Why we need Interface

- Java does not support multiple inheritance
  - Inheritance gives us two thing:
    - Code reuse
    - Ability to represent the object polymorphically

- Interface support the concept of multiple inheritance

What is Interface?

- An interface in the java programming language is an abstract type that is used to specify an interface that classes must implement.

- An interface is a special type of “class” where methods and attributes are implicitly **public**
  - Attributes are implicitly **static and final**
  - Methods are **implicitly abstract** (no body)
  - Cannot be instantiated (no new)
  - Can be used to define references

What is an Interface?

- An interface is a special type of class
- Interfaces are declared using the "interface" keyword
- Interfaces are implemented by classes using the "implements" keyword.
- **Interface is a collection of abstract methods and constants.**

Declaring an Interface

In Turn.java:

```
public interface Turn
{
  public void turnLeft(int degrees);
  public void turnRight(int degrees);
}
```

Abstract methods (no body)

When a class "implements" an interface, the compiler ensures that it provides an implementation for all methods defined within the interface.

In Car.java:

```
 public class Car extends Vehicle implements Turn
 {
   public void turnLeft(int degrees)
   {
        [...]
   }

   public void turnRight(int degrees)
   {
        [...]
   }
```

Implementing Interfaces

- A Class can only inherit from one super class. However, a class may implement several Interfaces
  - The interfaces that a class implements are separated by commas
- Any class which implements an interface must provide an implementation for all methods defined within the interface.
  - NOTE: if an abstract class implements an interface, it NEED NOT implement all methods defined in the interface. HOWEVER, each concrete subclass MUST implement the methods defined in the interface.
- Interfaces can extends multiple interfaces. But can not extends a class.

Declaring an Interface

In Car.java:

```
public class Car extends Vehicle implements Turn, Driveable
{
  public int turnLeft(int degrees)
  {
       [...]
  }

  public int turnRight(int degrees)
  {
       [...]
  }

  // implement methods defined within the Driveable interface
```

Inheriting Interfaces

- If a superclass implements an interface, it's subclasses also implement the interface

```
public abstract class Vehicle implements Turn
{
  private String make;
  [...]
```

Vehicle
- make: String
- model: String
- tireCount: int

```
public class Car extends Vehicle
{
  private int trunkCapacity;
  [...]
```

Car
- trunkCapacity: int

Truck
- bedCapacity: int

```
public class Truck extends Vehicle
{
  private int bedCapacity;
  [...]
```

Multiple Inheritance?

- Some people (and textbooks) have said that allowing classes to implement multiple interfaces is the same thing as multiple inheritance
- This is NOT true. When you implement an interface:
  - The implementing class does not inherit instance variables
  - The implementing class does not inherit methods (none are defined)
  - The Implementing class does not inherit associations
- Implementation of interfaces is not inheritance. An interface defines a list of methods which must be implemented.

Interfaces as Types

- When a class is defined, the compiler views the class as a new type.
- The same thing is true of interfaces. The compiler regards an interface as a type.
  - It can be used to declare variables or method parameters

```
int i;
Car myFleet[];
Turn anotherFleet[];

[...]

  myFleet[i].start();

  anotherFleet[i].turnLeft(100);
  anotherFleet[i+1].turnRight(45);
```

Example

```
Public interface Area // interface methods and attributes are
 {                                    //implicitly public
     final float PI = 3.14F; // static and final
     float compute (float x, float y);    }
```

Public class Circle implements Area **{** **public float compute (float x, float y)** **{** **return PI * x * y;** **}** **}**	**Public class Rectangle implements Area** **{** **public float compute (float x, float y)** **{** **return x * y;** **}** **}**

Abstract Class vs. Interface

• An abstract class can not be instantiated. • A concrete sub class of an abstract class must define all the inherited abstract methods. • A class can extend another class. A subclass can add methods and override some of its super class's methods. • A class can extend only one class. • A class can have fields. •A class defines its own constructor. • An abstract class has one or more abstract method.	• Interface can not be instantiated. • A concrete class that implements an interface must define all the methods specified by the interface. • An interface can extend another interface. • A class can implement any number of interface. • An interface can not have field. • An interface has no constructor. • All methods of interface are abstract by default.

Abstract Class vs. Interface

• **Every class is a part of hierarchy of classes with object at the top.**	• **An interface may belongs to a small hierarchy of interfaces, but this is not as common.**

Abstract Classes Versus Interfaces

- When should one use an Abstract class instead of an interface?
  - If the subclass-superclass relationship is genuinely an "is a" relationship.
  - If the abstract class can provide an implementation at the appropriate level of abstraction
- When should one use an interface in place of an Abstract Class?
  - When the methods defined represent a small portion of a class
  - When the subclass needs to inherit from another class
  - When you cannot reasonably implement any of the methods