Java Generics (#2) [Type Erasure]

Prof. Chris Jermaine
cmj4@cs.rice.edu

Type Erasure

- To discuss type erasure, will use simple, example parameterized type
- Encapsulates the idea of a generic “distance”

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

Type Erasure: What's the Deal?

- 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

What Is the Practical Effect?

```
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!
```

What Is the Practical Effect?

```
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
```

- Why?

How To Create Objcts of Parameter Type?

- Only way to create object is to include appropriate method in interface:

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

But This is Annoying

- Only way to create object is to include appropriate method in interface:

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

- Consider the following:

```
public N sumUp (ArrayList <N extends INumeric<N>> sumUs) {
  // what does this code look like?
}
```

But This is Annoying

- Only way is to include appropriate method in interface:

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

- Consider the following:

```
public N sumUp (ArrayList <N extends INumeric<N>> sumUs) {
  N returnVal = sumUs.get (0).getZero ();
  for (N n : sumUs)
    returnVal = returnVal.addTo (n);
  return returnVal;
}
```

— What's the problem here?

But This is Annoying

- Might we use static method to get around this?

```
interface INumeric <N> {
  N addTo (N toMe);
  boolean greaterThan (N me);
  static N getZero ();
}


public N sumUp (ArrayList <N extends INumeric<N>> sumUs) {
  N returnVal = N.getZero ();
  for (N n : sumUs)
    returnVal = returnVal.addTo (n);
  return returnVal;
}
```

Now what is the problem?

So Instead…

```
public N sumUp (ArrayList <N extends INumeric<N>> sumUs,
                N zero) {
  N returnVal = zero;
  for (N n : sumUs)
    returnVal = returnVal.addTo (n);
  return returnVal;
}
```

- Really need to pass as a parameter
- Not too elegant
- Question: why is `N returnVal = zero;` okay?

Questions?