CS1007: Object Oriented Design and Programming in Java

Lecture #15

Nov 17

Shlomo Hershkop

shlomo@cs.columbia.edu

Outline

- Working with unknown objects
- Generic Objects
- Java Beans

- Reading: 7.3-7.8

Last time

- Working with objects
- Overview of types
- Comparing types
- .class object
- Shallow copy
- Deep copy
- Combination

Working with the unknown

- Generally when you have Object from some class,
  - you wrote it yourself, so have doc/source
  - Using standard library, have docs
  - Unknown class, have no idea how to:
    - Instantiated
    - Construct
    - If you don't know how to use, probably not a good idea to use ☺

Reflection

- Ability of running program to find out about its objects and classes
- Class object reveals
  - superclass
  - interfaces
  - package
  - names and types of fields
  - names, parameter types, return types of methods
  - parameter types of constructors

Reflection

- Class getSuperclass()
- Class[] getInterfaces()
- Package getPackage()
- Field[] getDeclaredFields()
- Constructor[] getDeclaredConstructors()
- Method[] getDeclaredMethods()

Enumerating Fields

- Print the names of all static fields of the Math class:

```
Field[] fields =
  Math.class.getDeclaredFields();
for (Field f : fields)
   if (Modifier.isStatic(f.getModifiers()))
      System.out.println(f.getName());
```

Enumerating Constructors

```
for (Constructor c : cons)
{
   Class[] params = cc.getParameterTypes();
   System.out.print("Rectangle(");
   boolean first = true;
   for (Class p : params)
   {
      if (first) first = false; else
  System.out.print(", ");
      System.out.print(p.getName());
   }
   System.out.println(")");
}
```

Output

```
Rectangle()
Rectangle(java.awt.Rectangle)
Rectangle(int, int, int, int)
Rectangle(int, int)
Rectangle(java.awt.Point,
  java.awt.Dimension)
Rectangle(java.awt.Point)
Rectangle(java.awt.Dimension)
```

Getting a single method descriptor

- Supply method name
- Supply array of parameter types
- Example: Get Rectangle.contains(int, int):

```
Method m =
  Rectangle.class.getDeclaredMethod(
    "contains", int.class, int.class);
```

- Example: Get default Rectangle constructor:

```
Constructor c =
  Rectangle.class.getDeclaredConstructor();
```

- getDeclaredMethod, getDeclaredConstructor are varargs methods

Invoking a Method

- Supply implicit parameter (null for static methods)
- Supply array of explicit parameter values
- Wrap primitive types
- Unwrap primitive return value
- Example: Call System.out.println("Hello, World") the hard way.

```
Method m =
  PrintStream.class.getDeclaredMethod(
    "println", String.class);
m.invoke(System.out, "Hello, World!");
```

- invoke is a varargs method

Inspecting Objects

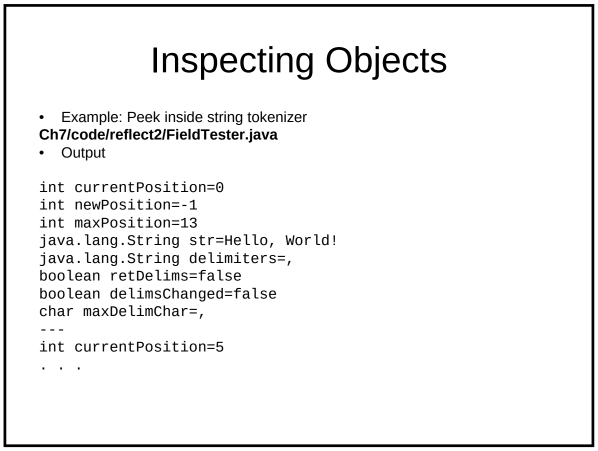
- Can obtain object contents at runtime
- Useful for generic debugging tools
- Need to gain access to private fields

```
Class c = obj.getClass();
Field f = c.getDeclaredField(name);
f.setAccessible(true);
```

- Throws exception if security manager disallows access
- Access field value:

```
Object value = f.get(obj);
f.set(obj, value);
```

- Use wrappers for primitive types



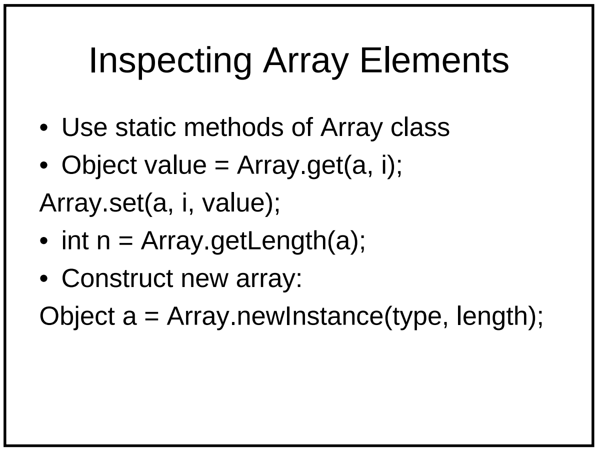
Inspecting Objects

- Example: Peek inside string tokenizer

Ch7/code/reflect2/FieldTester.java

- Output

```
int currentPosition=0
int newPosition=-1
int maxPosition=13
java.lang.String str=Hello, World!
java.lang.String delimiters=,
boolean retDelims=false
boolean delimsChanged=false
char maxDelimChar=,
---
int currentPosition=5
. . .
```



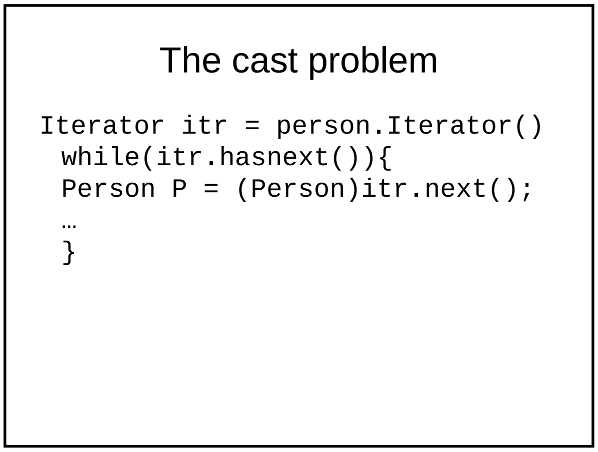
Inspecting Array Elements

- Use static methods of Array class
- Object value = Array.get(a, i);

Array.set(a, i, value);

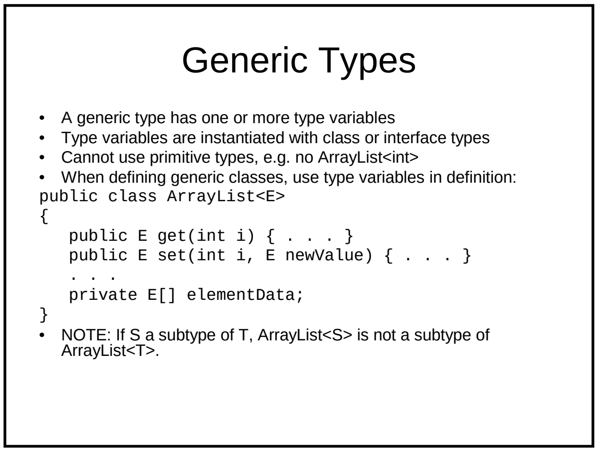
- int n = Array.getLength(a);
- Construct new array:

Object a = Array.newInstance(type, length);



The cast problem

```
Iterator itr = person.Iterator()
 while(itr.hasnext()){
 Person P = (Person)itr.next();
 …
 }
```

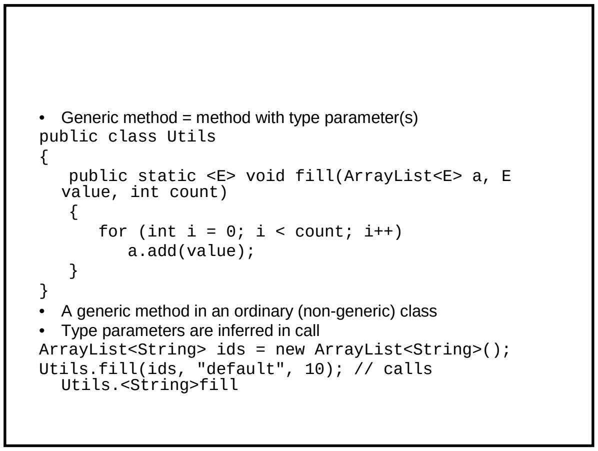


Generic Types

- A generic type has one or more type variables
- Type variables are instantiated with class or interface types
- Cannot use primitive types, e.g. no ArrayList<int>
- When defining generic classes, use type variables in definition:

```
public class ArrayList<E>
{
   public E get(int i) { . . . }
   public E set(int i, E newValue) { . . . }
   . . .
   private E[] elementData;
}
```

- NOTE: If S a subtype of T, ArrayList<S> is not a subtype of ArrayList<T>.

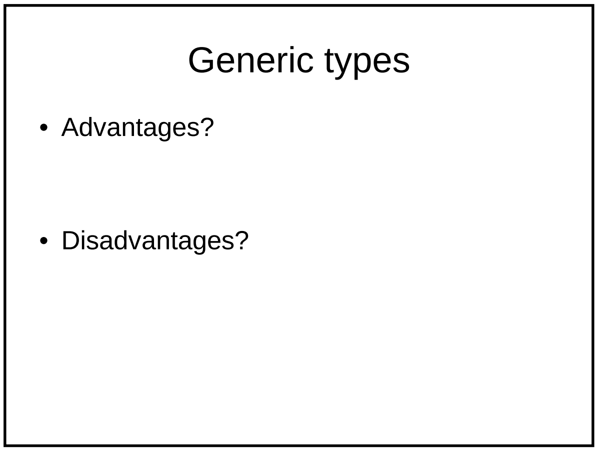


- Generic method = method with type parameter(s)

```
public class Utils
{
   public static <E> void fill(ArrayList<E> a, E
  value, int count)
   {
      for (int i = 0; i < count; i++)
         a.add(value);
   }
}
```

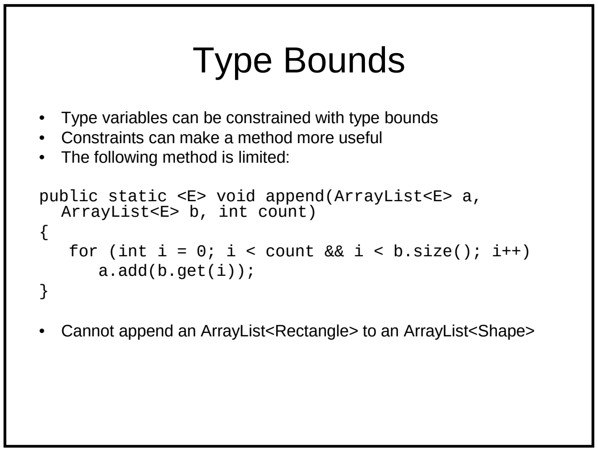
- A generic method in an ordinary (non-generic) class
- Type parameters are inferred in call

```
ArrayList<String> ids = new ArrayList<String>();
Utils.fill(ids, "default", 10); // calls
  Utils.<String>fill
```



Generic types

- Advantages?
- Disadvantages?

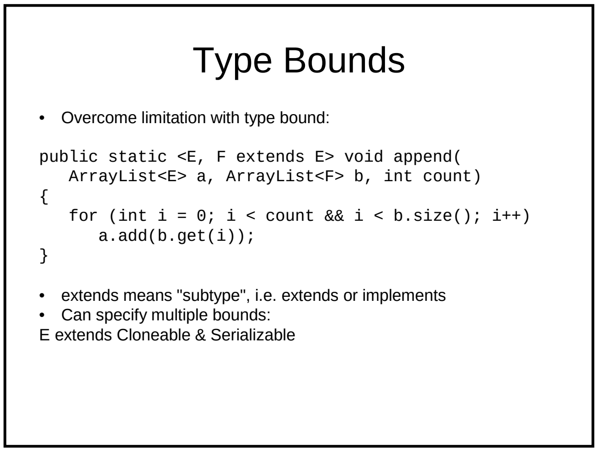


Type Bounds

- Type variables can be constrained with type bounds
- Constraints can make a method more useful
- The following method is limited:

```
public static <E> void append(ArrayList<E> a,
  ArrayList<E> b, int count)
{
   for (int i = 0; i < count && i < b.size(); i++)
      a.add(b.get(i));
}
```

- Cannot append an ArrayList<Rectangle> to an ArrayList<Shape>



Type Bounds

- Overcome limitation with type bound:

```
public static <E, F extends E> void append(
   ArrayList<E> a, ArrayList<F> b, int count)
{
   for (int i = 0; i < count && i < b.size(); i++)
      a.add(b.get(i));
}
```

- extends means "subtype", i.e. extends or implements
- Can specify multiple bounds:

E extends Cloneable & Serializable

Wildcards

- Definition of append never uses type F. Can simplify with wildcard:

```
public static <E> void append(
   ArrayList<E> a, ArrayList<? extends E> b, int
  count)
{
   for (int i = 0; i < count && i < b.size(); i++)
      a.add(b.get(i));
}
```

Wildcards

- Wildcards restrict methods that can be called:

ArrayList<? Extends E>.set method has the form

? extends E add(? extends E newElement)

- You cannot call this method!
- No value matches ? extends E because ? is unknown
- Ok to call get:

? extends E get(int i)

- Can assign return value to an element of type E

Wildcards

- Wildcards can be bounded in opposite direction
- ? super F matches any supertype of F

```
public static <F> void append(
  ArrayList<? super F> a, ArrayList<F> b, int count)
{
  for (int i = 0; i < count && i < b.size(); i++)
    a.add(b.get(i));
}
```

- Safe to call ArrayList<? super F>.add:

boolean add(? super F newElement)

- Can pass any element of type F (but not a supertype!)

- Typical example--start with

```
public static <E extends Comparable<E>> E
  getMax(ArrayList<E> a)
{
   E max = a.get(0);
   for (int i = 1; i < a.size(); i++)
      if (a.get(i).compareTo(max) > 0) max =
  a.get(i);
   return max;
}
```

- E extends Comparable<E> so that we can call compareTo

- Too restrictive--can't call with ArrayList<GregorianCalendar>
- GregorianCalendar does not implement Comparable<GregorianCalendar>, only Comparable<Calendar>
- Wildcards to the rescue:

```
public static <E extends Comparable<?
  super E>> E getMax(ArrayList<E> a)
```

Advantage/disadvantage

- Really good to move errors to compile time and not run time
- How to be backwards compatible?

Erasure

- Virtual machine does not know about generic types
- Type variables are erased--replaced by type bound or Object if unbounded
- Ex. ArrayList<E> becomes

```
public class ArrayList
{
   public Object get(int i) { . . . }
   public Object set(int i, Object newValue) { . . . }
   . . .
   private Object[] elementData;
}
```

- Ex. getmax becomes

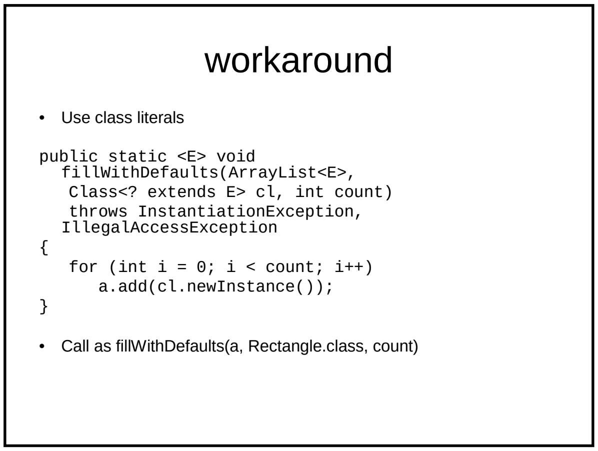
```
public static Comparable getMax(ArrayList a)
   // E extends Comparable<? super E> erased to Comparable
```

- Erasure necessary to interoperate with legacy (pre-JDK 5.0) code

Limitations of Generics

- Cannot replace type variables with primitive types
- Cannot construct new objects of generic type

a.add(new E()); // Error--would erase to new Object()

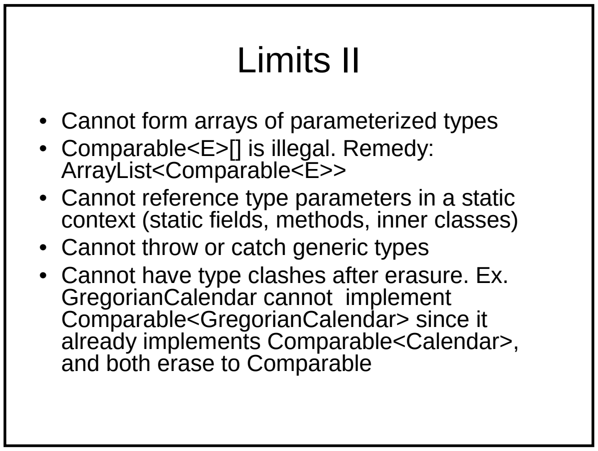


workaround

- Use class literals

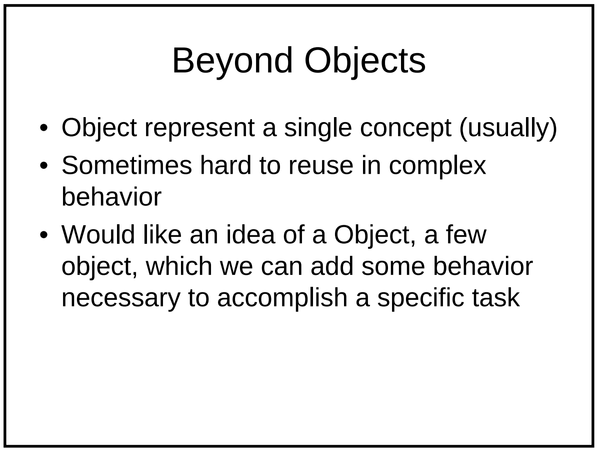
```
public static <E> void
  fillWithDefaults(ArrayList<E>,
   Class<? extends E> cl, int count)
   throws InstantiationException,
  IllegalAccessException
{
   for (int i = 0; i < count; i++)
      a.add(cl.newInstance());
}
```

- Call as fillWithDefaults(a, Rectangle.class, count)



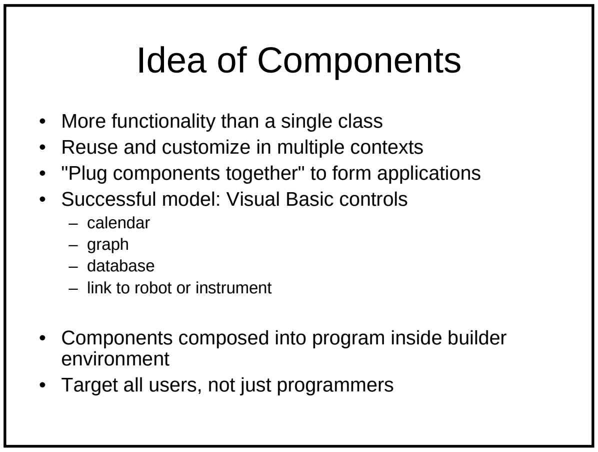
Limits II

- Cannot form arrays of parameterized types
- Comparable<E>[] is illegal. Remedy: ArrayList<Comparable<E>>
- Cannot reference type parameters in a static context (static fields, methods, inner classes)
- Cannot throw or catch generic types
- Cannot have type clashes after erasure. Ex. GregorianCalendar cannot implement Comparable<GregorianCalendar> since it already implements Comparable<Calendar>, and both erase to Comparable



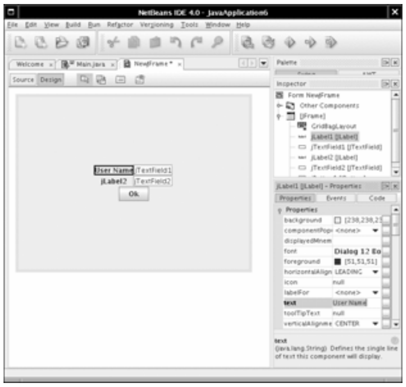
Beyond Objects

- Object represent a single concept (usually)
- Sometimes hard to reuse in complex behavior
- Would like an idea of a Object, a few object, which we can add some behavior necessary to accomplish a specific task



Idea of Components

- More functionality than a single class
- Reuse and customize in multiple contexts
- "Plug components together" to form applications
- Successful model: Visual Basic controls
  - calendar
  - graph
  - database
  - link to robot or instrument
- Components composed into program inside builder environment
- Target all users, not just programmers



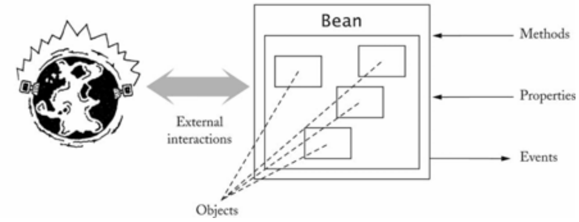
Properties

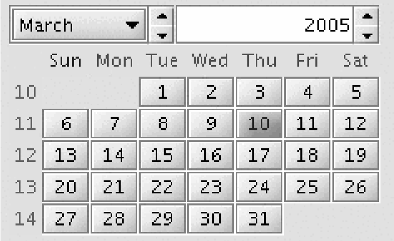
background	□ [238,238,2
componentPop	<none>
displayedMnem	
font	Dialog 12 Bo
foreground	■ [51,51,51]
horizontalAlign	LEADING
icon	null
labelFor	<none>
text	User Name
toolTipText	null
verticalAlignme	CENTER

text
(java.lang.String) Defines the single line of text this component will display.

Introducing Java Beans

- Java component model
- Bean has
  - methods (just like classes)
  - properties
  - events

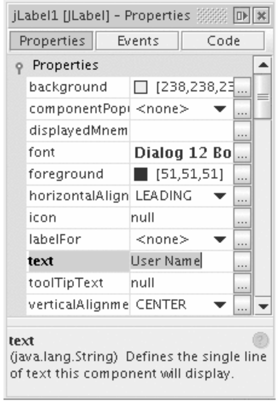




March 2005

	Sun	Mon	Tue	Wed	Thu	Fri	Sat
10			1	2	3	4	5
11	6	7	8	9	10	11	12
12	13	14	15	16	17	18	19
13	20	21	22	23	24	25	26
14	27	28	29	30	31		

Property sheet



Properties

background	□ [238,238,23
componentPopı	<none>
displayedMnem	
font	**Dialog 12 Bo**
foreground	■ [51,51,51]
horizontalAlign	LEADING
icon	null
labelFor	<none>
text	User Name
toolTipText	null
verticalAlignme	CENTER

text
(java.lang.String) Defines the single line of text this component will display.

Façade class

- Bean usually composed of multiple classes
- One class nominated as facade class
- Clients use only facade class methods

Next time

- Reading 7.8+, start 8-8.3