**Interfaces**

- **interface**: A list of methods that a class promises to implement.
  - Interfaces give you an is-a relationship *without* code sharing.
    - Only method **stubs** in the interface
    - Allows object with no common ancestor to act same way
    - Object **can-act-as** any interface it **implements**
  - Analogous to non-programming idea of roles or certifications
    - "I'm certified as a CPA accountant. The certification assures you that I know how to do taxes, perform audits, and do management consulting."
    - "I can have many certifications, thus allowing me to do many things. I am both a CPA and a certified massage therapist"
English/Spanish Interpreter

To qualify, one needs to be able to:

- Convert English to Spanish
  - public String englishToSpanish(String english)
- Convert Spanish to English
  - public String spanishToEnglish(String spanish)

```java
public interface EnglishSpanishInterpreter {
    public String englishToSpanish(String english);
    public String spanishToEnglist(String spanish);
}
```

Sample English/Spanish Interpreters
Using an Interface

```java
public class PenelopeCruz implements EnglishSpanishInterpreter;
public class Chihuahua implements EnglishSpanishInterpreter;
public class C3PO extends Robot implements EnglishSpanishInterpreter;
EnglishSpanishInterpreter esi = any object that implements EnglishSpanishInterpreter
esi.englistToSpanish(“Hello World”);
```

When you declare a variable of an interface type, you are saying that you are **only** interested in the methods defined for that interface.

Implementing an interface

- A class can declare that it *implements* an interface.
  - This means the class contains an implementation for each of the method stubs in that interface.
    (Otherwise, the class will fail to compile. The method stubs are called "abstract methods")

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

- If we write a class that claims to be an EnglishSpanishInterpreter but doesn't implement the englishToSpanish and spanishToEnglish methods, it will not compile.

  Example:
  ```java
  public class Banana implements EnglishSpanishInterpreter {
      //without implementing methods
  }
  ```

  The compiler error message:
  ```
  Banana.java:1: Banana is not abstract and does not override abstract method
  in EnglishSpanishInterpreter
  ^
  ```

Comments about Interfaces

- The term interface refers to the set of public methods through which we can interact with objects of a class.

- Interfaces are used to define a contract for how you interact with an object, independent of the underlying implementation.

- Separate behavior (interface) from the implementation
Commonly used Java interfaces

- The Java class library contains classes and interfaces
- `Comparable` – allows us to order the elements of an arbitrary class
- `Serializable` (in `java.io`) – for classes whose objects are able to be saved to files.
- `List`, `Set`, `Map`, `Iterator` (in `java.util`) – describe data structures for storing collections of objects

```java
public interface Comparable<E> {
    public int compareTo(E other);
}
```

- A class can implement the `Comparable` interface to define a natural ordering for its objects.

- A call of `a.compareTo(b)` should return:
  - a value < 0 if `a` comes "before" `b` in the ordering,
  - a value > 0 if `a` comes "after" `b` in the ordering,
  - or 0 if `a` and `b` are considered "equal" in the ordering.
compareTo tricks

- delegation trick - If your object's fields are comparable (such as strings), you can use their `compareTo` results:

```java
// sort by employee name
public int compareTo(StaffMember other) {
    return name.compareTo(other.getName());
}
```

Comparable and sorting

- The `Arrays` class in `java.util` has a (static) method `sort` that sorts the elements of an array if they implement `Comparable`

```java
StaffMember[] staff = new StaffMember[4];
staff[0] = new Executive(...);
staff[1] = new Employee(...)
staff[2] = new Hourly(...);
staff[3] = new Volunteer(...);
Arrays.sort(staff);
```
ArrayList

- The ArrayList declaration:
  ```java
  public class ArrayList<E> implements List<E>...
  ```

  The List interface includes:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E get(int index)</td>
<td>Returns the element at the specified position</td>
</tr>
<tr>
<td>int indexOf(Object o)</td>
<td>Returns the index of the first occurrence of the specified element</td>
</tr>
<tr>
<td>E remove(int index)</td>
<td>Removes the element at the specified position</td>
</tr>
<tr>
<td>E set(int index, E element)</td>
<td>Replaces the element at the specified position</td>
</tr>
</tbody>
</table>