Objectives

- Collections
- Jar Files
- Compiled vs Interpreted

Review

- What interfaces/data structures have we been talking about in Java?
- Why do we use Interface objects instead of Implementations in our programs?

Review: Collections Framework

- **Interfaces**
  - Abstract data types that represent collections
  - Collections can be manipulated *independently* of implementation
- **Implementations**
  - Concrete implementations of the collection interfaces
  - Reusable data structures
- **Algorithms**
  - Methods perform useful computations on collections, e.g., searching and sorting
  - Polymorphic: same method can be used on many different implementations of collection interface
  - Reusable functionality

Traversing Collections

- For-each loop:

```
for (Object o : collection)
    System.out.println(o);
```

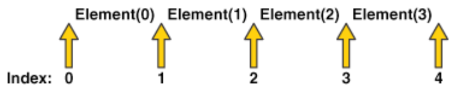
- Valid for all `Collections`
  - `Maps` (and its subclasses) are not `Collections`
  - But, Map's `keySet()` is a `Set` and `values()` is a `Collection`

Traversing Collections: `Iterator`

- Java Interface
- `<E> next()`
  - Get the next element
- `boolean hasNext()`
  - Are there more elements?
- `void remove()`
  - Remove the previous element
  - **Only safe way** to remove elements during iteration
    - Not known what will happen if remove elements in for-each loop

Iterator: Like a Cursor

- Always between two elements

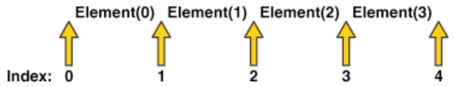


Polymorphic Filter Algorithm

```
static void filter(Collection c) {
   Iterator i = c.iterator();
   while( i.hasNext() ) {
     // if the next element does not
     // adhere to the condition, remove it
     if (!cond(i.next())) {
         i.remove();
     }
   }
}
```

Traversing Lists: `ListIterator`

- Methods to traverse list backwards
  - `listIterator(int position)`
  - Pass in `size()` as index to get at end of list
  - `hasPrevious()`
  - `previous()`
- Used for insertion/modification/deletion in linked lists in the middle



Element(0) Element(1) Element(2) Element(3)

Index: 0 1 2 3 4

Enumeration

- Legacy class
- Similar to `Iterator`
- `boolean hasMoreElements()`
- `Object nextElement()`
- Longer method names
- Doesn't have remove operation

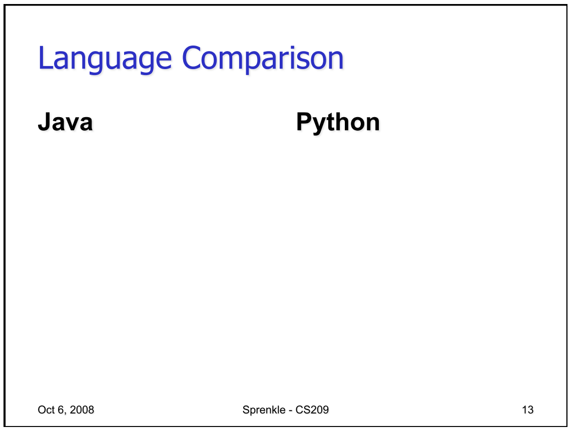
Synchronized Collection Classes

- For multiple threads sharing same collection
- Slow down typical programs
  - Avoid for now
- e.g., `Vector, Hashtable`
- See `java.util.concurrent`

Utility Class: `Collections`

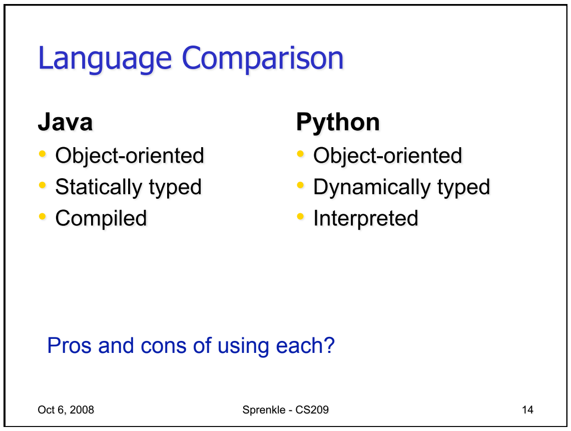
- Similar to `Arrays` class
- Contains methods for
  - Binary searching
  - Sorting
  - Min/max finding ("extremes")
  - Reversing
  - Shuffling
  - …

LANGUAGE COMPARISON



Language Comparison

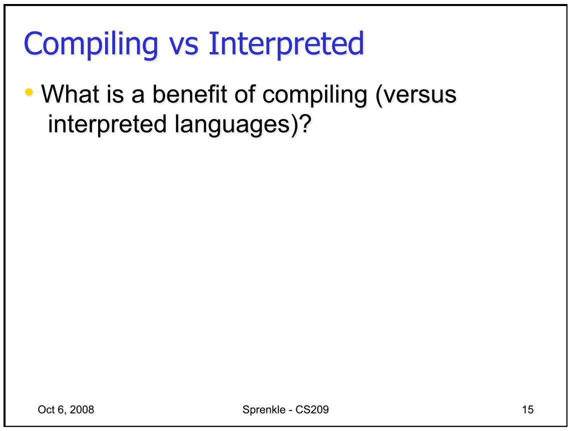
Java	**Python**



Language Comparison

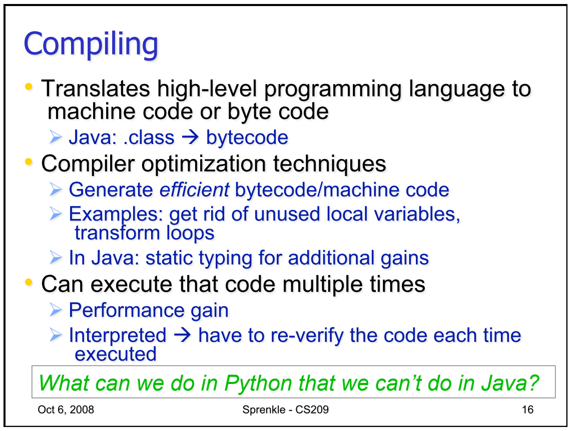
Java	**Python**
• Object-oriented	• Object-oriented
• Statically typed	• Dynamically typed
• Compiled	• Interpreted

Pros and cons of using each?



Compiling vs Interpreted

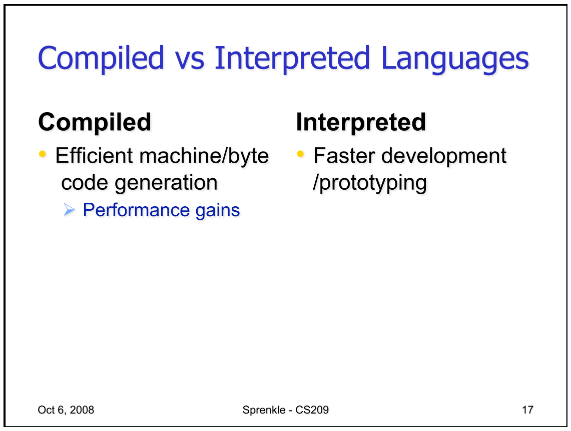
- What is a benefit of compiling (versus interpreted languages)?



Compiling

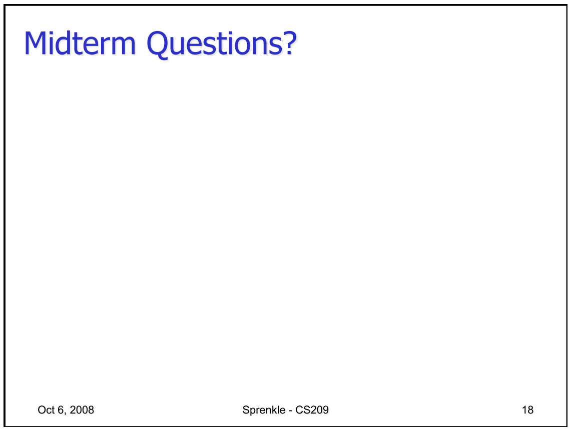
- Translates high-level programming language to machine code or byte code
  - ➢ Java: .class → bytecode
- Compiler optimization techniques
  - ➢ Generate *efficient* bytecode/machine code
  - ➢ Examples: get rid of unused local variables, transform loops
  - ➢ In Java: static typing for additional gains
- Can execute that code multiple times
  - ➢ Performance gain
  - ➢ Interpreted → have to re-verify the code each time executed

What can we do in Python that we can't do in Java?

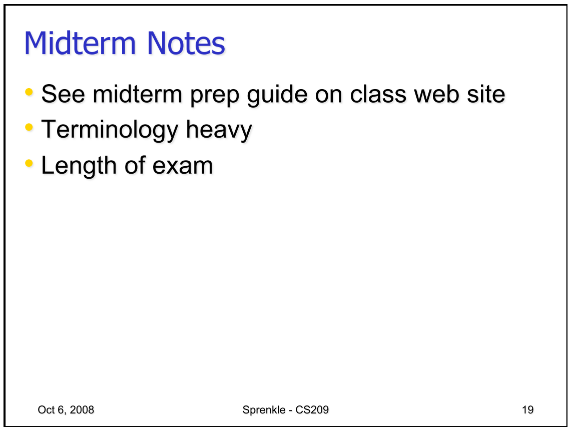


Compiled vs Interpreted Languages

Compiled	**Interpreted**
• Efficient machine/byte code generation ➢ Performance gains	• Faster development /prototyping



Midterm Questions?



Midterm Notes

- See midterm prep guide on class web site
- Terminology heavy
- Length of exam