



Generics

L04, Sep 04, 2007

Roadmap

- Last time:
 - Programming with interfaces
- Today:
 - Notes on generics

Wisdom o' the day

From the 10 Commandments for C Programmers:

VII Thou shalt study thy libraries with care and strive not to reinvent them without cause, that thy code may be short and readable and thy days pleasant and productive.

-- Henry Spencer



Generics

The Generic worldview

- Constants: same program, same data
 - Commit to data *early*
- Variables: same program, different data
 - Commit to data *late*
- Dynamic data structures: different *amounts* of data
 - Commit to amounts *late*

The Generic worldview

- Problem: still have to commit to data *types* early
- Can't easily change data types on the fly
- Can't recycle a data structure to apply to different data types later
- Solution: generics
 - Data structures *parameterized by type*
 - Late commitment of data types
 - Write-once, use-many

The Generic worldview

Design principle: **Late commitment**

*Commit early for ease of coding;
Commit late for flexibility.*

Generics hints

- Arrays+generics = evil
- Java will not let you do:

```
public class Foo<T> {  
    private T[] _data;  
    public Foo(int size) {  
        _data=new T[size];  
    }  
}
```

- Advice: use an existing generic as your internal data store wherever possible

Role of Generics in Java

- Obviously, generic code (type independence)
- Clarify intent: restrict type

```
List<Integer> myIntList = new LinkedList<Integer>(); // 1'  
myIntList.add(new Integer(0)); //2'  
Integer x = myIntList.iterator().next(); // 3'
```

- **!!! Eliminate redundancy !!!**

Redundancy

```
// method printArray to print Integer array
public static void printArray( Integer[] inputArray
{
    // display array elements
    for ( Integer element : inputArray )
        System.out.printf( "%s ", element );

    System.out.println();
} // end method printArray

// method printArray to print Double array
public static void printArray( Double[] inputArray )
{
    // display array elements
    for ( Double element : inputArray )
        System.out.printf( "%s ", element );

    System.out.println();
} // end method printArray
....
```

Generic version

```
// method printArray to print Integer array
public static void printArray( Integer[] inputArray
{
    // display array elements
    for ( Integer element : inputArray )
        System.out.printf( "%s ", element );

    System.out.println();
} // end method printArray

// method printArray to print Double array
public static void printArray( Double[] inputArray )
{
    // display array elements
    for ( Double element : inputArray )
        System.out.printf( "%s ", element );

    System.out.println();
} // end method printArray
```

....

```
public static <E> void printArray( E[] inputArray )
{
    // display array elements
    for ( E element : inputArray )
        System.out.printf( "%s ", element );

    System.out.println();
} // end method printArray
```

Notes on Generics

- Know the difference between a generic class and a generic function
- You can have multiple generic types $\langle E, T \rangle$
- Nested generic classes have separate “name spaces”
- Bounded Type Parameters are Key (concepts)
 - `double sum(List<? extends Number> list){...}`
- There are sticky issues related to “erasure”

Concepts

- Interfaces + Generics allows development of consistent concepts

```
public interface Comparable<T> {  
    int compareTo( T obj );  
}
```

- Use accordingly:

```
public class MyThing implements Comparable<MyThing>  
{...};
```

```
public class Comparator<E extends Comparable<E> >  
{....}
```



Critiques

Procedure

- General procedure:
 - 1 person presents their code to a small group
 - Group members give feedback/suggestions/help find bugs/etc.

Feedback & Meta-feedback

- Not only to make code better
- Also make *you* better at giving and receiving feedback...
- Learn how to give good presentations & give good critiques
- I'll check up on you and make feedback notes
- You'll give feedback on each other's presentation style *and* on your crit feedbacks

Crit tips

- Code review
- Purpose is to make code better
- Not to rip presenter up, show off how smart you are, etc.
- Look at overall structure of design
- Drill down where necessary to pull details out
- Run code (if possible)
- (Maybe) Show bits of your code for comparison

Social issues

- Social interaction a critical part of working in teams
- (Yes, even for computer scientists...)
- *Say at least one good thing before saying anything negative*
- Phrase negatives as ways to improve, not “you’re stupid”
- More useful tips on group interactions:

<http://www.alice.org/Randy/teams.htm>