Java topics that are central to this course

- Java generics
- Java interfaces
- Java iterators

Java Generics

- A **generic type** is a type that is not defined at class compilation time, but rather at class usage time
- Use: define a class in terms of one or more **type parameters**
- The type parameters come immediately after the class name, and appear in between < and >

**Example: Java ArrayList class**

- Designed to hold any type of data
- However, the data must be objects, not primitive types
- Use:

```java
ArrayList<String> names = new ArrayList<String>();
names.add("Jane Johnson");
names.add("John Jones");
names.add("Sara Smith");
System.out.println(names);

[Jane Johnson, John Jones, Sara Smith]
```

Generic classes and primitives types

- For each primitive type, there is a corresponding **wrapper class**
  - Example: Integer is the wrapper for `int`
  - Character, Double, Boolean
- Useful in using primitive types with generics
Example:

```java
ArrayList<Integer> numbers = new ArrayList<Integer>();
for (int i=0; i<5; i++)
    numbers.add(i);
if (numbers.contains(5))
    System.out.println("List contains 5");
```

- Example of defining a generic class

```java
public class MyList<T> {
    private T[] items = (T[]) new Object[100];
    private int size;

    public void add(T item) {
        items[size++] = item;
    }

    public String toString() {
        if (size == 0) return "[]";
        StringBuilder str = new StringBuilder("[");
        for (int i=0; i<size-1; i++)
            str.append(items[i] + ", ");
        str.append(items[size-1] + "]");
        return str.toString();
    }
}
```

Usage: same as above (more or less)

```java
MyList<String> names = new MyList<String>();
names.add("Jane Johnson");
names.add("John Jones");
names.add("Sara Smith");
System.out.println(names);

[Jane Johnson, John Jones, Sara Smith]
```

- Another example

```java
public class Pair<K,V> {
    private K key;
    private V value;

    public void set(K key, V value) {
        this.key = key;
        this.value = value;
    }

    public K getKey() { return key; }
```
public V getValue() { return value; }

public String toString() {
    return "[" + key + ", " + value + "]";
}

- **Example use of Pair**

  Maintain a list of employees, accessible by their SSNs

---

- **Java Interfaces**

  - In Java, an **interface** is a named collection of method definitions (without implementations)
  - An interface is a "contract" that a class signs -- it guarantees that it will respond to certain methods
  - To define an interface, syntax is

    ```java
    public interface <name> {} ...
    ```

  - Body consists of method **declarations**, not definitions
  - Use with classes:

    ```java
    public class <class-name> implements <interface-name> {} ...
    ```

  - A class can **implement** an interface (or more than one)
  - If so, body of the class **must** contain definitions for the methods declared in the interface

- **Example Interface: NumberSequence**

  - Definition

    ```java
    public interface NumberSequence {
        public boolean hasNext();
        public int next();
    }
    ```

  - Using it: CountDown, Primes
  - But what's the point? So far, we could write the same classes without using interfaces
• **Subtypes and Interfaces**
  
  o If a class implements an interface, it is called a *subtype* of the interface  
  o If a variable is declared to be a certain type, it can refer to an instance of any *subtype*  
  o Examples:
    
    ```java
    NumberSequence n = new Counter(10);
    ```
    
    o But you can't do this:
    
    ```java
    NumberSequence n = new NumberSequence(); // can't instantiate an interface
    ```
    
    o Allows client to "program to the interface"
    
    ```java
    List<Integer> numbers = new ArrayList<Integer>();
    numbers.add(1);
    System.out.println(numbers.contains(1));  // prints true
    ```

• **Example: Java Collections class**
  
  o Members operate on various types of Java collections  
  o Example: sort  
    
    ```java
    public static <T extends Comparable<T>> void sort(List<T> list);
    ```
    
    o Allows sorting of either *ArrayList* or *LinkedList* objects  
      ▪ Or a user-defined class that implements List

• **Java inner classes**
  
  o A class may defined inside another class  
  o Motivation: may not want to make a class accessible  
  o Example: MyList iterator

• **Anonymous inner classes**
  
  o Often, inner classes are only used once  
  o In data structures, the inner class often implements an interface  
  o Syntax:
    
    ```java
    new <X>() {
      < inner class member definitions >
    };
    ```
    
  o This creates an inner class (anonymous) that is a subtype of *X* as defined by the body in {...}, and an object that's an instance of the class
The **Iterable**<**T**> and **Iterator**<**T**> interface

```java
// there's more, but we'll ignore for now
public interface Iterable<**T**> {
    public Iterator<**T**> iterator( );
}

public interface Iterator<**T**> {
    public boolean hasNext( );
    public **T** next( );
    public void remove( );
}
```

A class must implement **Iterable** in order to use it in a **ForEach** construction. For example:

```java
List<String> names = new ArrayList<String>();
names.add("Jane Johnson");
names.add("John Jones");
names.add("Sara Smith");

for (String n : names)
    System.out.println(n);
```

Arrays are **Iterable**. For example,

```java
String names[ ] = {"Jane Johnson", "John Jones", "Sara Smith"};

// arrays are **Iterable**
for (String name : names)
    if (name.charAt(0) == 'J')
        System.out.println(name); // prints "Jane" and "John"
```

**Iterators and Anonymous inner classes**

- **MyList2.java**

**The Comparable**<**T**> interface
public interface Comparable<T> {
    public int compareTo(T o);
}

- Required for an ordered comparison of two objects
- Example:

  String s1 = "data", s2 = "structures",
  s3 = "computer", s4 = "computer";

  System.out.println(s1.compareTo(s2));  // a negative number
  System.out.println(s2.compareTo(s3));  // a positive number
  System.out.println(s3.compareTo(s4));  // zero

- compareTo requires its parameter to implement the Comparable interface

- The Comparator<T> interface

  - Useful when we wish to order or sort a collection in some way other than its "natural" ordering
  - Or, to be able to order in more than one way
  - Example: wiktionary.txt, used in WordCompletion application