# Post-Mortem on A5 and A7

**Prof. Chris Jermaine**
**cmj4@cs.rice.edu**

**Prof. Scott Rixner**
**rixner@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?

# A7

- Lots of possibilities for A7 design
- What’d I do?

# Most Fundamental Class: IMTreeNode

```
interface IMTreeNode <PointInMetricSpace extends IPointInMetricSpace
  <PointInMetricSpace>, DataType> {

  public IMTreeNodeABCD <PointInMetricSpace, DataType> insert
     (PointInMetricSpace keyToAdd, DataType dataToAdd);

  public ArrayList <DataWrapper <PointInMetricSpace, DataType>> find
     (Sphere <PointInMetricSpace> query);


  public Sphere <PointInMetricSpace> getBoundingSphere ();

  public int depth ();
}
```

# Then Actual MTree Code Is Easy

```
class MTree <PointInMetricSpace extends IPointInMetricSpace <PointInMetricSpace>,
     DataType> implements IMTree <PointInMetricSpace, DataType> {

  // this is the actual tree
  private IMTreeNode <PointInMetricSpace, DataType> root;

  // constructor... the two params set the number of entries in leaf and internal nodes
  public MTree (int intNodeSize, int leafNodeSize) {
    ...
    root = new LeafMTreeNode <PointInMetricSpace, DataType> ();
  }

  // insert a new key/data pair into the map
  public void insert (PointInMetricSpace keyToAdd, DataType dataToAdd) {

    // insert the new data point
    IMTreeNode <PointInMetricSpace, DataType> res = root.insert (keyToAdd, dataToAdd);

    // if we got back a root split, then construct a new root
    if (res != null) {
      root = new InternalMTreeNodeABCD <PointInMetricSpace, DataType> (root, res);
    }
  }

  public ArrayList <DataWrapper <PointInMetricSpace, DataType>> find
     (PointInMetricSpace query, double distance) {
    return root.find (new Sphere <PointInMetricSpace> (query, distance));
  }
}
```

# Both Nodes Built Around IMetricDataList

```
class InternalMTreeNode <PointInMetricSpace extends IPointInMetricSpace
  <PointInMetricSpace>, DataType> extends AMTreeNode <PointInMetricSpace, DataType> {

  // this holds all of the data in the internal node
  private IMetricDataList <PointInMetricSpace, Sphere <PointInMetricSpace>,
    IMTreeNode <PointInMetricSpace, DataType>> myData;

  ...
}


class LeafMTreeNodeABCD <PointInMetricSpace extends IPointInMetricSpace
  <PointInMetricSpace>, DataType>  extends AMTreeNode <PointInMetricSpace, DataType> {

  // this holds all of the data in the leaf node
  private IMetricDataList <PointInMetricSpace,
    Point <PointInMetricSpace>, DataType> myData;

  ...
}
```

- myData is the list of objects contained in the node
- Generic: can hold points in the case of leaf nodes
- Or spheres in the case of internal nodes
- Responsible for storing all children, and implementing the clustering algorithm

# IMetricDataList

```
interface IMetricDataList <PointInMetricSpace extends IPointInMetricSpace
  <PointInMetricSpace>, KeyType extends IObjectInMetricSpace <PointInMetricSpace>,
  DataType> {

  // add a new pair to the list
  public void add (KeyType keyToAdd, DataType dataToAdd);

  // replace a key at the indicated position
  public void replaceKey (int pos, KeyType keyToAdd);

  // get the key, data pair that resides at a particular position
  public DataWrapper <KeyType, DataType> get (int pos);

  // return the number of items in the list
  public int size ();

  // split the list in two, using some clutering algorithm
  public IMetricDataList <PointInMetricSpace, KeyType, DataType> splitInTwo ();

  // get a sphere that totally bounds all of the objects in the list
  public Sphere <PointInMetricSpace> getBoundingSphere ();
}
```

— Can use any KeyType as long as it is an IObjectInMetricSpace

— Much of MTree complexity is in this class; don't need to redo for int/leaf nodes

— Means we only have one clustering algorithm implementation

# IObjectInMetricSpace

```
interface IObjectInMetricSpace <PointInMetricSpace extends
  IPointInMetricSpace<PointInMetricSpace>> {

  // get the maximum distance from any point on the shape to a
  // particular point in the metric space
  double maxDistanceTo (PointInMetricSpace checkMe);

  // return some arbitrary point on the interior of the geometric object
  PointInMetricSpace getPointInInterior ();
}
```

- This has two implementations: points and spheres
- Clustering algorithm uses only “getPointInInterior”
- “add” updates bounding sphere using only “maxDistanceTo”

# So, To Recap

- MTree is really just a wrapper for a single IMTreeNode
- Two types of IMTreeNodes
  - LeafMTreeNode
  - InternalMTreeNode
- A split on insert returns a new IMTreeNode
- Most complexity associated w. node managed by IMetricDataList
  - This is a generic that handles clustering, bounding sphere
  - For both internal nodes and leaf nodes
- IMetricDataList is generic via use of IObjectInMetricSpace
  - Both Sphere and Point implement this

# Questions?