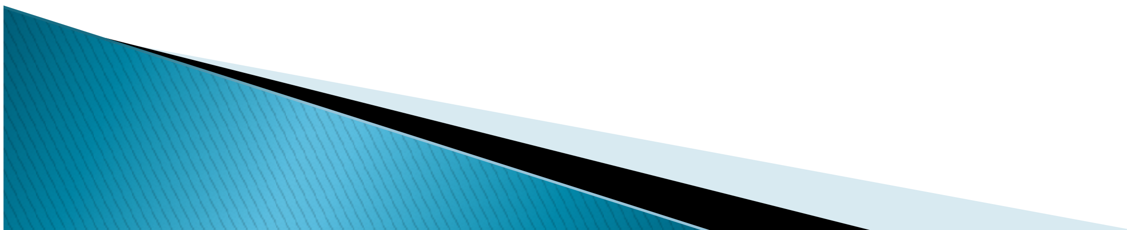
# Interfaces in Java

By
Sunil Kumar(Master of Sc.)
Bangalore, India

# 1. Agenda

- Interface definition
- Implementing an Interface
- Implementing Multiple Interfaces
- Declaring an Interface
- Summary



# Interface Definition

- An interface defines a contract
  - Provides no implementation
- Classes implement interfaces
  - Expresses that the class conforms to the contract
- Interface don’t limit other aspect of the class implementation.
- Lets take an example.

# Interface Definition

- Taking a built in interface called Comparable
  - Java.lang.Comparable
  - Comparable interface is used for determining relative order, how does one instance of a class compare to another instance?
  - It has one method called CompareTo and basically, inside that CompareTo, it’s passed in another instance. And return value of compareTo indicates how the current instance compares to, in sequence to the other instance that was passed in.

# Interface Definition

- If compareTo returns a negative value, it says the current instance should come before the one that was passed into it.
- If it returns positive value, it says the current instance should come after the one that was passed to it.
- If they are equal, it returns zero.

# Interface Example

- Passenger class has been used extensively. Now we want to add the concept of a frequent flyer program, so that we can give people premiums.
- So we will have a concept of a member level. Highest level is three and lowest level as one.
  - 1-Silver
  - 2-Gold
  - 3-Platinum

# Interface Example

- We want to keep track of how long someone has been member. So here we would like to do is compare to Passenger instances based on this idea of a frequent flyer program and have the people who have the highest priority come first and people who are lower come later.
- For this to do we will use implements keyword and then use the java inbuilt interface comparable.

# Interface Example

```
public class Passenger implements Comparable{
	//other members elided for clarity
	private int memberLevel;
	private int memberDays;
	public int compareTo(Object o){
		Passenger p = (Passenger) o;
		if(memberLevel > p.memberLevel)
			return -1;
		else if(memberLevel < p.memberLevel)
			return 1;
		else {
		  if(memberDays > p.memberDays)
		     return -1;
		   else if(memberDays < p.memberDays)
		     return 1;
		   else
		     return 0;
		}
	}
}
```

# Interface Example

- So now the value of implementing this interface is that we can take advantage of features that know how to work with the comparable contract.
- Lets create an object and then run the program to check the output.
- By implementing the comparable interface, and conforming to that contract, we are able to take advantage of the efficient sort capabilities provided by Java.

# Interface Example

```
Passenger bob = new Passenger();
bob.setLevelAndDays(1, 180);
Passenger jane = new Passenger();
bob.setLevelAndDays(1, 90);
Passenger dane = new Passenger();
bob.setLevelAndDays(2, 180);
Passenger nyasha = new Passenger();
bob.setLevel1AndDays(3, 730);

Passenger[] passengers = {bob, jane, dane, nyasha};
Arrays.sort(Passengers);
```

# Generic Interface

- Some interfaces require additional type information.
  - Uses a concept known as generics.

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

  - It means that Comparable interface implementation can be tied to a particular type.
  - It give us all the type safety of having typed parameters.

# Generic Interface Example

```
public class Passenger implements Comparable <Passenger> {
	//other members elided for clarity
	private int memberLevel1;
	private int memberDays;
	public int compareTo(Passenger P){			//Instead of Object type we can have Passenger
		//Passenger p = (Passenger) o;		//This line is not required as casting is not rquired
		if(memberLevel > p.memberLevel)
			return -1;
		else if(memberLevel < p.memberLevel)
			return 1;
		else {
		  if(memberDays > p.memberDays)
		     return -1;
		   else if(memberDays < p.memberDays)
		      return 1;
		   else
		      return 0;
		}
	}
}
```

# Implementing Multiple Interface

- Classes are free to implement multiple interfaces.
- A class can only implement one other class but a class can implement as many interfaces as it needs to. Thus a class can conform to as many interfaces-based contracts as it needs to.

# Multiple Interface Ex.

```
public class Flight implements Comparable<Flight>, Iterable
        <Person> {
//other member elided for clarity
private int flightTime;
private Passenger[] roster;
private CrewMember[] crew;
public int compareTo(Flight f){
        //Flight f = (Flight) o;
        return flightTime – f.flightTime;
}
  public iterator<Person>  iterator(){
            return new FlightIterator(crew, roster);
  }
}
```

```
public class Person {
//other member elided for clarity
        private string name;
}
```

```
public interface Iterable<T>{
        Iterator<T> iterator();
}
```

```
public class CrewMember extends Person {
        //other member elided for clarity
}
```

```
public class Passenger extends Person {
        //other member elided for clarity
}
```

# Multiple Interface Ex.

```
public interface Iterable<T>{
        Iterator <T> iterator();
}
```

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

```
Public class FlightIterator implements Iterator<Person>{
  private CrewMember[] crew;
  private Passenger[] roster;
  private int index = 0;
  public FlightIterator(CrewMember[] crew, Passenger[] roster){
                this.crew = crew;
                this.roster = roster;
}
boolean hasNext(){
        return index < (crew.length + roster.length);
}
public person next(){
    Person p = (index < crew.length)  ?  crew[index] : roster[index – crew.length];
    index++;
return p;
  }
}
```

# Multiple Interface Ex.

```
Flight lax045 = new Flight(45);
//Add crew members;
//Pilot Patty, CoPilot Karl, Air Hostess Mary

//Add Passenger
//Bob, Jane, Dane, Nyasha, Sunil
for(Person p : lax045)
        System.out.println(p.getName(());
```

We will get the output as below.

Pilot Patty
CoPilot Karl
Air Hostess Mary
Bob
Jane
Dane
Nyasha
Sunil

Behind the scene, Java compiler will try to execute below line. This is just for understanding

```
Iterable<Person> laxIterable = lax045;
Iterator<Person> persons = laxIterable.iterator();
while(person.hasNext()) {
        Person p = persons.next();
        System.out.println(p.getName());
}
```

# Multiple Interface Ex.

- When we create that for statement, what effectively happens is that, it takes the flight, lax045 casts that into its iterable interface, then using that, we get the iterator, so here we are putting that in something we are calling persons, and then basically we loop over that. So we have loop in place to just make sure that hasNext is returning true, saying there is more to come. Then we use the next method to get whatever is next, the next person i.e. 1st the Crew Member, then the passengers, and then finally whatever is the body of our loop execute i.e. in this case printing out the name of each.
- So the power of interfaces is that the for each statement is useful for many, many , many different types and those types have very little or nothing in common. But because as long as at type conforms to that iterable interface's contract, it can take advantage of the power of the for statement.

# Declaring an Interface

- Declaring an interface is similar to declaring a class.
- Use the interface keyword
- Supports a subset of the features available to classes.
- Methods
- Name, parameters, and return type
- Implicitly public

# Declaring an Interface

- Interfaces can also have constant –
- Typed Named values
- This constant should be implicitly treated as public, final and static. (So need to use these keyword but by default it is treated like this by Java)
- This means that any constant when we associate with an interface has the same value for implementers of the interface. There is no concept of a per-instance constant.
- Interface can extended another interface i.e. we can add additional method and constants to it. So in a way we extend class, similar way one interface can extend another.
- Implementing extended interface implies implementation of base interface. So any class that implements an interface that extends another interface is automatically considered to implement base interface.
- Please note, interfaces do not have an implementation and they are just defining a contract. The methods of an interface are implicitly public.

# Summary

- An interface defines a contract
  - Provide no implementation
  - Can include methods and constants
- Classes implement interfaces
  - Classes are able to implement multiple interfaces
- Interfaces are able to extend other interfaces
  - Implementing an extended interface implicitly implement the base interface.