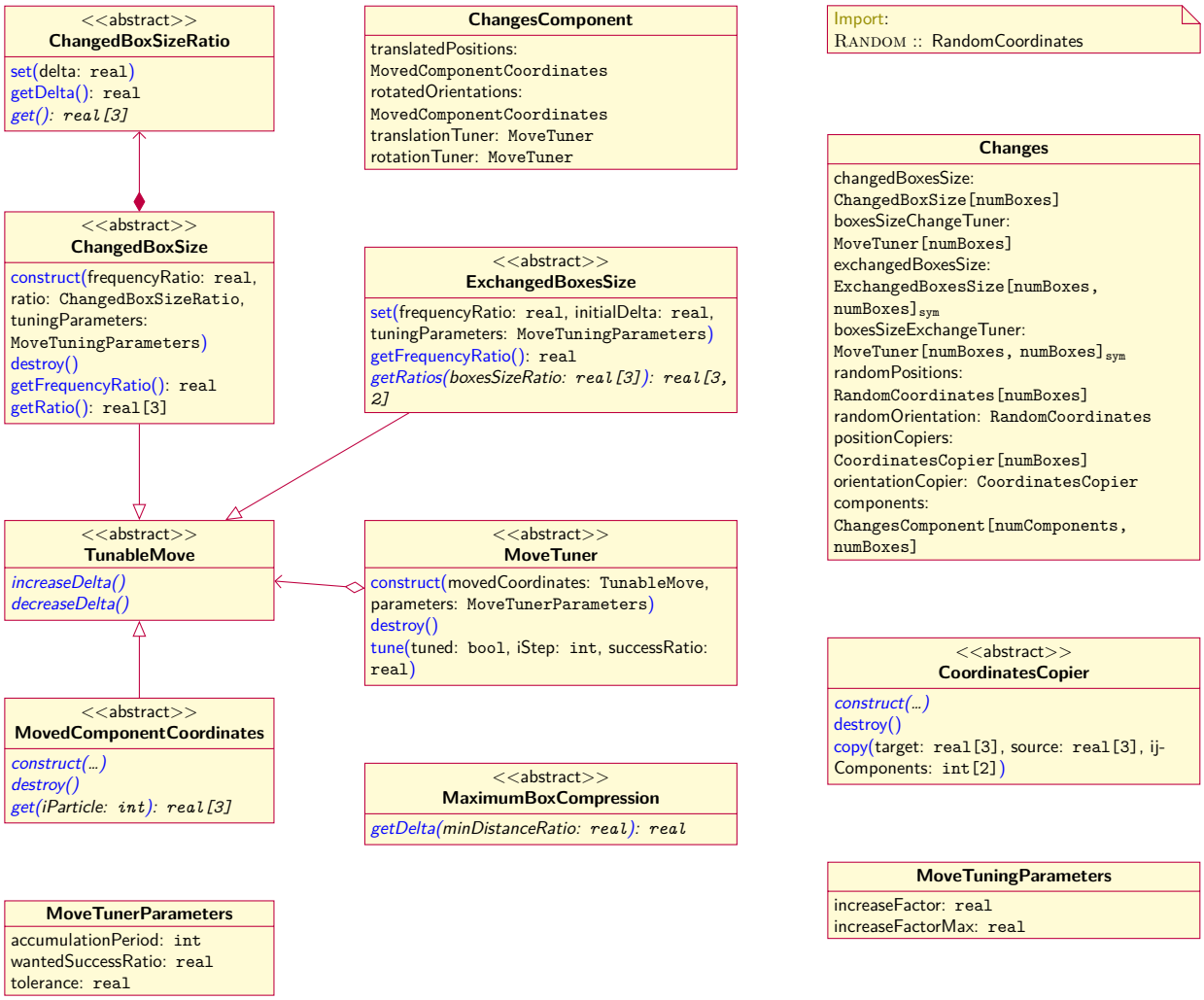



Figure 8: CHANGES

## 3.2 Generating algorithms

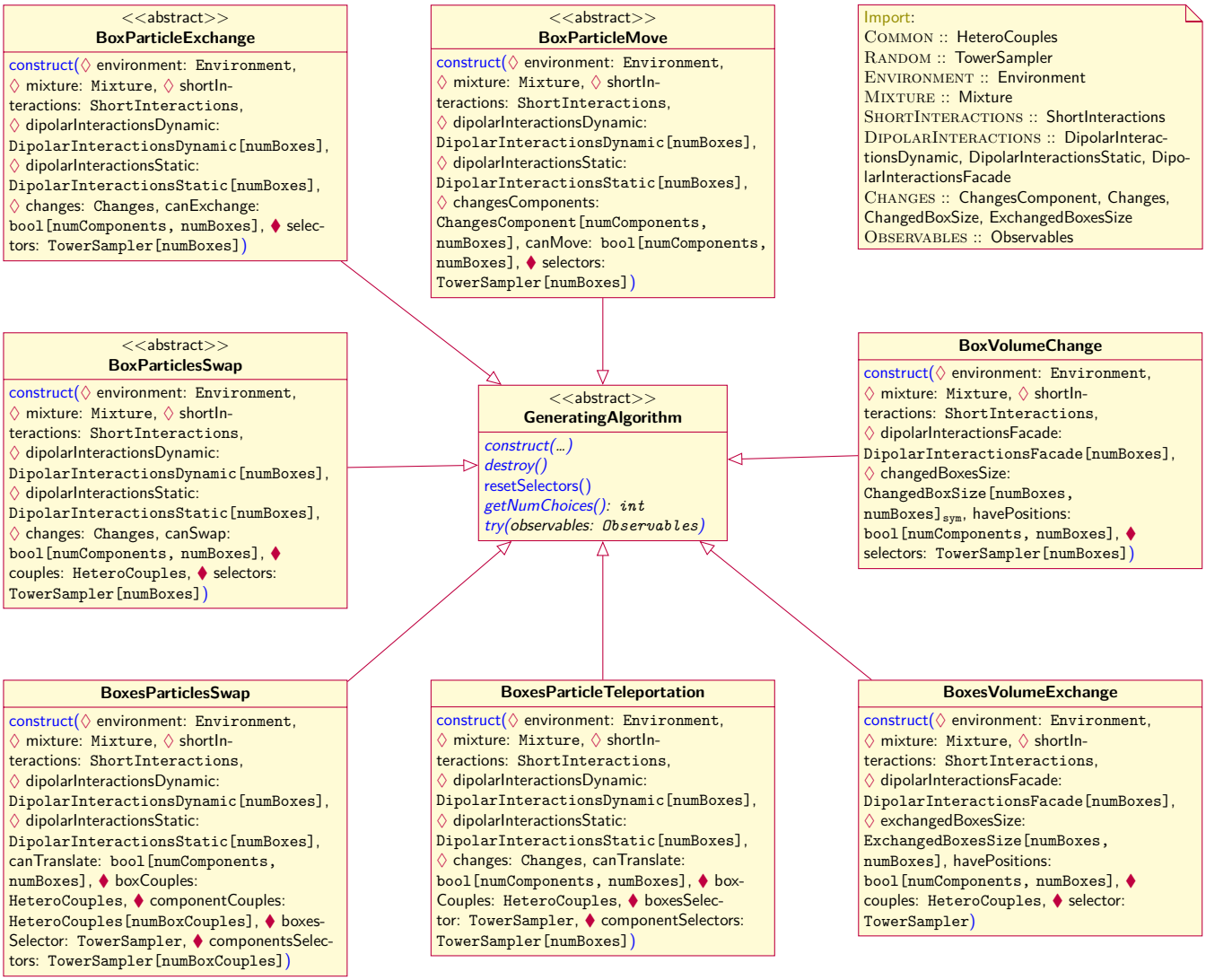



Figure 9: Generating Algorithms

# 4 Markov chain explorer

## 4.1 Exploring algorithms

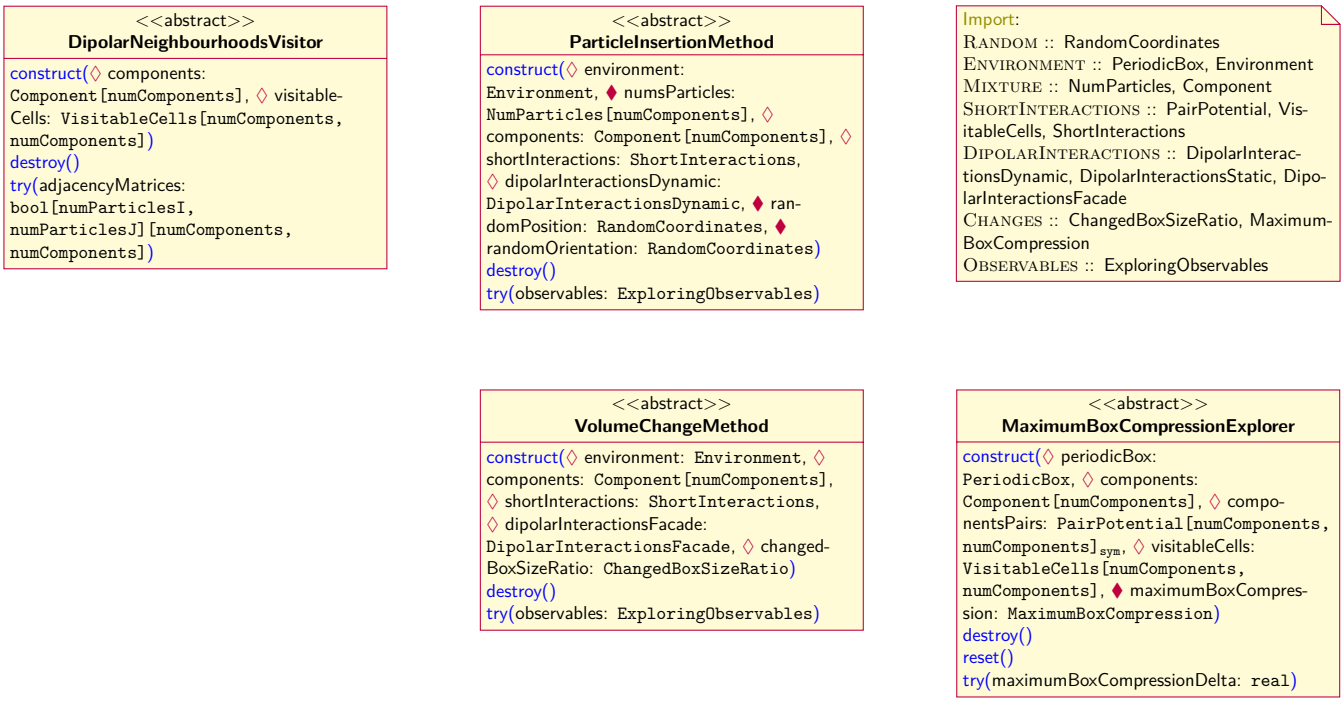



Figure 10: Exploring Algorithms

# 5 Observables


Figure 11: Observables

# 6 Input / Output

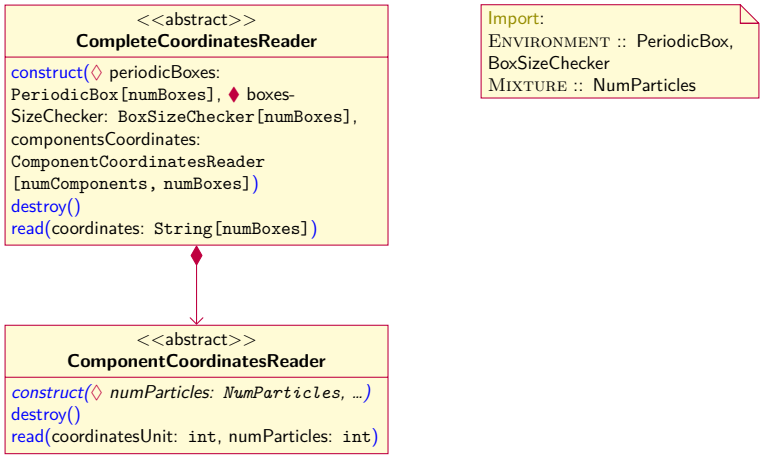



Figure 12: Readers

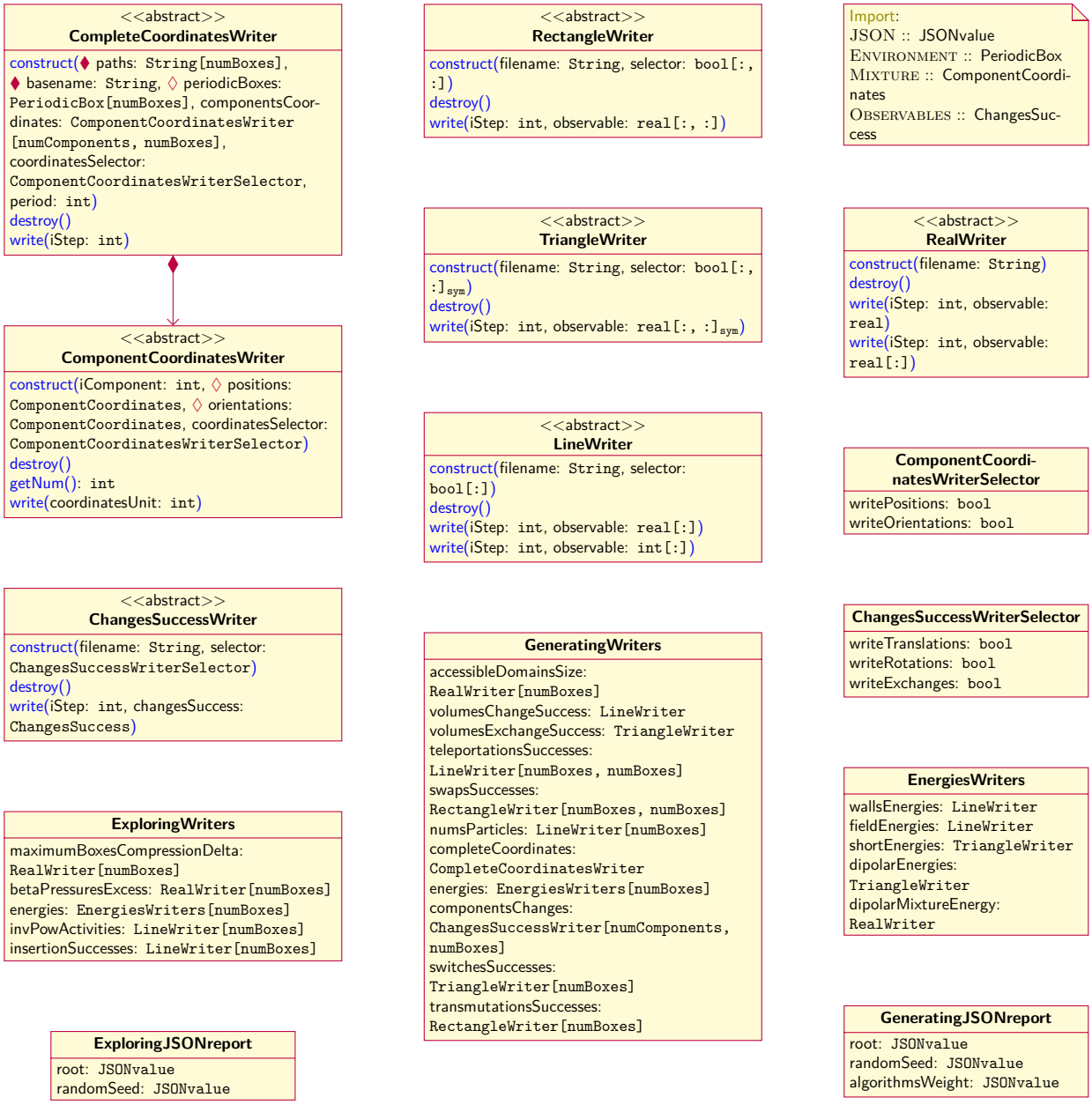



Figure 13: Writers

# References

[1] Eric Freeman et al. *Head First Design Patterns: A Brain-Friendly Guide*. 1st edition. Sebastopol, CA: O'Reilly Media, Oct. 2004. 694 pp. ISBN: 978-0-596-00712-6.

[2] Daan Frenkel and Berend Smit. *Understanding Molecular Simulation*. 2nd. Orlando, FL, USA: Academic Press, Inc., 2001. ISBN: 978-0-12-267351-1.