Who Am I?

- Computer 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.

Dynamic and Static Types

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

Dynamic and Static Types

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

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

Dynamic and Static Types

- Static methods are dispatched by static type
- Constructors are dispatched by static type
- Finalizers are dispatched by dynamic type
Interesting Questions
1. Where can this typing scheme go wrong?
2. What does “super” do to the static and dynamic types?
3. Can you imagine an “anti-super” keyword?

Question #1
1. Consider the up-cast:
   Ball b;
   CBall c = …;
   b = (Ball) c;
2. Consider the down-cast:
   CBall d;
   Ball e = …;
   d = (CBall) e;

Question #2
1. “super” changes the effective dynamic type of the object for a single method dispatch
2. “super” has no effect on static types
3. “static” methods always dispatch from the effective dynamic type

Question #3
1. The Beta language contains an “anti-super” keyword named “inner”

Exceptions
1. [slides 53 and 54 from Friday’s lecture]
2. “out of band” return value
3. Exceptions are objects
4. Exceptions are part of method signatures
5. throw new Exception( … )
6. try { … }
   catch ( Exception e ) { … }

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

```java
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

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

Collections

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

Collections Interfaces

```
Collection
    - Set
    - List
    - SortedMap
```

plus Comparator & Iterator & ListIterator

Collections Classes

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

Iterators

1. Iterating over an array:
   ```java
   for (int i = 0; i < a.size; i++)
       a[i].bounce();
   ```
2. Abstracted iteration over a collection:
   ```java
   Iterator iter = collec.iterator();
   while (iter.hasNext()) {
       Bounceable b = (Bounceable) iter.next();
       b.bounce();
   }
   ```
Collections Classes

1. Immutable Wrappers
2. Synchronization Wrappers
   Collection c = Collections.synchronizedCollection(mc);
   synchronized(c) {
       Iterator i = c.iterator();
       while (i.hasNext())
           foo(i.next());
   }

Collections and Iterators Caveats

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

Additional Classes and Iterators

1. Iterator is not restricted to collections (?)
   - AVLTree
   - CaseInsensitiveHashtable
   - CombineIterator
   - DemandMap
   - ...

Class Loading

1. Import java.util.HashMap;
   ...
   Map m = new HashMap();
2. Compile time determination of classes to load.
3. Advantages?

Dynamic Class Loading

1. Add code to system at run-time.
2. Why?
   String name = "cse311.example.CBall";
   Class cls =
       ClassLoader.classForName( name );
   Object obj = cls.newInstance();
   Bounceable b = (Bounceable) obj;

Interesting Questions

1. Remove code at run-time?
2. Replace code at run-time?
References

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

References

1. Objects are pass-pointers-by-value
   Primitives are pass-by-value
2. Instance variables cannot escape
   (values in instance variables still can)
3. Returning multiple values requires an extra object
4. ???

Memory

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

Memory Control

1. java.lang.ref
2. [strong reference]
3. SoftReference
4. WeakReference
5. PhantomReference