The Type Hierarchy and the Class Hierarchy

and what they have to do with subclasses, interfaces, and iterators
Why Types?

Interpret the meaning of an operation
- Does that + mean integer addition, floating point addition, or string concatenation?
- Is x.toString() the method defined in Foo.java or in Bar.java?

Prevent meaningless (dangerous) operations
- Applying integer addition to strings would create nonsense. Calling a method that doesn’t exist might be a security back door.

Static or dynamic checking – same purpose
Classes are Types

Every class in Java is also a type
Some types are not classes

The primitive types int, boolean, float, ...
Interfaces are types but not classes

Type checking includes both classes and other types
Subclasses and Subtypes

class AvailTime extends Agenda { ... }

Every AvailTime object “is-a” Agenda object
(AvailTime is a subtype of Agenda)

Any operation that is allowed on Agenda is also allowed on AvailTime
(even though it might do something different)
A hierarchy of types

class Foo { ... }
class Bar extends Foo { ... }
class Zot extends Foo { ... }
Class BarNone extends Bar { ... }

Diagram: 

Object

   Foo

      Bar

         BarNone

      Zot
Variables have *two* types

Foo b = new BarNone();

- *static type of b is Foo*
- *dynamic type of b is BarNone*

*OK because BarNone is a subtype of Foo.*
Covariance and Contravariance
(forget the words, don’t forget the concept)

```java
public Bar myMethod(Bar x) { ... }

...
Foo foo; Bar bar;
BarNone barNone;
barNone = bar.myMethod(foo); // WRONG
foo = bar.myMethod(barNone); // OK
```
Interfaces allow multiple supertypes (but still only one superclass)

class Bar extends Foo implements Iterable {
  ...
}
Iterators again ...

Java has a lot of “Collection” classes
Each has its own way of looping through the elements ... but that’s too hard to learn and remember

“Iterable” and “Iterator” (and “ListIterator”) allow many different implementations to provide a common, easy to remember interface
Many implementations of one interface