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”

```java
interface ISummable <T> {
    void addYourselfTo (T addToMe);
    T getNothing ();
}
```
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
  — Can’t do things you’d want to do, like call “new”
    Requires type-specific constructor!

Called “type erasure” ‘cause Java doesn’t remember generic type past compilation
class SummableSet <T extends ISummable <T>> { 
    ArrayList <T> myData = new ArrayList <T> ();

    void addItem (T addMe) {
        myData.add (addMe);
    }
    T getSum () {
        T sum = null;
        for (T curItem : myData) {
            if (sum != null) {
                curItem.addYourselfTo (sum);
            } else {
                sum =
                    curItem.addYourselfTo (curItem.getNothng ());
            }
        }
        return sum;
    }
}
What Is the Practical Effect?

class SummableSet <T extends ISummable <T>> {
    ArrayList <T> myData = new ArrayList <T> ();

    void addItem (T addMe) {
        myData.add (addMe);
    }

    T getSum () {
        T sum = null;
        for (T curItem : myData) {
            if (sum != null)
                curItem.addYourselfTo (sum);
            else
                sum =
            curItem.addYourselfTo (new T ());
        }
        return sum;
    }
}
What Is the Practical Effect?

class SummableSet <T extends ISummable <T>> {
    ArrayList <T> myData = new ArrayList <T>();

    void addItem (T addMe) {
        myData.add (addMe);
    }

    T getSum () {
        T sum = new T ();
        for (T curItem : myData) {
            curItem.addYourselfTo (sum);
        }
        return sum;
    }

}
How To Create Objects of Parameter Type?

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

```java
interface ISummable <T> {
    void addYourselfTo (T addToMe);
    T getNothing () ;
}
```
But This is Annoying

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

```java
interface ISummable <T> {
    void addYourselfTo (T addToMe);
    T getNothing ();
}
```

• Consider the following... what is the problem?

```java
T getSum () {
    T sum = null;
    for (T curItem : myData) {
        if (sum != null)
            curItem.addYourselfTo (sum);
        else
            sum = curItem.addYourselfTo (curItem.getNothing ()�);
    }
    return sum;
}
```
But This is Annoying

• Might we use static method to get around this?

```java
interface ISummable <T> {
    void addYourselfTo (T addToMe);
    static T getNothing ();
}
```

class SummableSet <T extends ISummable <T>> {
    ArrayList <T> myData = new ArrayList <T> ();

    void addItem (T addMe) {
        myData.add (addMe);
    }
    T getSum () {
        T sum = T.getNothing ();
        for (T curItem : myData) {
            curItem.addYourselfTo (sum);
        }
        return sum;
    }
}

Now what is the problem?
class SummableSet <T extends ISummable <T>> {  
    ArrayList <T> myData = new ArrayList <T> ();  
    void addItem (T addMe) {
        myData.add (addMe);
    }
    T getSum (T zero) {
        T sum = zero;
        for (T curItem : myData) {
            curItem.addYourselfTo (sum);  
        }
        return sum;  
    }
}

• Really need to pass as a parameter
• Not too elegant
• Question: why is T sum = zero; okay?
Questions?