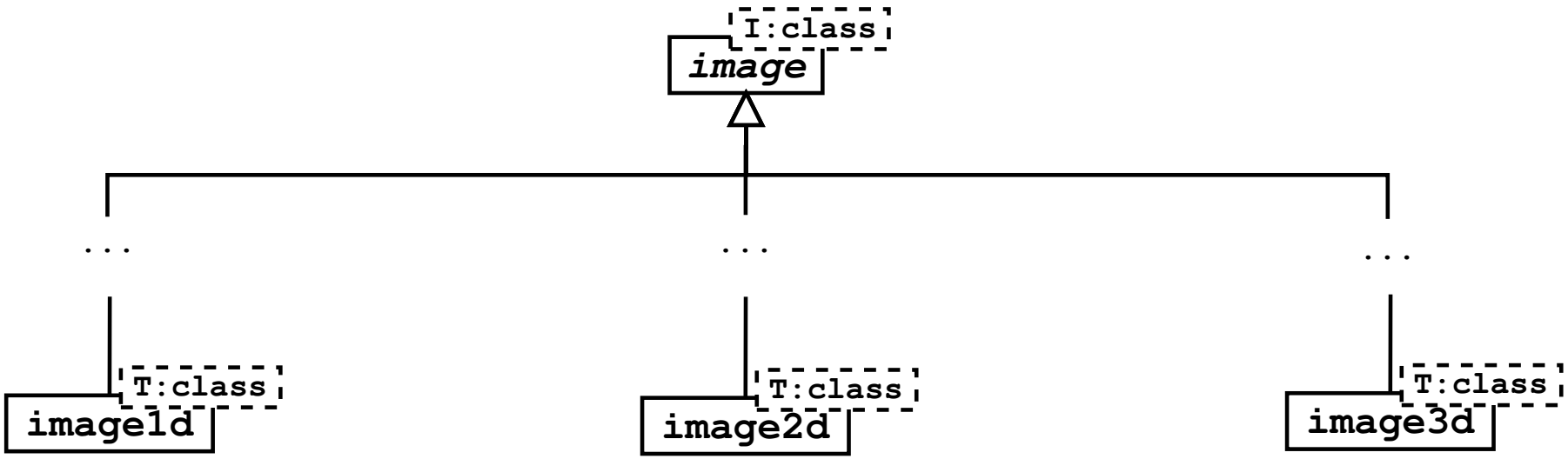
Olena: image classes

David Lesage <david@lrde.epita.fr>

LRDE Olena Day, February 18, 2003

Table des matières

Image hierarchy



- Static, parametrized hierarchy
- Statically recursive hierarchy

Image classes

Pre-requisites

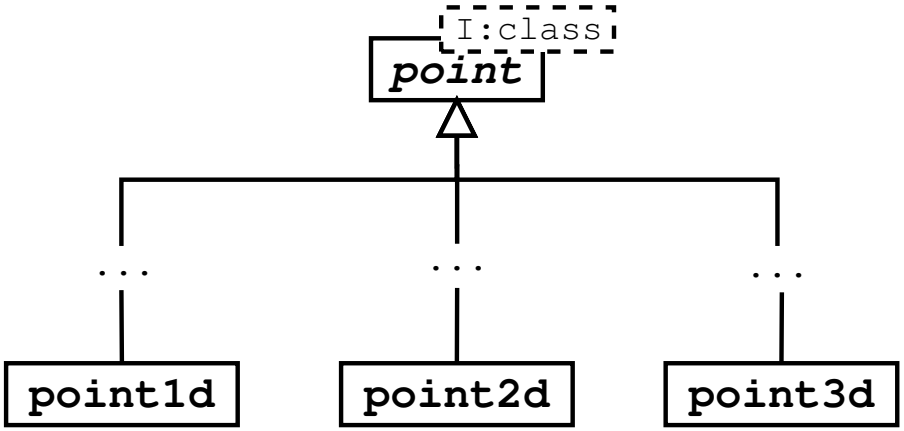
- Hidden data gestion
- Concrete Image = data proxy
  - ▷ smart pointer to allocated data
- “Non-concrete” images design capability
  - ▷ ex: “function” images generating values at calling

Concrete image classes available

- Three concrete image types available:
  - `Image1d<T>`
  - `Image2d<T>`
  - `Image3d<T>`
- Algorithms' genericity:
  - towards data type : T parameter
  - towards image's dimension :
    base properties factored **and** specialized behaviors available (multi-scale type control)

Image-Related classes

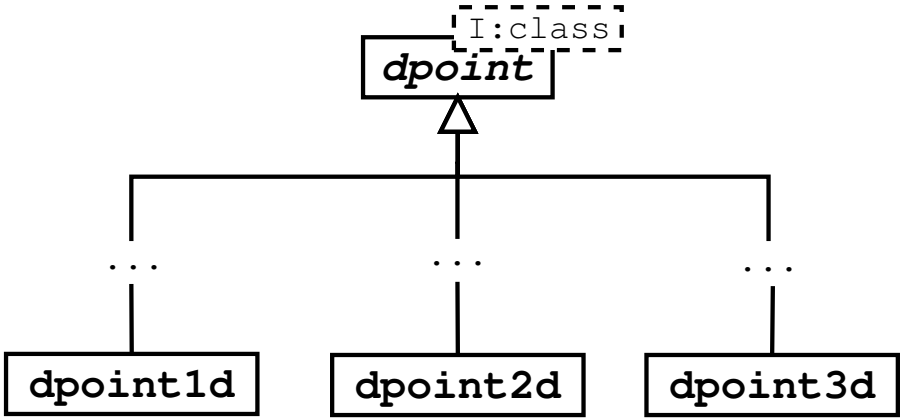
Points



- Point type associated to an image:
  - `point`
    typedef
  - `Point(I)`
    macro

```
Point(oln::image2d<oln::int_u8>) p;
 //  same as
oln::image2d<oln::int_u8>::point p;
 //  same as
oln::point2d p;
```

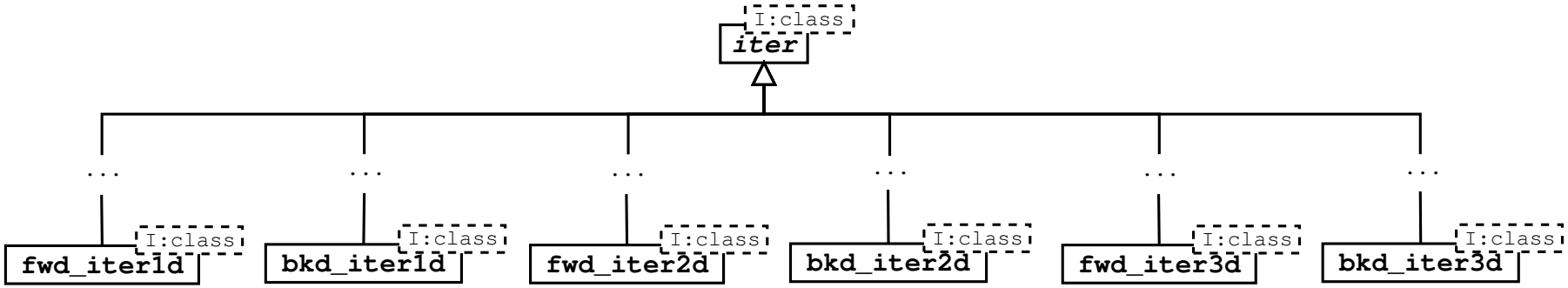
Delta-Points



- Delta-Point type associated to an image:
  - `dpoint`
    typedef
  - `DPoint(I)`
    macro

```
DPoint(oln::image2d<oln::int_u8>) dp;
 //  same as
oln::image2d<oln::int_u8>::dpoint p;
 //  same as
oln::dpoint2d p;
```

Iterators



- Iterators types associated to an image:
  - `Iter(I)` macro
  - `iter` typedef ⇔ `fwd_iter` typedef

- `bkd_iter`
  typedef

```
image2d<oln::int_u8> ima( //...) ;

 //  default  =  forward  iterators

 Iter (oln :: image2d<oln::int_u8>) it(ima);
 //  same as
oln :: image2d<oln::int_u8>:: iter   it (ima);
 //  same as
oln :: image2d<oln::int_u8>::fwd_iter  it (ima);
 //  same as
oln :: fwd_iter2d<> it(ima);

 //  backward iterators

oln :: image2d<oln::int_u8>::bkd_iter  it (ima);
 //  same as
oln :: bkd_iter2d<> it(ima);
```

How to create images ?

Constructors

Examples based on image2d

- Empty image (no allocated data):

```
image2d()
```

- Blank image :

```
image2d(coord nrows, coord ncols, coord border = 2)
image2d(const image2d_size& size, coord border = 2)
```

- Shallow Copy (no real data copy):

```
image2d(self& rhs)
```

- Creation by IO:

```
image2d(const io::internal::anything& r)
```

- **WARNING**:

```
image2d(const self& rhs); // without implementation
```

```
oln::image2d<oln::int_u8> ima; // empty

oln::image2d<oln::int_u8> ima(2, 2);  //  blank & default  border

oln::image2d_size s(2, 2);
oln::image2d<oln::int_u8> ima(s);  //  blank & default  border

oln::image2d<oln::int_u8> ima1(2, 2);
oln::image2d<oln::int_u8> ima(ima1); // shallow copy

oln::image2d<oln::int_u8> ima("lena.pnm") ;   //  IO
oln::image2d<oln::int_u8> ima = oln::io::load (" lena.pnm") ;   //  IO

const oln::image2d<oln::int_u8> ima1(2, 2);
oln::image2d<oln::int_u8> ima(ima1);  // /!\  won't compile !
```

Copies

- Shallow copies (no data copy)

```
self& operator=(self rhs) // shallow assignment
```

- Real copies

```
self clone() const // deep copy
```

```
oln::image2d<oln::int_u8> ima1(2, 2);

oln::image2d<oln::int_u8> ima2;

ima2 = ima1;  //  shallow assignment  /!\  no data copy

 //   different   from

ima2 = ima1.clone();  //  deep copy
```

How to manipulate images ?

Data access

- operator(// coords)
  - image1d
    ```
    T& operator()(coord row)
    ```
  - image2d
    ```
    T& operator()(coord row, coord col)
    ```
  - image3
    ```
    T& operator()(coord row, coord col, coord slice)
    ```
  - Read/write and read-only versions

- operator[// point]
  - image1d
    ```
    T& operator[](const point1d& p)
    ```
  - image2d
    ```
    T& operator[](const point1d& p)
    ```
  - image3d
    ```
    T& operator[](const point3d& p)
    ```
  - Read/write and read-only versions

```
oln::image2d<oln::int_u8> ima(10, 10);

oln::int_u8 i = ima(2, 2);
 // same as
oln::image2d<oln::int_u8>::point p(2, 2);
oln::int_u8 i = ima[p];
```

Useful informations retrieving

- data type:
  - `value`
    typedef
  - `Value(I)`
    macro
- dimension :
  - `enum {dim = Dim}`

```
 //  Data type

Value(oln::image2d<oln::int_u8>) v;
 //  same as
oln::image2d<oln::int_u8>::value v;
 //  same as
oln::int_u8  p;

 //  Dimension

unsigned dim = oln::image2d<oln::int_u8>::dim // 2
```

- Size informations (member functions):
  - image1d, image2d, image3d
    `image_size (typedef for getting size type)`
    `size()`
    `npoints()`
    `ncols()`
  - image2d, image3d
    `nrows()`
  - image3d
    `nslices()`

```
// Size informations

oln::image2d<oln::int_u8> ima(2, 3);

oln::coord ncols = ima.ncols(); // 2
oln::coord nrows = ima.nrows(); // 3

oln::size_t npoints = ima.npoints(); // 2 * 3

oln::image2d<oln::int_u8>::image_size size = ima.size(); // (2, 3)
oln::coord ncols = size.ncols(); // 2
// same as
oln::coord ncols = size.nth(0); // 2

oln::coord nrows = size.nrows(); // 3
// same as
oln::coord nrows = size.nth(1); // 3
```

- Border information
  - `border()` member function

```
oln::image2d<oln::int_u8> ima(2, 3, 42);

oln::coord border = ima.border(); // 42
```

Useful tools

- `mute`
  - ▷ util for image data conversion typing:
    ex: oln::image2d<oln::int_u8> → oln::image2d<oln::bin>

```
mute<oln::image2d<oln::int_u8>, oln::bin >:: ret&
algo_int_2_bin(oln :: image2d<oln::int_u8>& ima)
{
   //  treatment
}
```