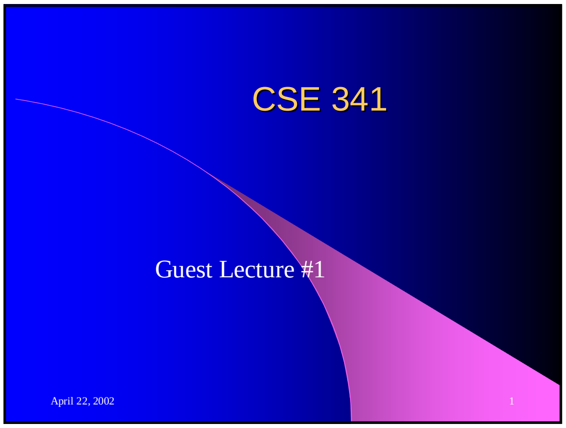
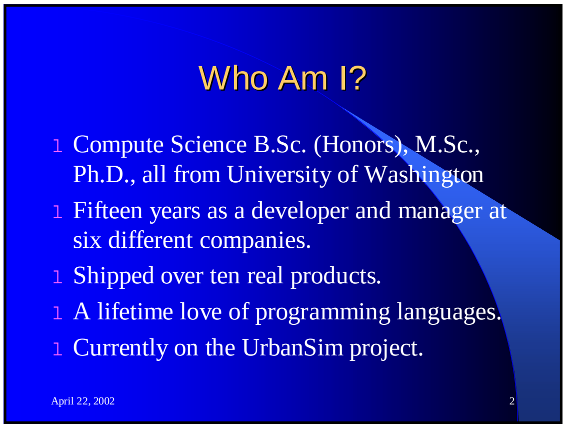


CSE 341

Guest Lecture #1

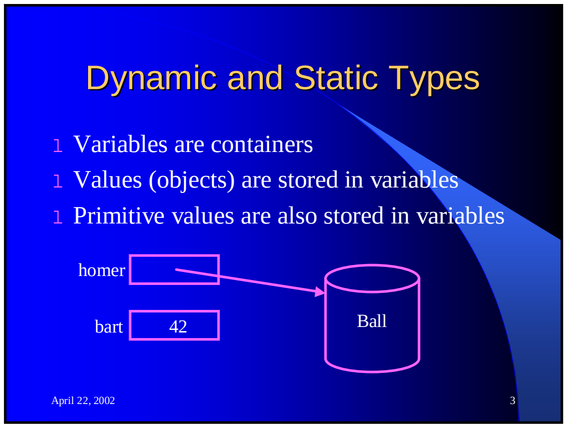
April 22, 2002 1



Who Am I?

- Compute Science B.Sc. (Honors), M.Sc., Ph.D., all from University of Washington
- Fifteen years as a developer and manager at six different companies.
- Shipped over ten real products.
- A lifetime love of programming languages.
- Currently on the UrbanSim project.

April 22, 2002 2

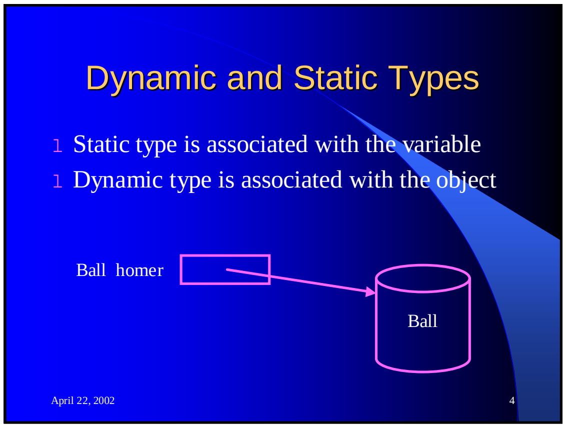


Dynamic and Static Types

- Variables are containers
- Values (objects) are stored in variables
- Primitive values are also stored in variables

homer
bart
42
Ball

April 22, 2002 3

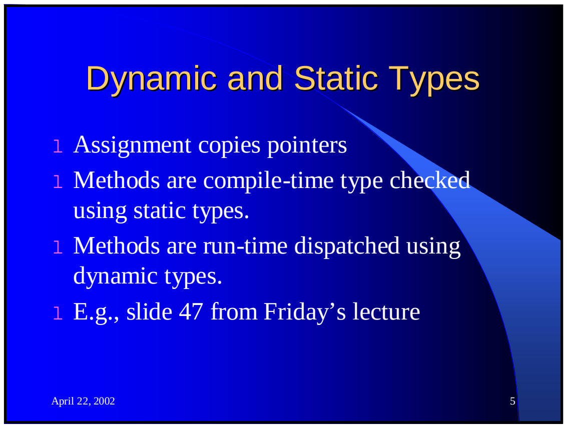


Dynamic and Static Types

- Static type is associated with the variable
- Dynamic type is associated with the object

Ball homer
Ball

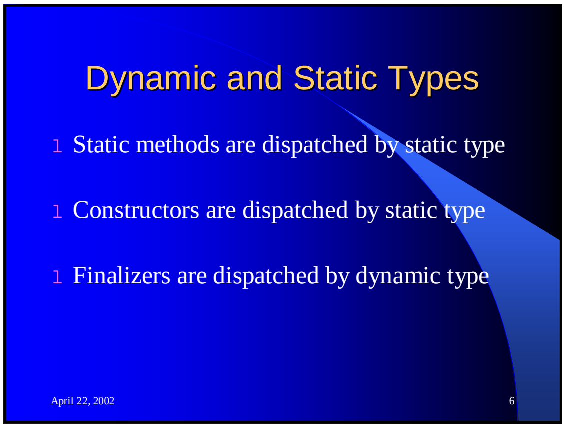
April 22, 2002 4



Dynamic and Static Types

- Assignment copies pointers
- Methods are compile-time type checked using static types.
- Methods are run-time dispatched using dynamic types.
- E.g., slide 47 from Friday’s lecture

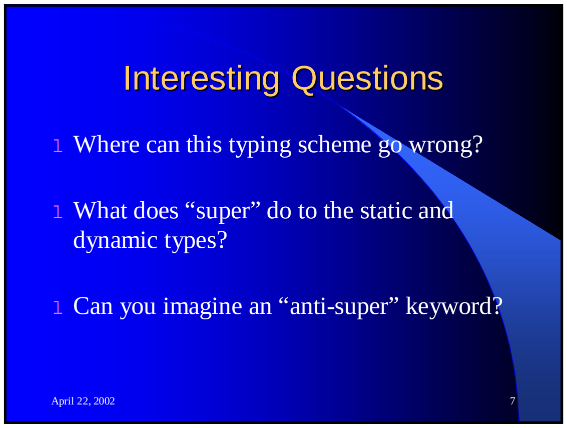
April 22, 2002 5



Dynamic and Static Types

- Static methods are dispatched by static type
- Constructors are dispatched by static type
- Finalizers are dispatched by dynamic type

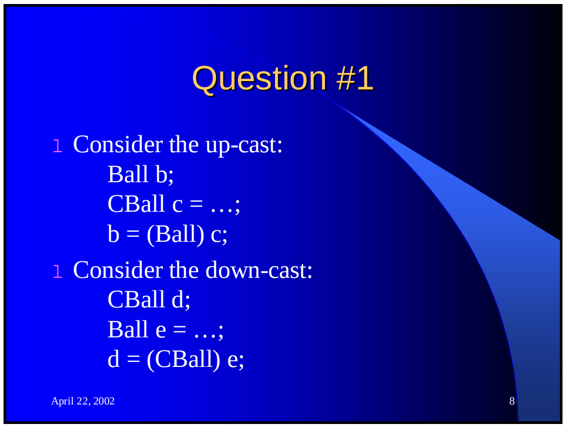
April 22, 2002 6



Interesting Questions

- Where can this typing scheme go wrong?
- What does “super” do to the static and dynamic types?
- Can you imagine an “anti-super” keyword?

April 22, 2002 7



Question #1

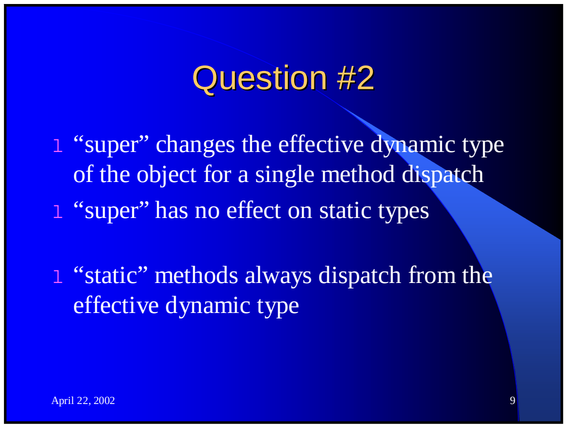
- Consider the up-cast:

```
Ball b;
CBall c = …;
b = (Ball) c;
```

- Consider the down-cast:

```
CBall d;
Ball e = …;
d = (CBall) e;
```

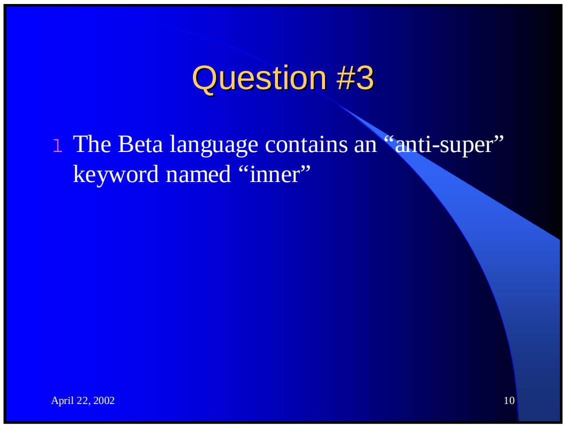
April 22, 2002 8



Question #2

- “super” changes the effective dynamic type of the object for a single method dispatch
- “super” has no effect on static types
- “static” methods always dispatch from the effective dynamic type

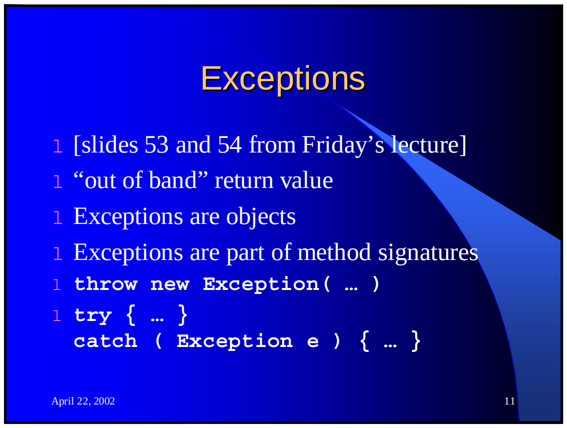
April 22, 2002 9



Question #3

- The Beta language contains an “anti-super” keyword named “inner”

April 22, 2002 10

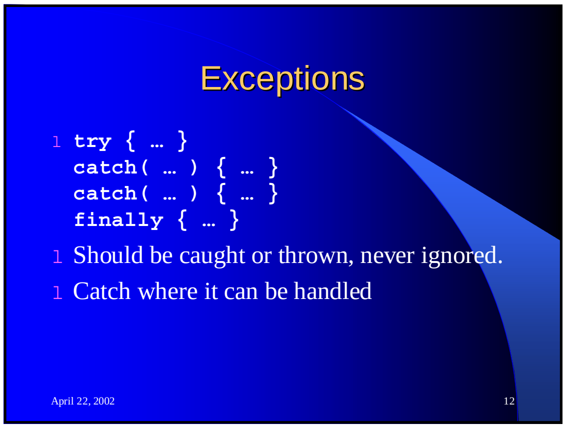


Exceptions

- [slides 53 and 54 from Friday’s lecture]
- “out of band” return value
- Exceptions are objects
- Exceptions are part of method signatures
- `throw new Exception( … )`

-
  ```
  try { … }
  catch ( Exception e ) { … }
  ```

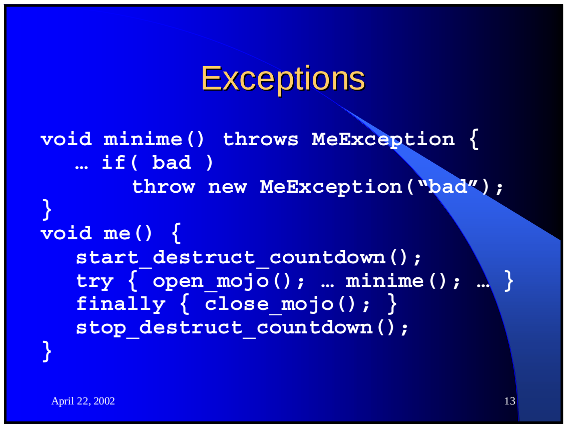
April 22, 2002 11



Exceptions

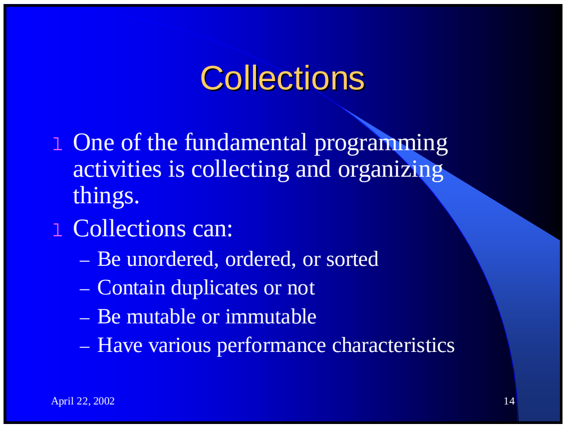
-
  ```
  try { … }
  catch( … ) { … }
  catch( … ) { … }
  finally { … }
  ```
- Should be caught or thrown, never ignored.
- Catch where it can be handled

April 22, 2002 12



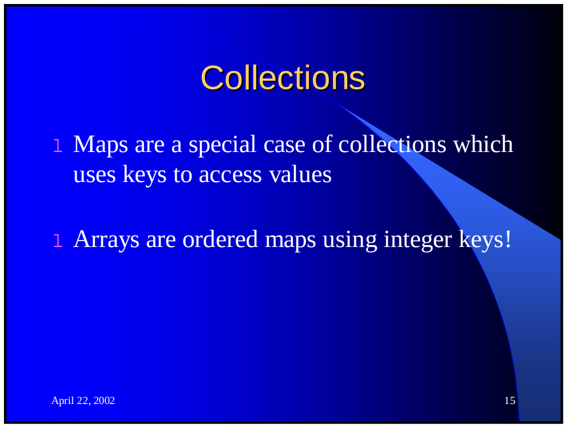
Exceptions

```
void minime() throws MeException {
   … if( bad )
        throw new MeException(“bad”);
}
void me() {
   start_destruct_countdown();
   try { open_mojo(); … minime(); … }
   finally { close_mojo(); }
   stop_destruct_countdown();
}
```



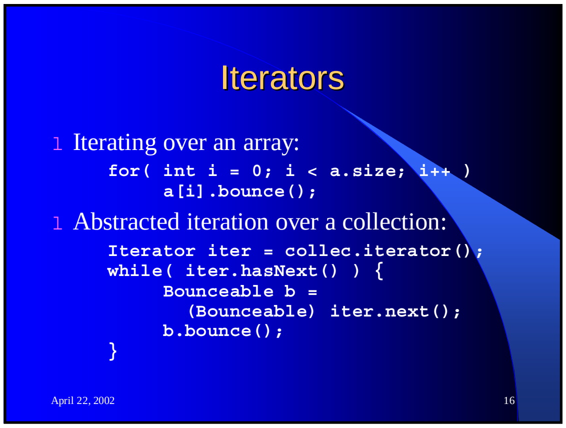
Collections

- One of the fundamental programming activities is collecting and organizing things.
- Collections can:
  - Be unordered, ordered, or sorted
  - Contain duplicates or not
  - Be mutable or immutable
  - Have various performance characteristics



Collections

- Maps are a special case of collections which uses keys to access values
- Arrays are ordered maps using integer keys!



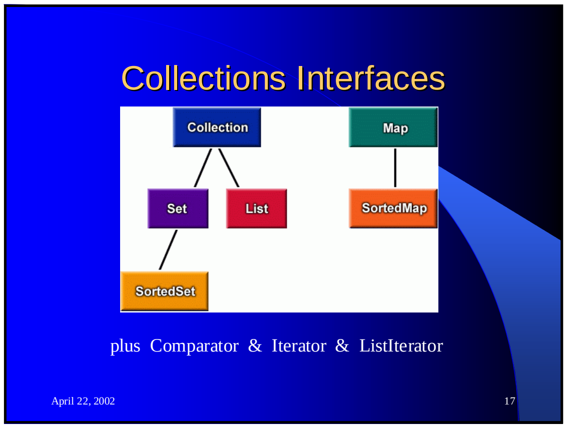
Iterators

- Iterating over an array:

```
for( int i = 0; i < a.size; i++ )
     a[i].bounce();
```

- Abstracted iteration over a collection:

```
Iterator iter = collec.iterator();
while( iter.hasNext() ) {
     Bounceable b =
        (Bounceable) iter.next();
     b.bounce();
}
```



Collections Interfaces

Collection

Set

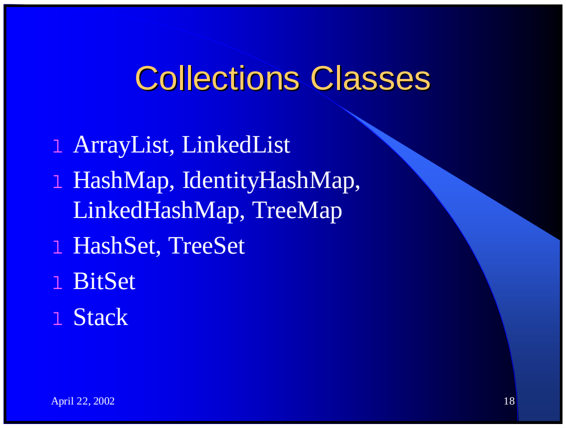
List

SortedSet

Map

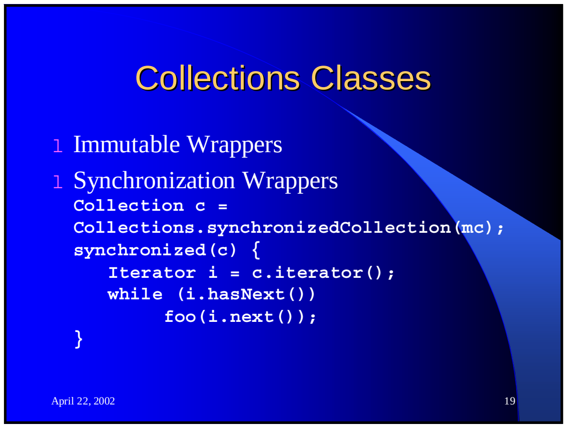
SortedMap

plus Comparator & Iterator & ListIterator



Collections Classes

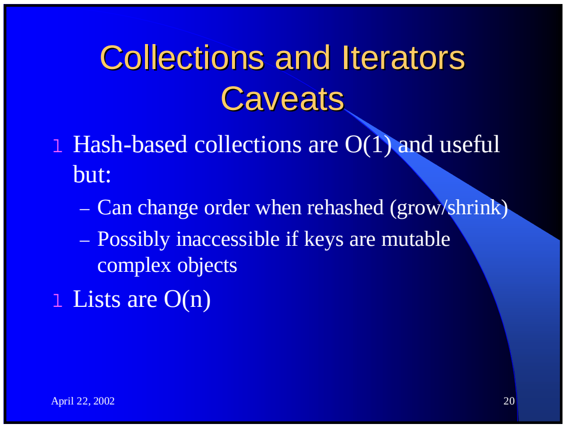
- ArrayList, LinkedList
- HashMap, IdentityHashMap, LinkedHashMap, TreeMap
- HashSet, TreeSet
- BitSet
- Stack



Collections Classes

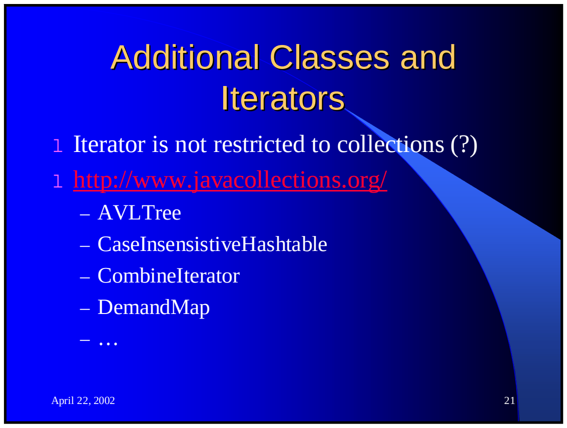
- Immutable Wrappers
- Synchronization Wrappers

```
Collection c =
Collections.synchronizedCollection(mc);
synchronized(c) {
   Iterator i = c.iterator();
   while (i.hasNext())
        foo(i.next());
}
```



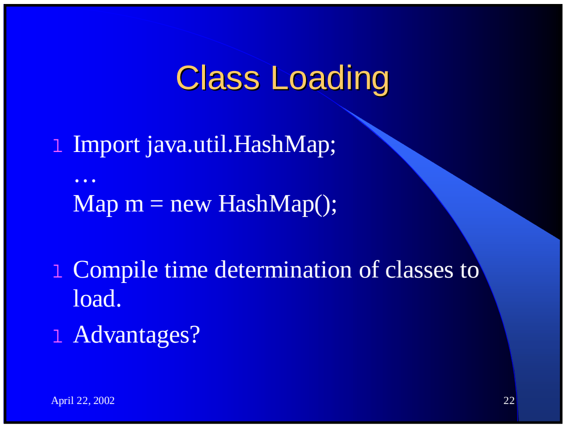
Collections and Iterators Caveats

- Hash-based collections are O(1) and useful but:
  - Can change order when rehashed (grow/shrink)
  - Possibly inaccessible if keys are mutable complex objects
- Lists are O(n)



Additional Classes and Iterators

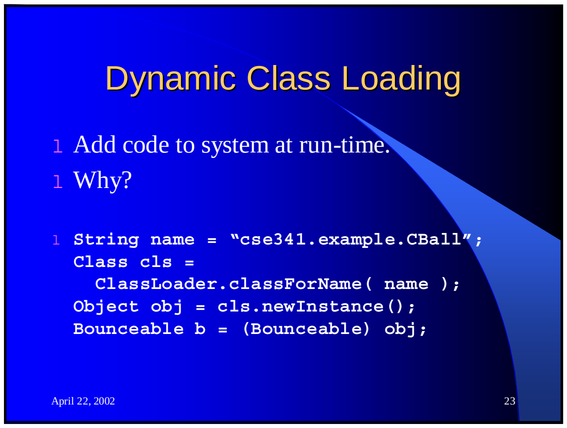
- Iterator is not restricted to collections (?)
- http://www.javacollections.org/
  - AVLTree
  - CaseInsensitiveHashtable
  - CombineIterator
  - DemandMap
  - …



Class Loading

- Import java.util.HashMap;
  …
  Map m = new HashMap();

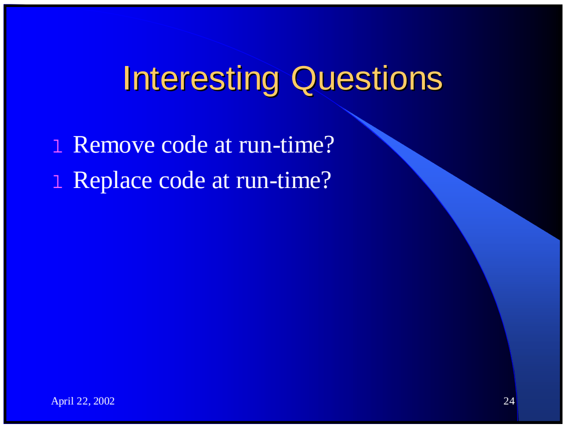
- Compile time determination of classes to load.
- Advantages?



Dynamic Class Loading

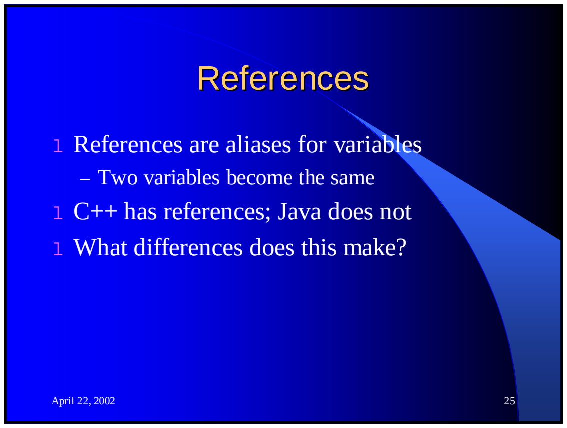
- Add code to system at run-time.
- Why?

```
String name = “cse341.example.CBall”;
Class cls =
  ClassLoader.classForName( name );
Object obj = cls.newInstance();
Bounceable b = (Bounceable) obj;
```



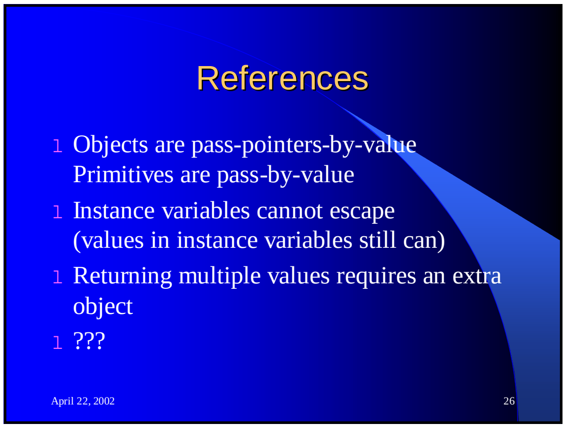
Interesting Questions

- Remove code at run-time?
- Replace code at run-time?



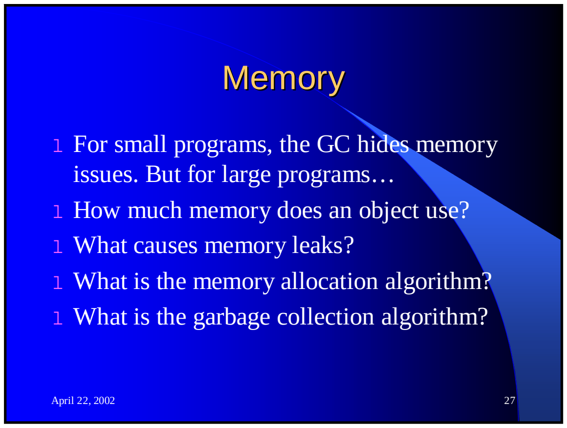
References

- References are aliases for variables
  - Two variables become the same
- C++ has references; Java does not
- What differences does this make?



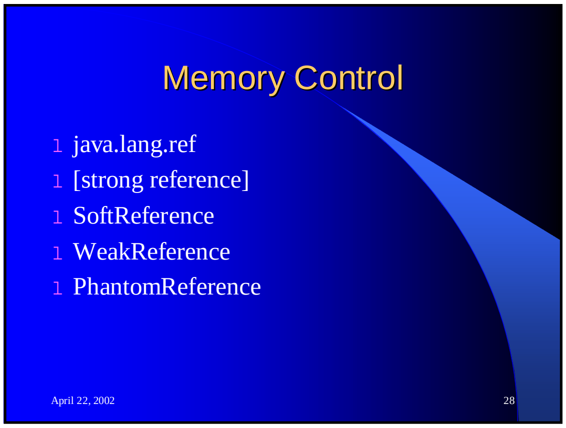
References

- Objects are pass-pointers-by-value
  Primitives are pass-by-value
- Instance variables cannot escape
  (values in instance variables still can)
- Returning multiple values requires an extra object
- ???



Memory

- For small programs, the GC hides memory issues. But for large programs…
- How much memory does an object use?
- What causes memory leaks?
- What is the memory allocation algorithm?
- What is the garbage collection algorithm?



Memory Control

- java.lang.ref
- [strong reference]
- SoftReference
- WeakReference
- PhantomReference