Just-in-Time Inheritance

Mattias De Wael
Janwillem Swalens
Wolfgang De Meuter

Problem: class with multiple representations

RowMajorMatrix
+ get(i, j)
+ inverse()

SparseMatrix
+ get(i, j)
+ density()

FlexibleMatrix
+ get(i, j)
+ inverse()
+ density()

```
matrix = new FlexibleMatrix();
element = matrix.get(0, 1);
density = matrix.density(); // switch to sparse
inverse = matrix.inverse(); // switch to row-major
```

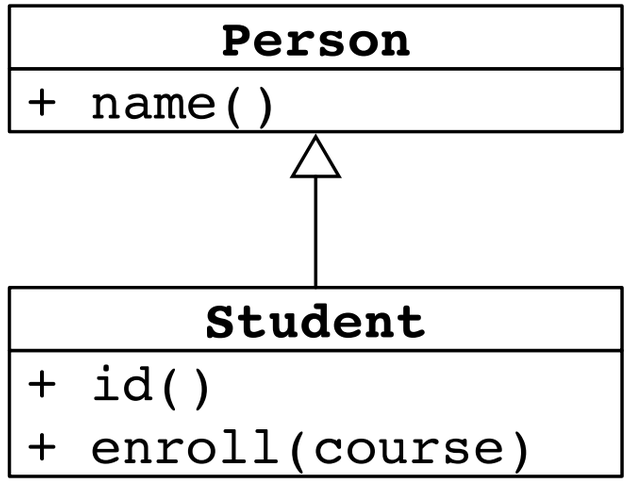
Recap: inheritance is…

Implementation inheritance is a mechanism to share behaviour between objects

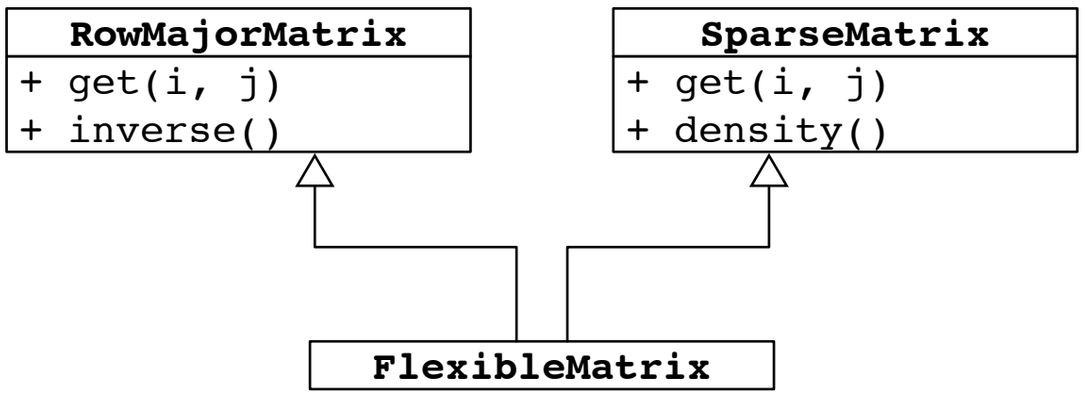
```
student.name()
```

Interface inheritance establishes a **subtype relation** (Liskov substitution principle)

```
Person person = new Student();
```



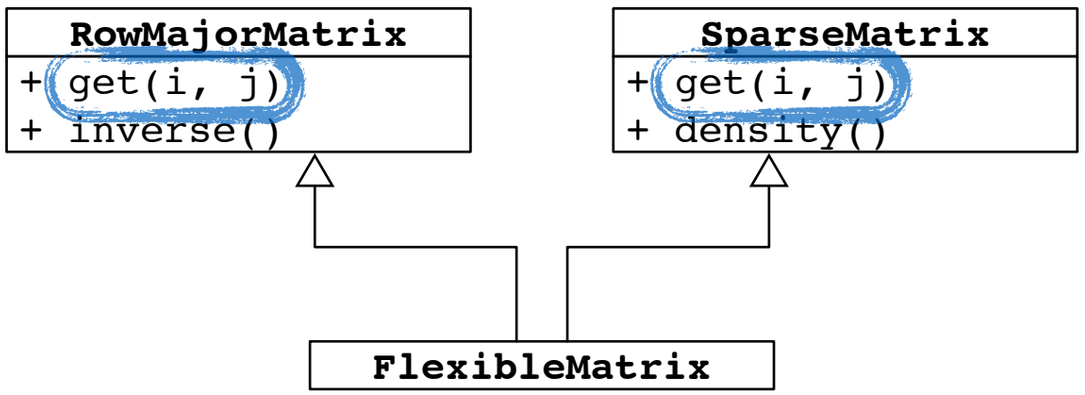
Problem: class with multiple representations



We want inheritance

```
matrix = new FlexibleMatrix();
element = matrix.get(0, 1);
density = matrix.density(); // switch to sparse
inverse = matrix.inverse(); // switch to row-major
```

Multiple inheritance revisited




```
matrix = new FlexibleMatrix();
element = matrix.get(0, 1);
density = matrix.density();
inverse = matrix.inverse();
```

Ambiguity: which `get(i, j)` is invoked?

Treaty of Orlando [Lieberman et al, 87]

classifies inheritance mechanisms

1. **static** (structural) ↔ **dynamic** (behavioural)

patterns of sharing
fixed when object
is created

patterns of sharing can
change when message
is received

2. **explicit** ↔ **implicit**

allow **explicit selection** of
inheritance (of single method)
by developer

automatic & uniform selection
by system

3. **per object** ↔ **per class**

[Lieberman et al, 87] H. Lieberman, L. Stein, and D. Ungar. Treaty of Orlando. In *Addendum to the Proceedings of OOPSLA '87*.

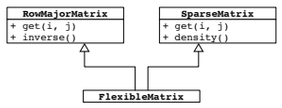
Existing solutions for ambiguity

Ambiguity Rejection (C++)

```
matrix.get(0, 1);
→ Error: member ‘get’ found in multiple base
  classes of different types.
```

static (structural) — implicit

Linearisation (Python)



→ `[FlexibleMatrix, RowMajorMatrix, SparseMatrix]`

static (structural) — implicit

Prioritised Multiple Inheritance (Self)

parents have priorities

static (structural) — explicit (per parent)

Select and Rename (Eiffel)

```
class FlexibleMatrix inherit
  RowMajorMatrix rename get as rowMajorGet
  SparseMatrix   rename get as sparseGet
```

static (structural) — explicit (per method)

Explicit Inheritance (C++)

```
matrix.RowMajorMatrix::get(0, 1);
```

dynamic (behavioural) — explicit (per call)

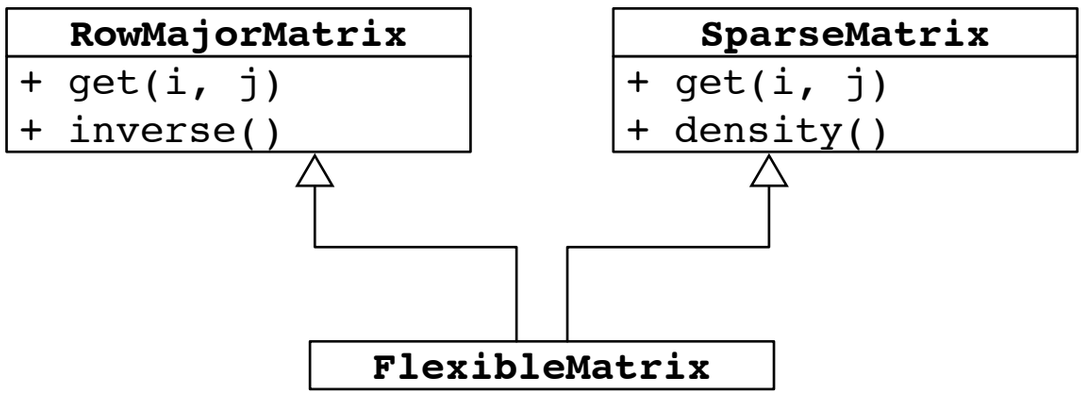
???

```
matrix.get(0, 1);
```

dynamic — implicit

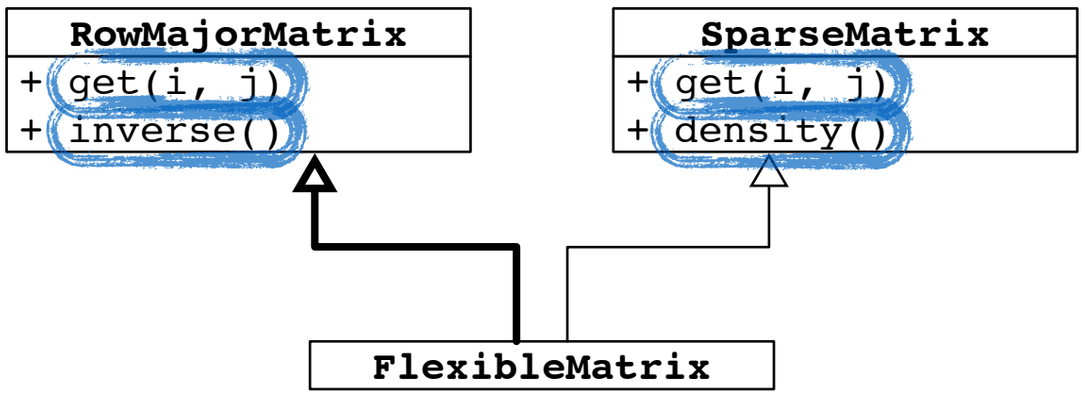
Just-in-Time inheritance

```
class FlexibleMatrix combines RowMajorMatrix, SparseMatrix { … }
```



Just-in-Time inheritance

```
class FlexibleMatrix combines RowMajorMatrix, SparseMatrix { … }
```



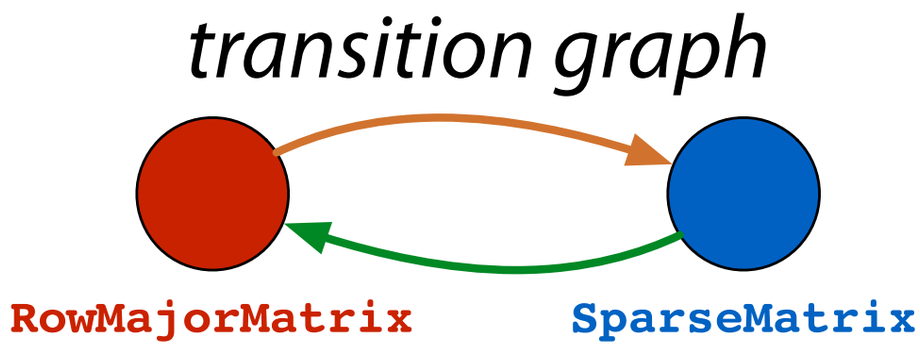
favoured parent

```
matrix = new FlexibleMatrix.SparseMatrix();
density = matrix.density();
element = matrix.get(0, 1);
inverse = matrix.inverse();
element = matrix.get(0, 1);
```

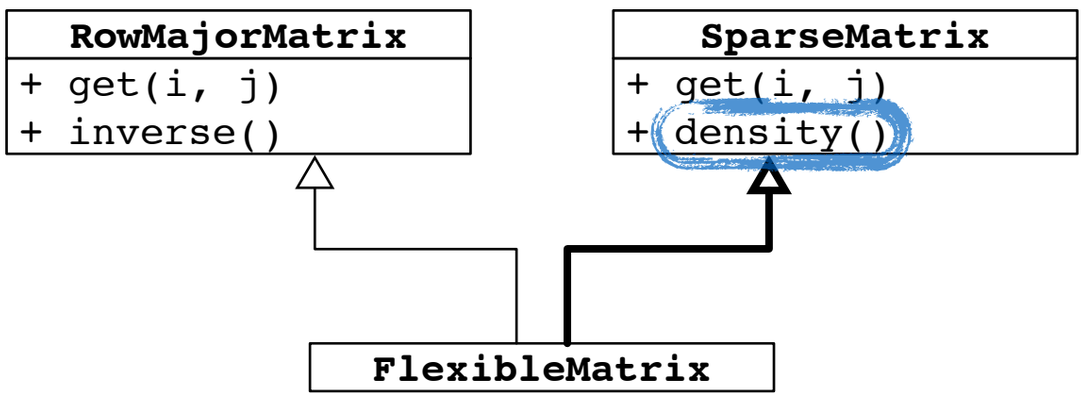
Transition functions

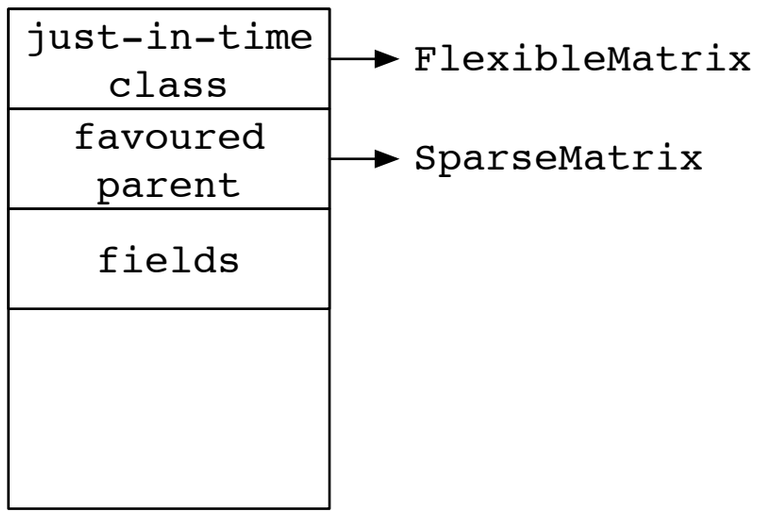
```
class FlexibleMatrix combines RowMajorMatrix, SparseMatrix {
    RowMajorMatrix to SparseMatrix {
        …
    }

    SparseMatrix to RowMajorMatrix {
        …
    }
}
```



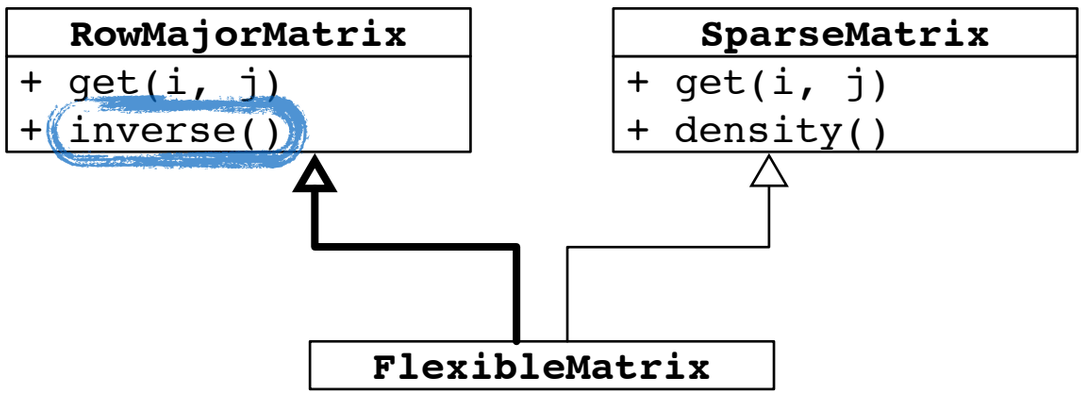
Method look-up: direct

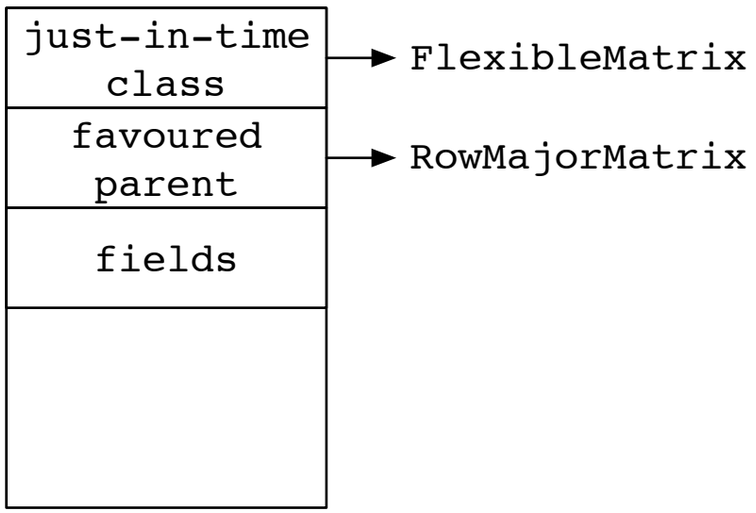




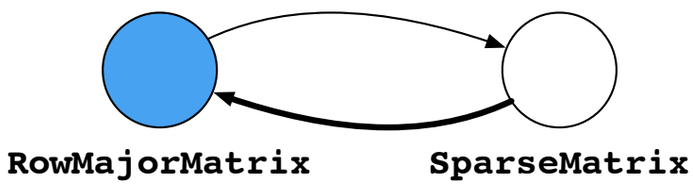

```
matrix = new FlexibleMatrix.SparseMatrix();
density = matrix.density();
```

Method look-up: indirect






```
matrix = new FlexibleMatrix.SparseMatrix();
matrix.inverse();
```



JIT inheritance is dynamic and implicit

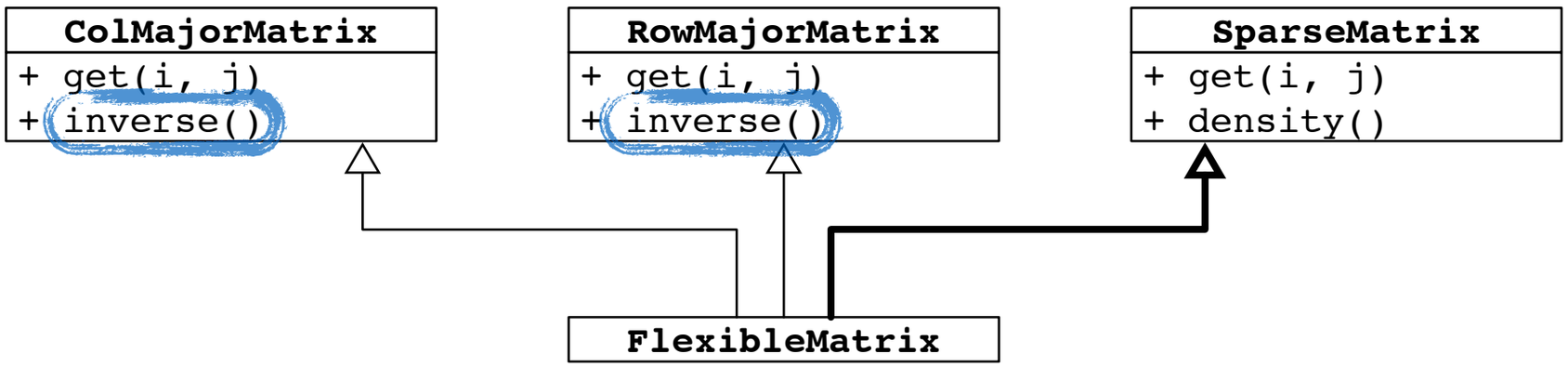
dynamic

patterns of sharing can change when message is received

implicit

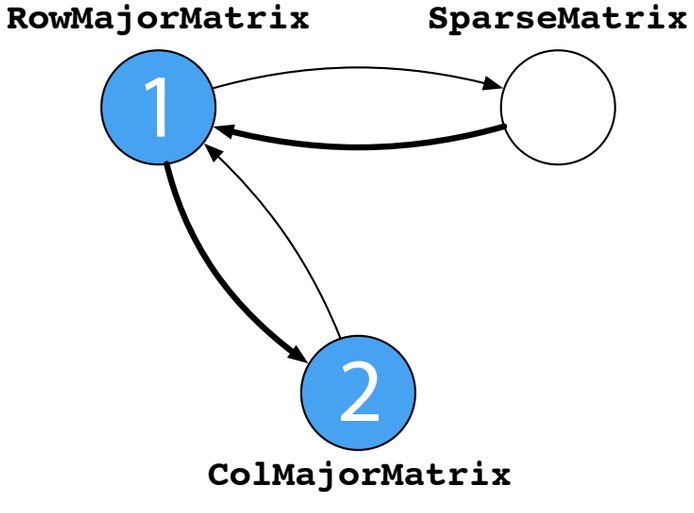
automatic & uniform selection by system,
developer does not specify which implementation to use

Method look-up: indirect, multiple paths

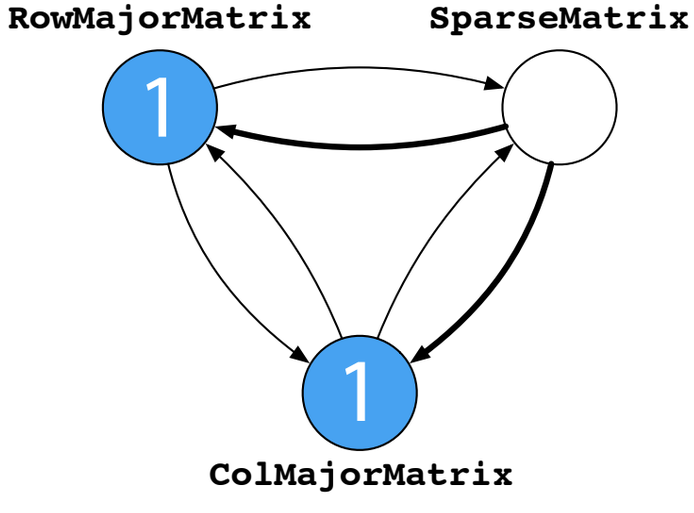



```
matrix = new FlexibleMatrix.SparseMatrix();
matrix.inverse();
```

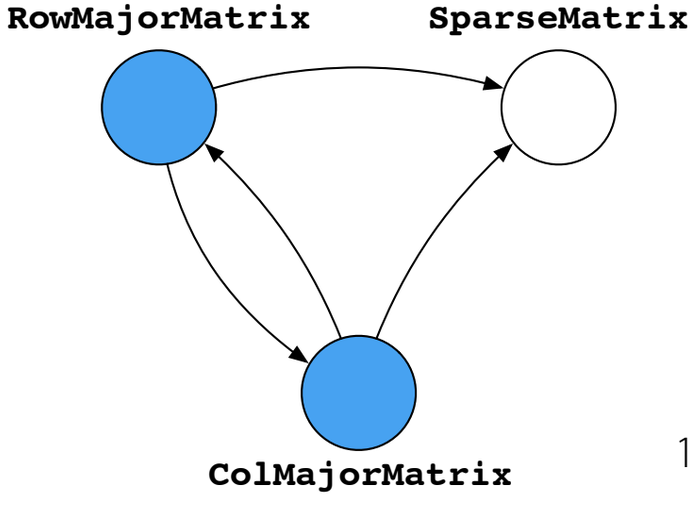
choose shortest path



tie-breaker: order in **`combines`**

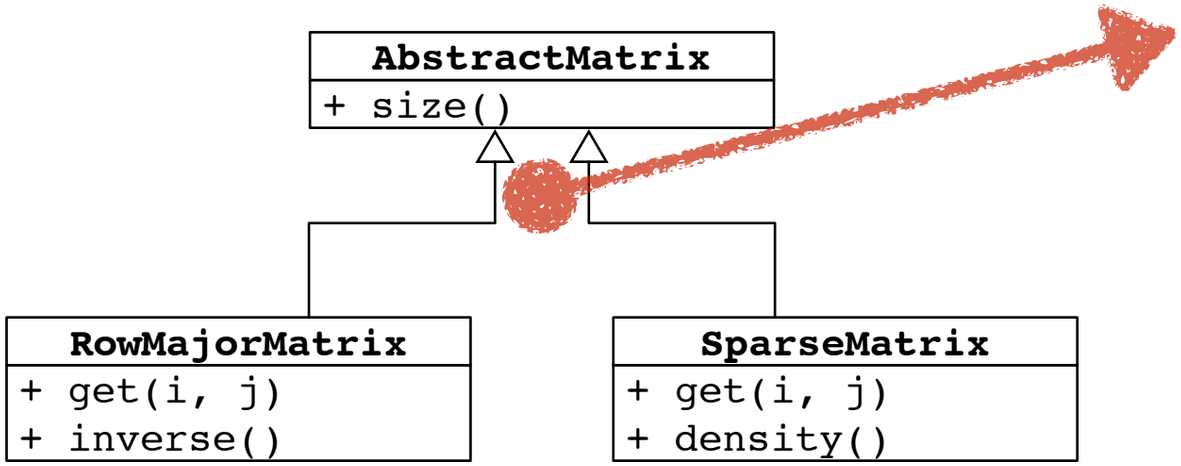


no path: exception



Class hierarchy is unconventional

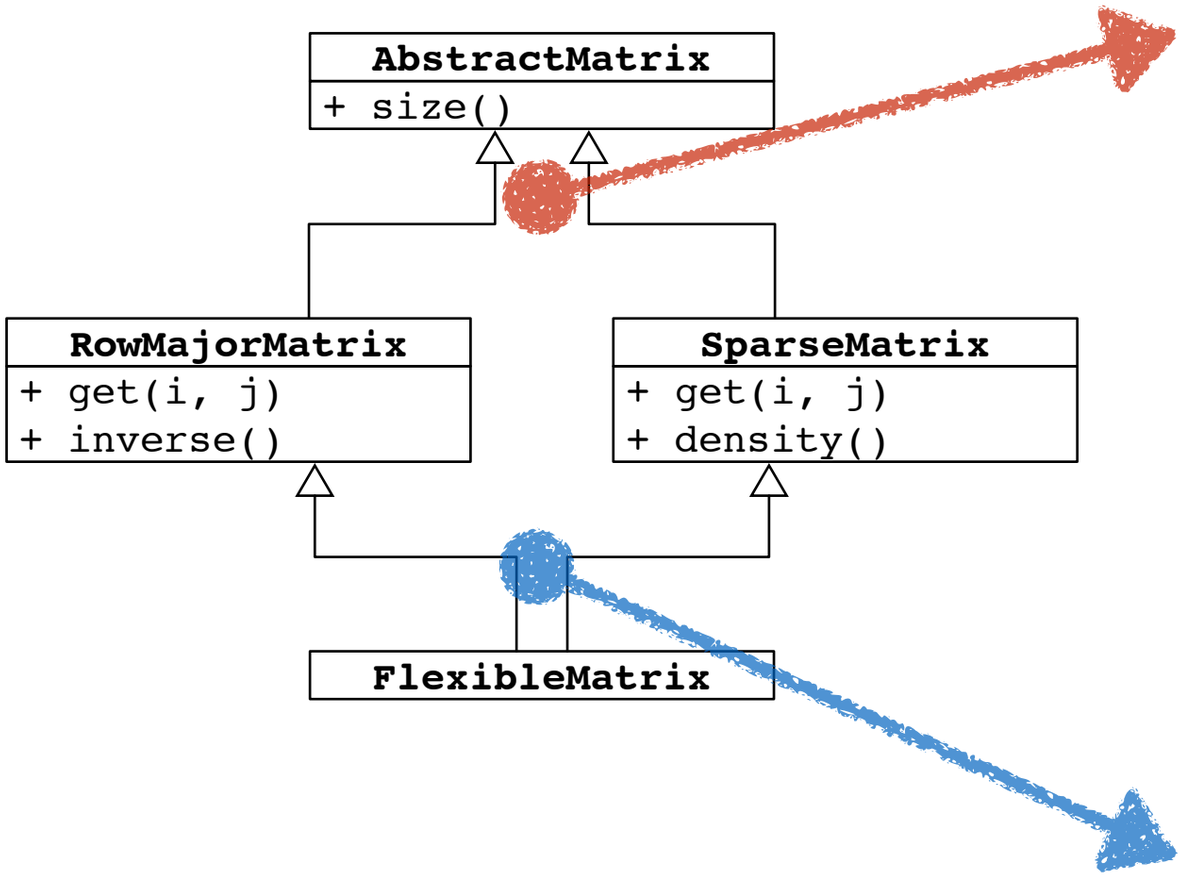
```
class RowMajorMatrix extends AbstractMatrix
class SparseMatrix   extends AbstractMatrix
```



`AbstractMatrix` represents the concept "matrix", defines common implementation and interface

Class hierarchy is unconventional

```
class RowMajorMatrix extends AbstractMatrix
class SparseMatrix   extends AbstractMatrix
class FlexibleMatrix combines RowMajorMatrix, SparseMatrix
```



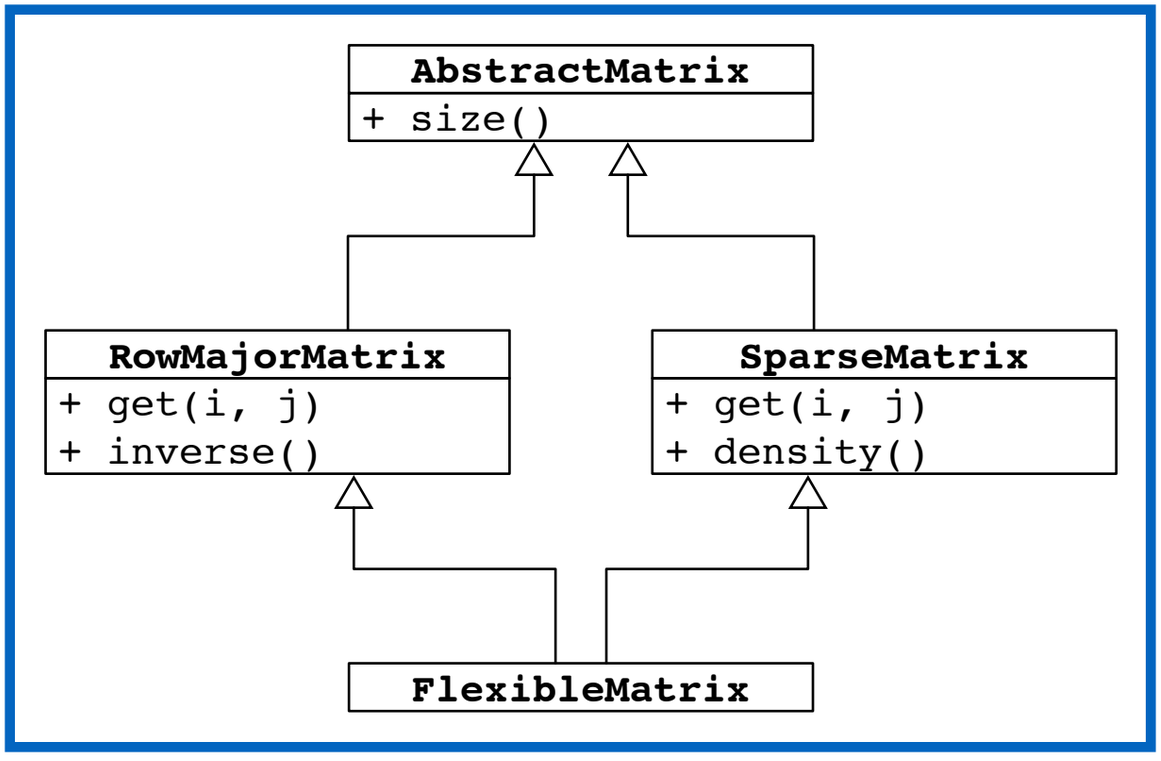
`AbstractMatrix` represents the concept "matrix", defines common implementation and interface

Implementation inheritance: `FlexibleMatrix` re-uses the behavior of its parents

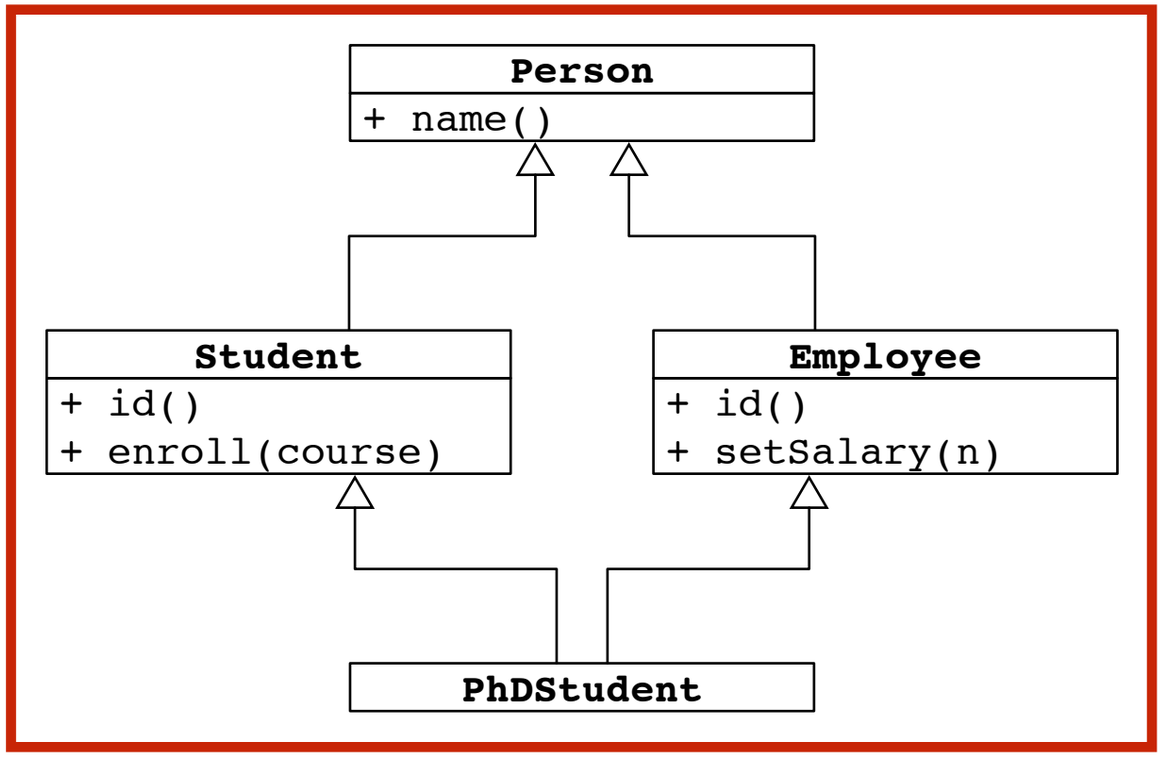
Interface inheritance: `FlexibleMatrix` is a subtype of its parents

Class hierarchy is unconventional since use case is different

```
class RowMajorMatrix extends AbstractMatrix
class SparseMatrix   extends AbstractMatrix
class FlexibleMatrix combines RowMajorMatrix, SparseMatrix
```



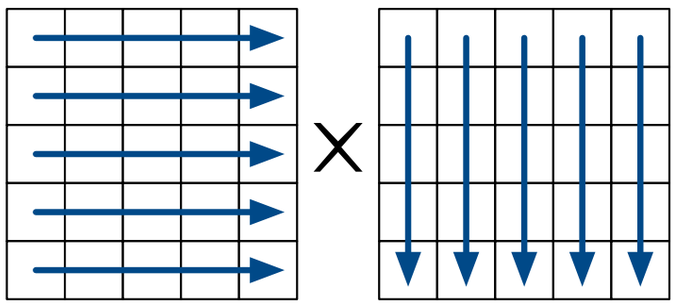

```
class PhDStudent extends Student, Employee
```

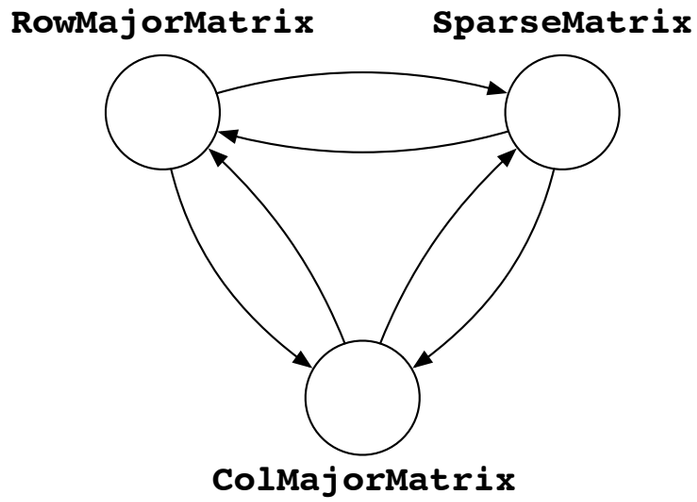


"is-a **OR** is-a" rather than "is-a **AND** is-a"

Application: data structure representations

```
c = multiply(a, b);
```

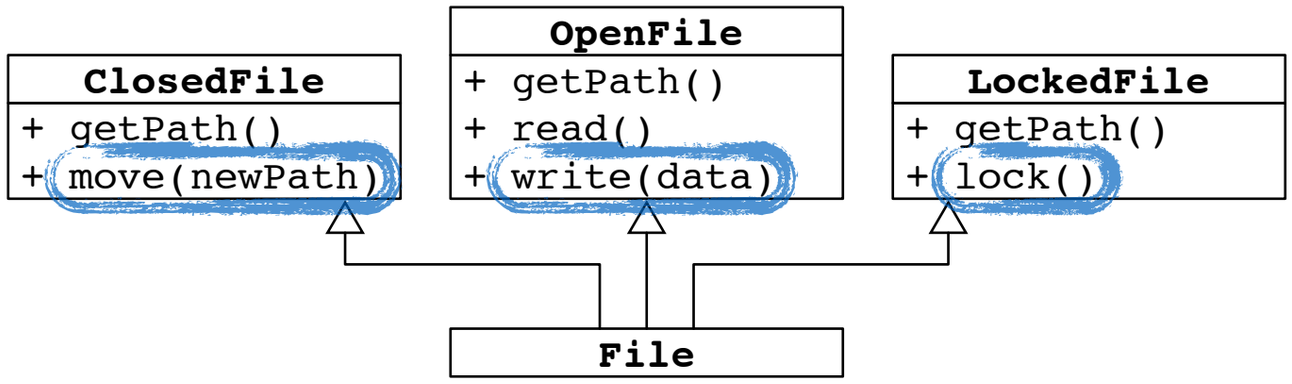


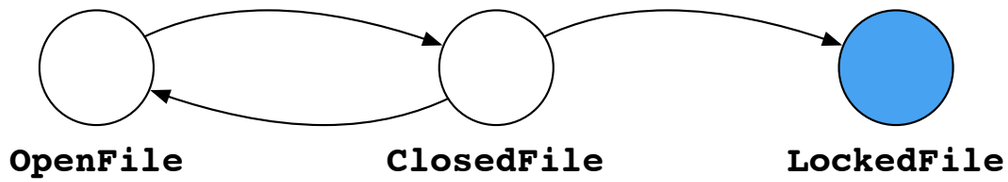


Alternative:
strategy pattern
→ verbose & less extensible (same interface)

non-functional representation changes

Application: class with states






```
file = new File.ClosedFile("temp.txt");
file.write("test123");
file.move("/tmp");
file.lock();
file.write("test123"); ↯
```

Alternative:

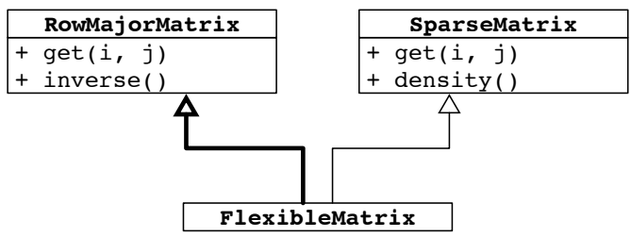
class with states; state pattern

→ verbose & less modular

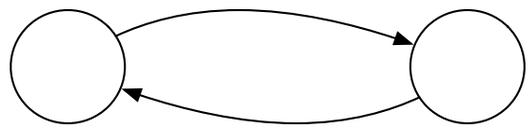
```
class File {
  read() {
    switch (this.mode) {
      case CLOSED:
        this.filePtr = System.fopen(this.path);
        this.mode = OPEN;
        break;
      case LOCKED:
        throw new Exception("Cannot open locked file.");
    }
    return this.filePtr.readln();
  }
}
```

dynamic functional representation changes

Conclusion



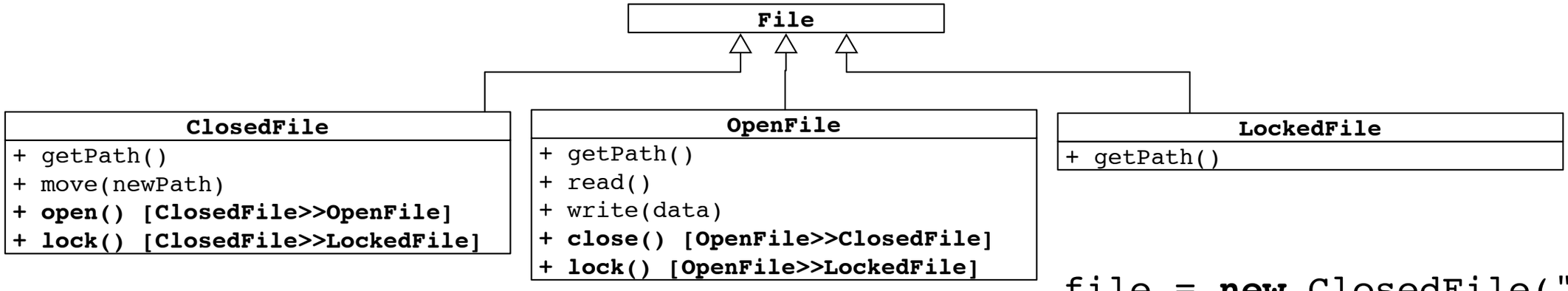
Just-in-Time inheritance:
multiple inheritance with **favoured parent**
transitions



dynamic & **implicit**

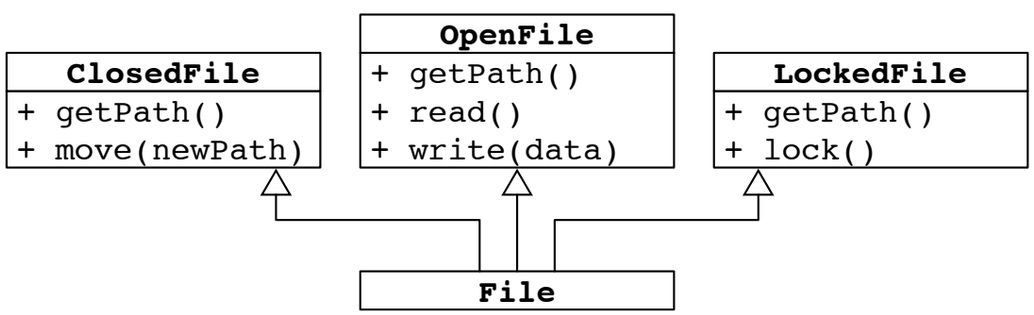
models “is-a **OR** is-a” relation: representation changes
replaces state, strategy pattern
verbosity ↘ modularity ↗ extensibility ↗

Typestate-oriented programming vs. JIT inheritance



- **explicit, static** transitions
- **static functional** repr. changes

```
file = new ClosedFile("temp.txt");
file.open();
file.write("test123");
file.close();
file.move("/tmp");
file.lock();
file.write("test123"); ↯
```

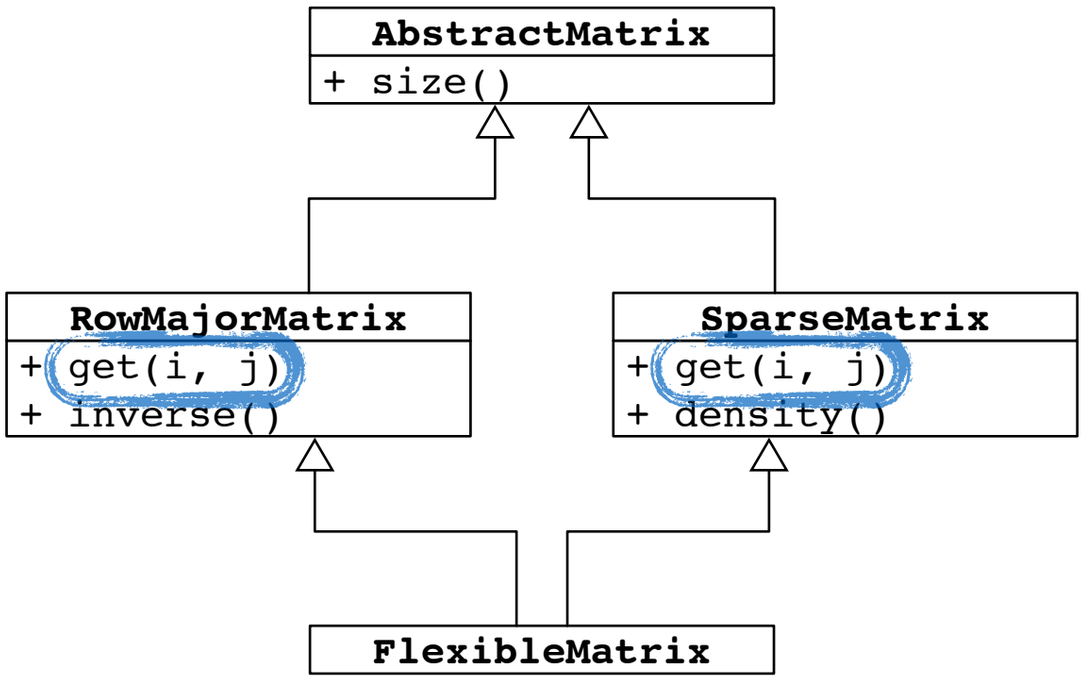


- **implicit, dynamic** transitions
- **non-functional** repr. changes and **dynamic functional** repr. changes

```
file = new File.ClosedFile("temp.txt");
file.write("test123");
file.move("/tmp");
file.lock();
file.write("test123"); ↯
```

[Aldrich et al, 09] J. Aldrich, J. Sunshine, D. Saini, Z. Sparks. Typestate-Oriented Programming. In *Onward '09*.

Diamond problem




```
matrix = new FlexibleMatrix();
matrix.get(0, 1);
```

Which `get(i, j)` is invoked?

"diamond problem"
"diamond of death"