

# Interfaces

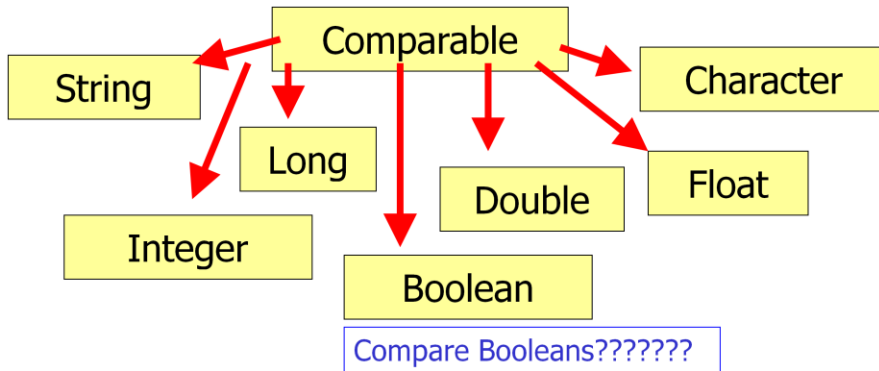
© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Comparable

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# The Comparable Interface

**Most all of the classes in Java that can be compared implement Comparable.**



© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Interfaces

```
Comparable x = 54;  
Comparable y = 67;
```

```
out.println(x.compareTo(y));
```

Why is this okay??

**OUTPUT**

**-1**

© A+ Computer Science - www.apluscompsci.com

x and y are Comparable references that can refer to any class that implements the Comparable interface.

54 and 67 are integers. Java instantiates Integer objects using 54 and 67.

x = 54 is essentially the same as x = new Integer(54);

# Interfaces

```
Comparable x = 9.21;  
Comparable y = 8.54;
```

```
out.println(x.compareTo(y));
```

**OUTPUT**

**1**

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Interfaces

```
Comparable x = "23";
```

```
Comparable y = "45";
```

```
out.println(x.compareTo(y));
```

**OUTPUT**

**-2**

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

When comparing String references via `compareTo()`, Java compares the ASCII values of the first characters that differ.

In this case, 2 and 4 differ as the ASCII value of 2 is 50 and the ASCII value of 4 is 52.

0 has an ASCII value of 48.

The output is -2 as  $50 - 52$  is 0.

# Interfaces

**Comparable x = "dog";**

**Comparable y = "hog";**

**out.println(x.compareTo(y));**

**OUTPUT**

**-4**

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Interfaces

**Comparable x = "dog";**

**Comparable y = "dig";**

**out.println(x.compareTo(y));**

**OUTPUT**

**6**

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)



# Interfaces

```
Comparable x =  
    new Comparable();
```

```
out.println(x);
```

Is this okay??

**OUTPUT**

**no output  
compile error**

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

Comparable is an interface. Interfaces can **not** have instance variables, constructors, or implemented methods.

Interfaces cannot be instantiated because they have no instance variables and they have no constructors.

# Why use an interface?

```
public interface Comparable
{
    int compareTo(Object o);
}
```

**ABSTRACT**  
Lots of  
unknowns!

**No instance variables!**  
**No constructors!**  
**No method implementations!**

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

Comparable is an interface used by most classes that would need to be compared. Comparable has only one abstract method, `compareTo()`.

# Open comparableone.java

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Open comparabletwo.java

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# **Making the abstract concrete**

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# What is an interface?

```
public interface MyHope
{
    boolean makeAFiveInCompSciAP();
}
```

**abstract** - an idea of what is wanted

**concrete** - having enough information  
to actually make it happen

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

All methods listed in an interface are abstract. Abstract methods have no code. Abstract methods are simply the method signature with a semi-colon at the end.

Classes implementing an interface must implement all of the abstract methods listed in the interface.

# What is an interface?

```
public interface MyHope
{
    boolean makeAFiveInCompSciAP();
}
```

**Do know what I want you to do?  
Do I know if you are going to do it?  
Do I know how you are going to do it?**

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

All methods listed in an interface are abstract. Abstract methods have no code. Abstract methods are simply the method signature with a semi-colon at the end.

Classes implementing an interface must implement all of the abstract methods listed in the interface.

# What is an interface?

```
public class Student implements MyHope
{
    //instance variables and constructors not shown

    boolean makeAFiveInCompSciAP(){
        //implementation now shown
    }
}
```

**Now the abstract becomes concrete.  
More is now known.**

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

All methods listed in an interface are abstract. Abstract methods have no code. Abstract methods are simply the method signature with a semi-colon at the end.

Classes implementing an interface must implement all of the abstract methods listed in the interface.



# What is an interface?

```
public interface Comparable
{
    int compareTo(Object o);
}
```

**abstract** - an idea of what to do

**concrete** - having enough  
information to actually do it

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

All methods listed in an interface are abstract. Abstract methods have no code. Abstract methods are simply the method signature with a semi-colon at the end.

Classes implementing an interface must implement all of the abstract methods listed in the interface.

## Writing compareTo()

```
public class Creature implements Comparable
{
    private int size;

    public Creature(int girth) { size=girth; }

    public int compareTo(Object obj)
    {
        Creature other = (Creature)obj;
        if(size>other.size)
            return 1;
        else if(size<other.size)
            return -1;
        return 0;
    }

    public String toString() { return "" + size; }
}
```

The abstract  
becomes  
concrete.

© A+ Computer Science - www.apluscompsci.com

Class Creature implements Comparable. Class Creature must have a compareTo() method.

The compareTo() method must compare the properties of this Creature to the other Creature.

Creature only contains a size property.

## Writing compareTo()

```
public class Word implements Comparable<Word>
{
    private String orig;

    public Word(String s) { orig = s; };

    public int compareTo(Word other)
    {
        // must add code to complete

        return 1;
    }
    public String toString() { return orig; }
}
```

Because Word implements Comparable, Java knows that Word will have a compareTo() method.

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Open comparablethree.java

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Open sort.java

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# **How Java uses Comparable**

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Why use an interface?

**If an entire hierarchy of classes implements the same interface, you can write very generic code to manipulate any of those classes.**

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Interfaces

```
Comparable[] list =  
    new Comparable[25];
```

```
//load with Comparables
```

```
Arrays.sort(list);
```

© A+ Computer Science - www.apluscompsci.com

List is an array of Comparable references. Each spot in list stores the address of a Comparable object.

`Arrays.sort()` will use the `compareTo()` method to compare all Comparable references. This is a great example of polymorphic behavior. The `compareTo()` method calls are made dynamically at run-time by `Arrays.sort()`.



# Interfaces

**Arrays.sort() will use the compareTo() method of each object when sorting the array.**

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

`Arrays.sort()` will use the `compareTo()` method to compare all `Comparable` references. This is a great example of polymorphic behavior. The `compareTo()` method calls are made dynamically at run-time by `Arrays.sort()`.

# Interfaces

```
Comparable[] list = {3,8,7,6,5,4,9};  
Arrays.sort(list);  
for(Comparable num : list)  
{  
    out.println(num);  
}
```

	0	1	2	3	4	5	6
list	3	4	5	6	7	8	9

## OUTPUT

3  
4  
5  
6  
7  
8  
9

© A+ Computer Science - www.apluscompsci.com

Thanks to autoboxing and autounboxing, list can be initialized with a list of primitive integers.

Java instantiates an `Integer` object and passes in each primitive to the new `Integer( )` constructor call.

# Open sorttwo.java

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Interfaces

```
public void sort(Comparable[] stuff)
{
    for(int i=0;i<stuff.length-1;i++)
    {
        int spot=i;
        for(int j=i;j<stuff.length;j++){
            if(stuff[j].compareTo(stuff[spot])>0)
                spot=j;
        }
        Comparable save=stuff[i];
        stuff[i]=stuff[spot];
        stuff[spot]=save;
    }
}
```

Why would this method be passed an array of Comparable?

Does this demonstrate the power of interfaces and hierarchies?

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

The selection sort sorts Comparable arrays. Because stuff is an array of Comparable, I can write very generic code based on the compareTo() method.

As long as an array of Comparable references is sent in, this code will work perfectly every time.

# Open sortthree.java

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# **More Interfaces**

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Interfaces

```
public interface Exampleable
{
    int writeIt(Object o);
    int x = 123;
}
```

**Methods are public abstract!**  
**Variables are public static final!**

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

All methods listed in an interface are public abstract. Abstract methods have no code.

Each abstract method listed in an interface must be implemented in the class that implements the interface.

All variables listed in an interface are public static final, making them final class variables.

Interfaces cannot contain implemented methods, constructors, or instance variables.

# Interfaces

```
public interface Exampleable  
{  
    public abstract int writeIt(Object o);  
    public static final int x = 123;  
}
```

**Methods are public abstract!**  
**Variables are public static final!**

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

All methods listed in an interface are public abstract. Abstract methods have no code.

Each abstract method listed in an interface must be implemented in the class that implements the interface.

All variables listed in an interface are public static final, making them final class variables.

Interfaces cannot contain implemented methods, constructors, or instance variables.



# Interfaces

**An interface is a list of methods that must be implemented.**

**An interface may not contain any implemented methods.**

**Interfaces cannot have constructors!!!**

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

All methods listed in an interface are public abstract. Abstract methods have no code.

Each abstract method listed in an interface must be implemented in the class that implements the interface.

All variables listed in an interface are public static final, making them final class variables.

Interfaces cannot contain implemented methods, constructors, or instance variables.

# Interfaces

**Interfaces are typically used when you know what you want an Object to do, but do not know what will be used to get it done.**

**If only the behavior is known, use an interface.**

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

Interfaces are used to detail what things an Object should do. Interfaces are used typically when the way an Object will do things is unknown.

Comparable is a great example. With Comparable, it is clear that each Object should be compared to another Object of the same type. Comparable is an interface because it is not known what the Objects that implement Comparable will contain. It is known that the Object should be compared to other Objects of the same type in a certain way.

# Interfaces

```
public interface Locatable  
{  
    public int getX();  
    public int getY();  
}
```

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Interfaces

```
public interface Movable  
{  
    public void setPos( int x, int y);  
    public void setX( int x );  
    public void setY( int y );  
}
```

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Interfaces

```
class Ship implements Locatable, Movable
{
    private int xPos, yPos;

    //how many methods must
    //be implemented?
}
```

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Interfaces

**class A implements B { }**

**A class can implement multiple interfaces.**

**class A implements B,C { } //legal**

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Interfaces

**Interfaces are true abstract classes. All methods listed in an interface are abstract; as a result, you must implement every method.**

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

All methods listed in an interface are public abstract. Abstract methods have no code.

Each abstract method listed in an interface must be implemented in the class that implements the interface.

All variables listed in an interface are public static final, making them final class variables.

Interfaces cannot contain implemented methods, constructors, or instance variables.

# Open interface.java

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)