

Introduction to Computer Science II

CS112-2008S-24

Generics

David Galles

Department of Computer Science
University of San Francisco

24-0: ArrayList

- ArrayList stores Objects
 - Good, because we can store *anything* in the list
- Need to typecast the result of `it.next()` to what we want
- Compiler can't check for type errors – need to do it at runtime
- It would be nice if we could do the typechecking at compile-time instead of run-time

24-1: Generics

- We'd like to be able to create a specific kind of list
 - List of Strings
 - List of Integers
 - List of Creatures
- Not have to do any type casting
- Compiler would catch any errors that we make

24-2: Generics

```
ArrayList<Integer> intList;  
ArrayList<String> stringList;  
  
intList = new ArrayList<Integer>();  
stringList = new ArrayList<String>();  
  
intList.add(new Integer(3));  
intList.add(new Integer(4));  
stringList.add("foo");  
stringList.add("bar");  
  
String s = stringList.get(1);  
Integer i = intList.get(1);
```

24-3: Generics

- `intList` is not just an `ArrayList`, it is an `ArrayList` that can *only* hold integers
- No typecasting!
- Compiler finds errors!

24-4: Generics

```
ArrayList<Integer> intList;
```

```
intList = new ArrayList<Integer>();
```

```
intList.add(new Integer(3));
```

```
intList.add(new Integer(4));
```

```
intList.add("foo");    <== ERROR! Caught by compiler
```

```
Sring s = intList.get(1) <== ERROR! Caught by compiler
```

```
String s = (String) intList.get(1) <== ERROR! caught by compiler
```

24-5: Generics & Iterators

- Have an ArrayList of integers:
 - `ArrayList<Integer> al`
- Want an iterator to traverse this ArrayList
 - Like the `.next()` method of the iterator to return an Integer, not an Object
 - More Generics!

24-6: Generics & Iterators

```
ArrayList<Integer> al = new ArrayList<Integer>();  
al.add(new Integer(2));  
al.add(new Integer(3));  
al.add(new Integer(4));
```

```
Iterator<Integer> integerIt = al.iterator();  
while (integerIt.hasNext())  
{  
    Integer nextInt = integerIt.next();  
    // Do something with nextInt  
}
```

24-7: Generics

- Only need to write the `ArrayList` class once
- When we instantiate it, we say what type is to be stored in the list
- Don't need to typecast (since the compiler knows what types are stored in the list)
- Compiler checks for errors

24-8: Writing Generics

- See Eclipse code for Linked List version of generics
- Generics and Arrays don't get along very well ...
- Short answer on generics:
 - Using generics (built-in ArrayList, LinkedList, etc) fairly safe & easy
 - Writing generics (especially with arrays!) can be pretty complicated – lots of subtle issues

24-9: More using Generics

```
LinkedList<String> stringLL = new LinkedList<String>();  
stringLL.add("s1");  
stringLL.add("s2");  
stringLL.add("s3");
```

```
Iterator<String> stringIt = stringLL.iterator();  
while (stringIt.hasNext())  
{  
    String s = stringIt.next();  
    // Do something with s  
}
```

24-10: Iterators – Python review

- Recall how python loops work
- Iterate over elements of a data structure

```
for item in L:  
    print item
```

- If you want to iterate over integers, need to create a list first

```
for i in range(10):  
    print i
```

24-11: Iterators – Python review

- Java:

```
for (int i = 0; i < A.length; i++)  
    System.out.println(Integer.toString(i) + " " + A[i])
```

- Python:

24-12: Iterators – Python review

- Java:

```
for (int i = 0; i < A.length; i++)  
    System.out.println(Integer.toString(i) + " " + A[i])
```

- Python:

```
for i in range(len(L))  
    System.out.println i, A[i]
```

24-13: Java: for-each

- Java 1.5 allows for Python-style iterators
- C++ style iterators

```
ArrayList<String> myList = new ArrayList<String>(5);  
// Add elements to myList  
for (Iterator i = myList.iterator(); i.hasNext();)   
    System.out.println(i.next());
```

- Python style iterator, Using foreach construct:

```
ArrayList<String> myList = new ArrayList<String>(5);  
// Add elements to myList  
for (String s : myList)   
    System.out.println(s);
```

24-14: Java: for-each

- What's wrong with this code?

```
ArrayList<String> suits = new ArrayList<String>
ArrayList<String> ranks = new ArrayList<String>
// fill up suits with "diamonds", "hearts", etc
// fill up ranks with "A", "2", "3", etc

for (iterator<String> i = suits.iterator(); i.hasNext();)
    for (iterator<String> j = ranks.iterator(); j.hasNext();)
        deck.add(new Card(i.next(), j.next()));
```

24-15: Java: for-each

- Fix:

```
ArrayList<String> suits = new ArrayList<String>
ArrayList<String> ranks = new ArrayList<String>
// fill up suits with "diamonds", "hearts", etc
// fill up ranks with "A", "2", "3", etc

for (iterator<String> i = suits.iterator(); i.hasNext();)
    String suit = i.next();
    for (iterator<String> j = ranks.iterator(); j.hasNext();)
        deck.add(new Card(suit, j.next()));
```

24-16: Java: for-each

- Fix using foreach:

```
ArrayList<String> suits = new ArrayList<String>  
ArrayList<String> ranks = new ArrayList<String>  
// fill up suits with "diamonds", "hearts", etc  
// fill up ranks with "A", "2", "3", etc  
  
for (String suit : suits)  
    for (String rank : ranks)  
        deck.add(new Card(suit, rank));
```

24-17: Java: for-each

- Write a function that returns the sum of all elements in an array of integers

```
int sum(int A[])  
{  
    ...  
}
```

24-18: Java: for-each

- Write a function that returns the sum of all elements in an array of integers

```
int sum(int A[])
{
    result = 0;
    for (int i = 0; i < A.length; i++)
        result += A[i];
    return result;
}
```

- Can use foreach with arrays

24-19: Java: for-each

- Can use foreach with arrays:

```
int sum(int A[])
{
    result = 0;
    for (int elem : A)
        result += elem;
    return result;
}
```

24-20: Java: for-each

- When can you use for-each?
 - Arrays
 - Any collection that implements Iterable interface
 - Most all of the built-in collections
 - ArrayList
 - LinkedList
 - etc

24-21: MircoLab

- Create a class Card that contains a rank (integer, 1-13) and suit (string)
 - Give the Card a toString method
- Create an arrayList of Cards, using generics
- Add 52 cards to the arrayList
- Use forEach to print out all the cards in the deck