Interfaces
Relatedness of types

- Consider the task of writing classes to represent tour guides such as LasVegasTourGuide, ParisTourGuide, and UMStudentGuide.
- There are certain attributes or operations that are common to all tour guides: greeting, attractions, direct visitors, etc.
- By being a tour guide, you promise that you can implement those methods, but each tour guide computes them differently.
Interface as a contract

- Analogous to the idea of roles or certifications in real life:
  - "I'm certified as a CPA accountant. The certification assures you that I know how to do taxes, perform audits."

Compare to:
- "I'm certified as a tour guide. That means you can be sure that I know how to compute conduct a tour."
The attractions and greetings of tour guides

- **LasVegasTourGuide**:
  - attractions = *Bellagio, Venetian, etc.*
  - greeting = "Welcome to LasVegas"

- **ParisTourGuide**:
  - attractions = *Eiffel tower, Napoleon’s tomb, etc.*
  - greeting = “Bonjour, mes amis! Bienvenue Paris!”

- **UMTourGuide**
  - attractions = *Student Union, North Gym, etc.*
  - greeting = “Hey everybody — welcome to Maryland”
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
  - Object **can-act-as** any interface it **implements**
  - A LasVegasTourGuide object can be treated as a Tourguide as long as it implements the interface.
public interface TourGuide {
    public void sayGreeting();
    public String[] listAttractions();
    public void directVisitorsTo(String attraction);
    public void describe();
    public void sayGoodbye();
}
Java Interfaces

- An interface for tourguides:

```java
public interface TourGuide {
    public void sayGreeting();
    public String[] listAttractions();
    public void directVisitorsTo(String attraction);
    public void describe();
    public void sayGoodbye();
}
```

This interface describes the features common to all tour guides.

- Interface declaration syntax:

```java
public interface <name> {
    public <type> <name>(<type> <name>, ..., <type> <name>);
    public <type> <name>(<type> <name>, ..., <type> <name>);
    ...
    public <type> <name>(<type> <name>, ..., <type> <name>);
}
```

- All methods are public!
public class ParisTourGuide implements TourGuide {

    public void sayGreeting() {
        System.out.println("Bonjour,….");
    }

    public String[] listAttractions() {
        String[] attractions = {
            "Eiffel Tower", ...
        };
    }

    ....
}

Implementing an interface

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

- Syntax for implementing an interface
  ```java
  public class <name> implements <interface name> {
    ...
  }
  ```
Requirements

- If we write a class that claims to be a TourGuide but doesn't implement the interface methods, it will not compile.
  
  - Example:
    ```java
    public class Banana implements TourGuide {
        // without implementing any of the methods
    }
    ```

  - The compiler error message:
    ```plaintext
    Banana.java:1: Banana is not abstract and does not override abstract method sayGreeting() in TourGuides
    public class Banana implements TourGuide {
    ^
    ```