POST-MORTEM ON A5

Prof. Chris Jermaine
cmj4@cs.rice.edu

A5: IDoubleMatrix

- How to handle column-major and row-major w/o dup. code?
- Two reasonable solutions I can come up with...
  - — Perhaps others have other solutions?

A5: Design One

- Put everything in abstract class, in terms of major and minor axes

```
class ADoubleMatrix implements ... {

     protected ADoubleMatrix (int majorAxisLen, int minorAxisLen, double default) {}

     protected IDoubleVector getMajorAxisVector (int pos) {}

     protected IDoubleVector getMinorAxisVector (int pos) {}

     protected void setMajorAxisVector (int pos, IDoubleVector setToMe) {}

     protected void setMinorAxisVector (int pos, IDoubleVector setToMe) {}

     protected double getEntry (int majorAxisPos, int minorAxisPos) {}

     // and so on...
}
```

A5: Design One

- Then all ops in concrete just call appropriate op in abstract

```
class RowMajorDoubleMatrix extends ... {

     public RowMajorDoubleMatrix (int numCols, int numRows, double default) {
          super (numRows, numCols, default);
     }

     public IDoubleVector getRow (int j) {
          return getMajorAxisVector (j)
     }

     public IDoubleVector getColumn (int i) {
          return getMinorAxisVector (i);
     }

     public void setRow (int j, IDoubleVector setToMe) {
          setMajorAxisVector (j, setToMe);
     }

     // and so on...
```

A5: Design One

- Everything is reversed in “ColumnMajorDoubleMatrix”

```
class ColumnMajorDoubleMatrix  extends ... {

     public RowMajorDoubleMatrix (int numCols, int numRows, double default) {
          super (numCols, numRows, default);
     }

     public IDoubleVector getRow (int j) {
          return getMinorAxisVector (j)
     }

     public IDoubleVector getColumn (int i) {
          return getMajorAxisVector (i);
     }

     public void setRow (int j, IDoubleVector setToMe) {
          setMinorAxisVector (j, setToMe);
     }

     // and so on...
```

A5: Design Two

- Abstract class is empty (or mostly empty)
- Implement one of the concrete classes directly
  - — For example, write “RowMajorDoubleMatrix” from scratch
- Then “ColumnMajorDoubleMatrix” is built on RowMajor

A5: Design Two

```
// the “real” implementation is here
class RowMajorDoubleMatrix  extends ... {

     // bunch of data structures here

     public RowMajorDoubleMatrix (int numCols, int numRows, double default) {
          // actual code here
     }

     public IDoubleVector getRow (int j) {
          // actual code here
     }

     public IDoubleVector getColumn (int i) {
          // actual code here
     }

     public void setRow (int j, IDoubleVector setToMe) {
          // actual code here
     }

     ...and so on...
}
```

A5: Design Two

```
// this one is just built upon the other
class ColumnMajorDoubleMatrix  extends ... {

     private RowMajorDoubleMatrix myData;

     public ColumnMajorDoubleMatrix (int numCols, int numRows, double default) {
          myData = new RowMajorDoubleMatrix (numRows, numCols, default);
     }

     public IDoubleVector getRow (int j) {
          return myData.getColumn (j);
     }

     public IDoubleVector getColumn (int i) {
          return myData.getRow (i);
     }

     public void setRow (int j, IDoubleVector setToMe) {
          myData.setColumn (j, setToMe);
     }

     ...and so on...
```

A5: What Not To Do

- Have an “if” in the abstract
  - — if (myType == “columnMajor”)
- Repeat code
  - — So same code in both concretes
  - — Possibly in different methods
- Have one concrete class extend another
  - — Can certainly get it to work, but it is strange and difficult to understand
  - — “Extends” means “is a”
  - — Is a RowMajorDoubleMatrix really a ColumnMajorDoubleMatrix?