From STL to Java

- In STL an iterator is a concept, there are refinements
  - Input, output, forward, bidirectional, random access
    - A forward iterator is an input iterator and an output iterator
    - The iterator may be immutable (or const)---read only
  - Refinements not implemented by inheritance, but by design, contract, and subsequently implementation
    - What happens if you try to implement an STL iterator?
- In Java *Iterator* is an interface (like a base class), similar to Tapestry iterators
  - Collection(s) are required to have iterators, these are used in some operations like max, min, construct vector, …
  - Related to STL as algorithm glue, but very different

Wordlines.java, print strings, line #'s

```
public void print()
{
  Iterator allKeys = myMap.keySet().iterator();  // words

  while (allKeys.hasNext()) {
    String key = (String)allKeys.next();
    System.out.print(key + "\t");
    Iterator lines = ((Set)myMap.get(key)).iterator();
    while (lines.hasNext()) {
      System.out.print((Integer)lines.next() + " ");
    }
    System.out.println();
  }
}
```

- Differences between Java and Tapestry in practice?
  - Must store current element since `next()` does two things
  - Must cast since Collections store Objects

Interfaces, Comparator, Inner classes

- The `java.util.Comparator` interface is used in sorting
  - Different from the `java.lang.Comparable` interface?
  - What must be implemented?

- Suppose we want to change sort in WordLines
  - If we change keySet to entrySet what’s in ArrayList?
  - Program compiles/does not run sorting Map.Entry objects
    - How is this different from C++ behavior?

- How can we sort by size-of set while still sorting strings?
  - Use anonymous inner class that implements Comparable
  - Syntax is strange: create new interface
  - Access local variables, but some rules on parameters

Class and class design in Java

- Classes can be nested in Java
  - Inner class has access to an object’s internal state
  - Static Inner class doesn’t belong to an object
    - Similar to use of Node we’ve seen in C++ programs
    - Why should Node be nested, private?
  - We will see anonymous inner classes later