JAVA GENERICS (#2)

Prof. Chris Jermaine
cmj4@cs.rice.edu

Prof. Scott Rixner
rixner@cs.rice.edu

Let's Look At Another Generics Example

- One of most classic CS algorithms is "Dijkstra's algorithm"
- Used to solve single-source shortest path problem
  - — Say I have a bunch of objects ("vertices" or "nodes" in graph-speak)
  - — And pair-wise distances for each
  - — Goal is to find shortest path from source object to all others
  - — Runs in O(|E| + |V|log|V|) time with careful implementation
  - — |E| is number of pairwise distances, |V| number of objects
- I'll now give an outline of algorithm on the board
  - — Like all/most shortest path algorithms, relies on idea of "relaxation"
  - — Stores all objects in priority Q, sorted based on smallest known distance

Our Goal

- Implement Dijkstra's in a very generic way
- So it operates over a set of objects of any type
- And it can work with any distance measure
  - — Time, miles, weight, plain ints, etc.

We'll First Define the INumeric Generic

- Encapsulates the idea of a generic "distance"

```
interface INumeric <T> {
  T addTo (T toMe);
  boolean greaterThan (T me);
}
```

- What's the idea here?
  - — INumerics must be addable to themselves
  - — And comparable with themselves

Next is the IDstanceComputer

```
interface IDistanceComputer <T, N extends INumeric <N>> {
  N computeDistance (T fromMe, T toMe);
  N getHugeOne ();
  N getTinyOne ();
}
```

- This class is sort of a “factory” for INumerics
- It knows how to create tiny ones, and huge ones
- And it knows how to look at two T objects
  - — And compute the distance between them, returning it as an INumeric
- Question: why is IDistanceComputer separated out from T?

Now We Can Implement Dijkstra's

```
class Dijkstra <T, N extends INumeric <N>> {

  // lists all of the nodes we are computing over
  ArrayList <T> everyone;
  // used to compute distances
  IDistanceComputer <T, N> distanceFunc;
  // used to store the best distance for each object
  HashMap <T, N> distanceFromOrig = new HashMap <T, N> ();
  // the central priority queue used by the alg
  PriorityQueue <T> myQ =
    new PriorityQueue <T> (10, new ComparisonClass ());
```

Now We Can Implement Dijkstra's

```
class Dijkstra <T, N extends INumeric <N>> {

  ...
  // this is a "private inner class''
  // needed so we can get the priority queue to work
  private class ComparisonClass implements Comparator <T> {
    public int compare (T me, T withMe) {
      N distOne = distanceFromOrig.get (me);
      N distTwo = distanceFromOrig.get (withMe);
      if (distOne.greaterThan (distTwo))
        return 1;
      else if (distTwo.greaterThan (distOne))
        return -1;
      else
        return 0;
    }
  }
```

Now We Can Implement Dijkstra's

```
class Dijkstra <T, N extends INumeric <N>> {
  ...
  public N getDistanceFromOrigin (T forMe) {
    return distanceFromOrig.get (forMe);
  }

  public Dijkstra (IDistanceComputer <T, N> myComputer,
    ArrayList <T> myData) {
    distanceFunc = myComputer; boolean firstOne = true;
    for (T curNode : myData) {
      if (firstOne) {
        distanceFromOrig.put (curNode, distanceFunc.getTinyOne ());
        firstOne = false;
      } else {
        distanceFromOrig.put (curNode, distanceFunc.getHugeOne ());
      }
      myQ.add (curNode);
    }
    everyone = myData;
    runTheAlgorithm ();
  }
```

```
private void runTheAlgorithm () {
  // pull an item off the top of the priority queue
  for (T lowNode=myQ.poll(); lowNode!=null; lowNode=myQ.poll ()) {

    // look through everyone
    for (T curNode : everyone) {
      // get the current item's current distance
      N distance = distanceFromOrig.get (curNode);

      // get his relaxed distance
      N relaxedDistance = distanceFunc.computeDistance (lowNode,
        curNode).addTo (distanceFromOrig.get (lowNode));

      // if it better, then use it
      if (distance.greaterThan (relaxedDistance)) {
        myQ.remove (curNode);
        distanceFromOrig.put (curNode, relaxedDistance);
        myQ.add (curNode);
      }
    }
  }
}
```

To Use This? Easy

```
class IntDistance implements INumeric <IntDistance> {
  int val;
  ...
}

class IntDistanceComputer implements
  IDistanceComputer <Integer, IntDistance> {
  // gives an inf. distance to anything >= 10
}

// put 45, 34, 12, 25, 39, 56 into ArrayList <Integer> myData
Dijkstra <Integer, IntDistance> myAlgorithm = new Dijkstra
  <Integer, IntDistance> (new IntDistanceComputer (),
                          myData);
```

— Result is (45, 0), (34, 11), (12, big), (25, 20), (39, 6), (56, big)

One Final Note: Type Erasure

- For backwards compatibility,
  - — No JVM changes were made to support generics
  - — Only the compiler was changed
- Result is that inside of generic code,
  - — If you've got an object of a generic type
  - — Can only run those ops compiler knows are supported via runtime polymorphism
  - — Called "type erasure" 'cause Java doesn't remember generic type past compilation

```
class Dijkstra <T, N extends INumeric <N>> {

  private class ComparisonClass implements Comparator <T> {
    public int compare (T me, T withMe) {
      N distOne = distanceFromOrig.get (me);
      N distTwo = distanceFromOrig.get (withMe);
      if (distOne.greaterThan (distTwo)) ... // this is OK!
```

One Final Note: Type Erasure

- For backwards compatibility,
  - — No JVM changes were made to support generics
  - — Only the compiler was changed
- Result is that inside of generic code,
  - — If you've got an object of a generic type
  - — Can only run those ops compiler knows are supported via runtime polymorphism
  - — Called "type erasure" 'cause Java doesn't remember generic type past compilation

```
class Dijkstra <T, N extends INumeric <N>> {

  private class ComparisonClass implements Comparator <T> {
    public int compare (T me, T withMe) {
      N distOne = new N (); // this is not
      N [] distTwo = new N [100]; // neither is this
```

Questions?