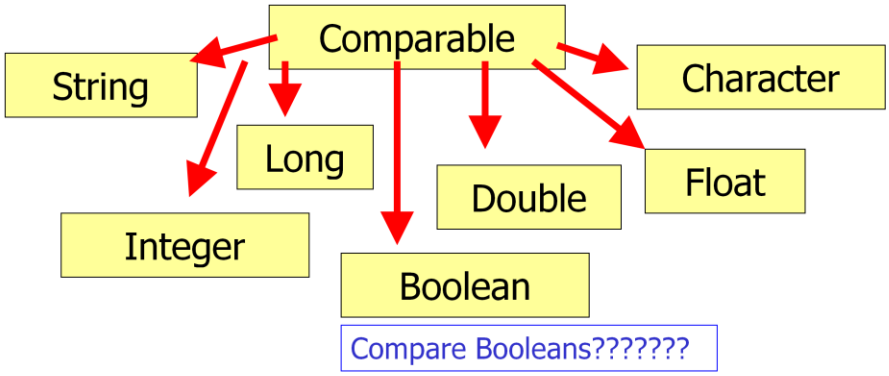
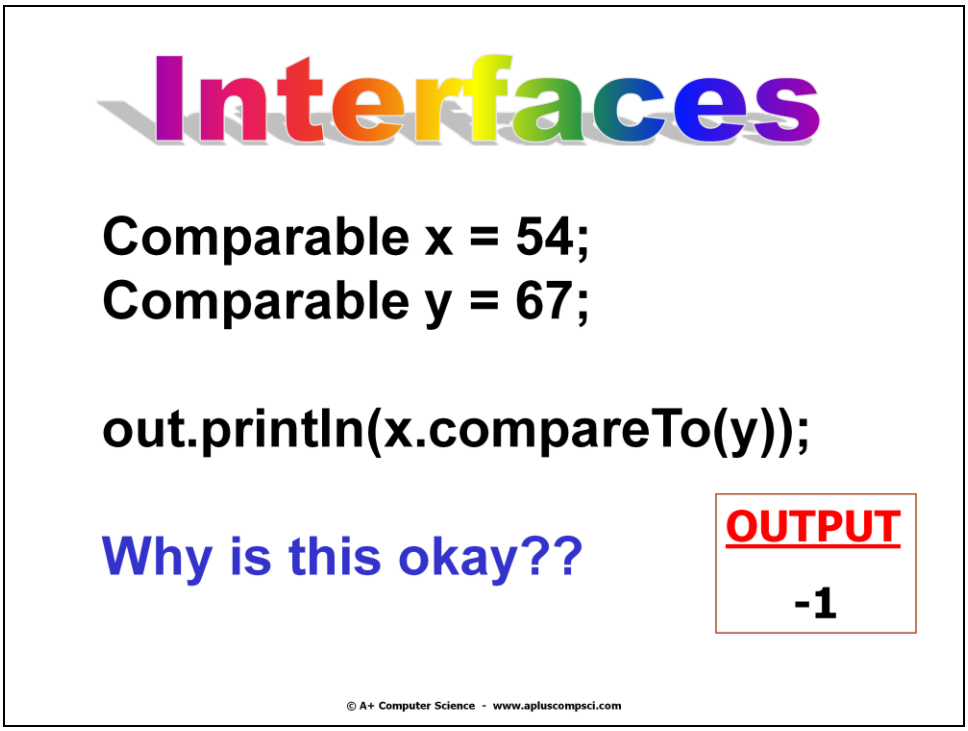
Interfaces

Comparable

The Comparable Interface

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





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.

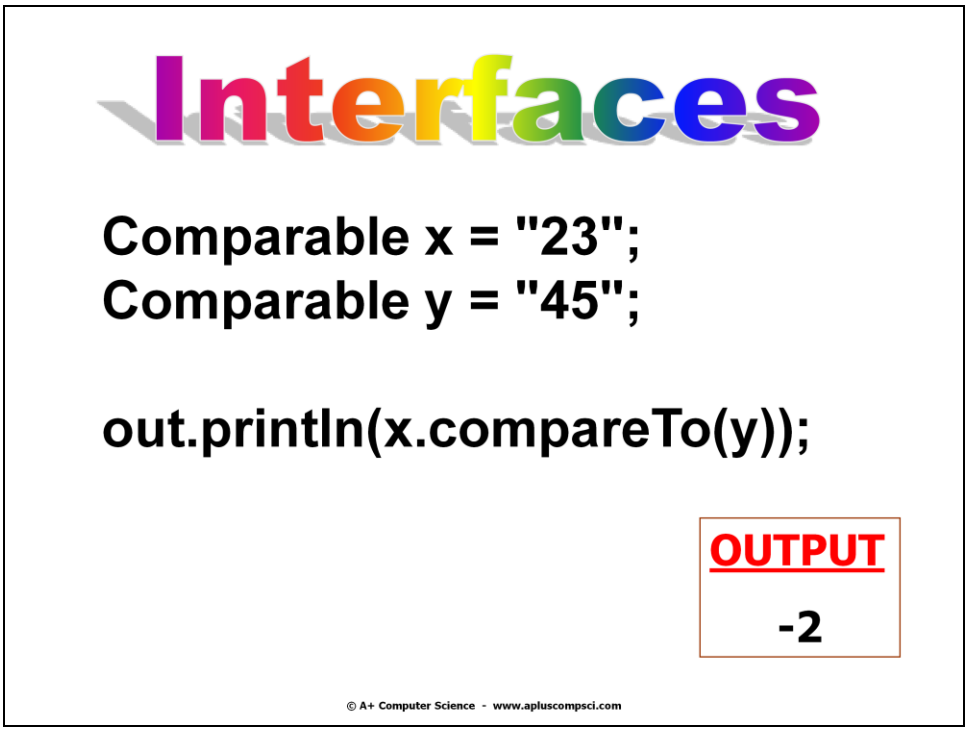
`x = 54` is essentially the same as `x = new Integer(54);`

Interfaces

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

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

OUTPUT

1



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

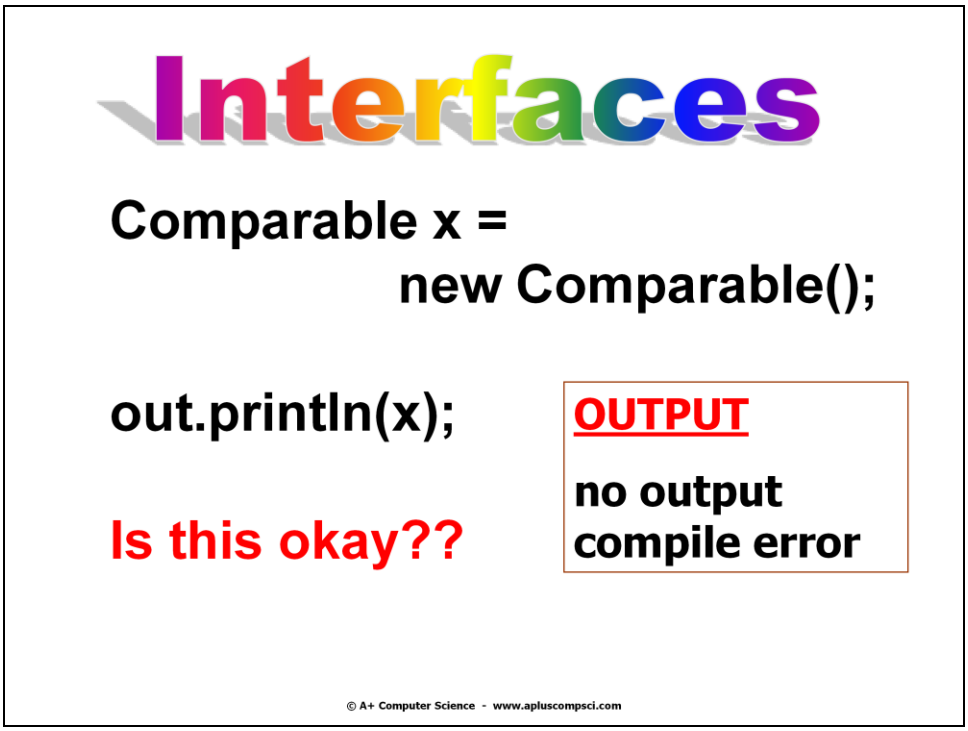
-4

Interfaces

```
Comparable x = “dog";
Comparable y = “dig";

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

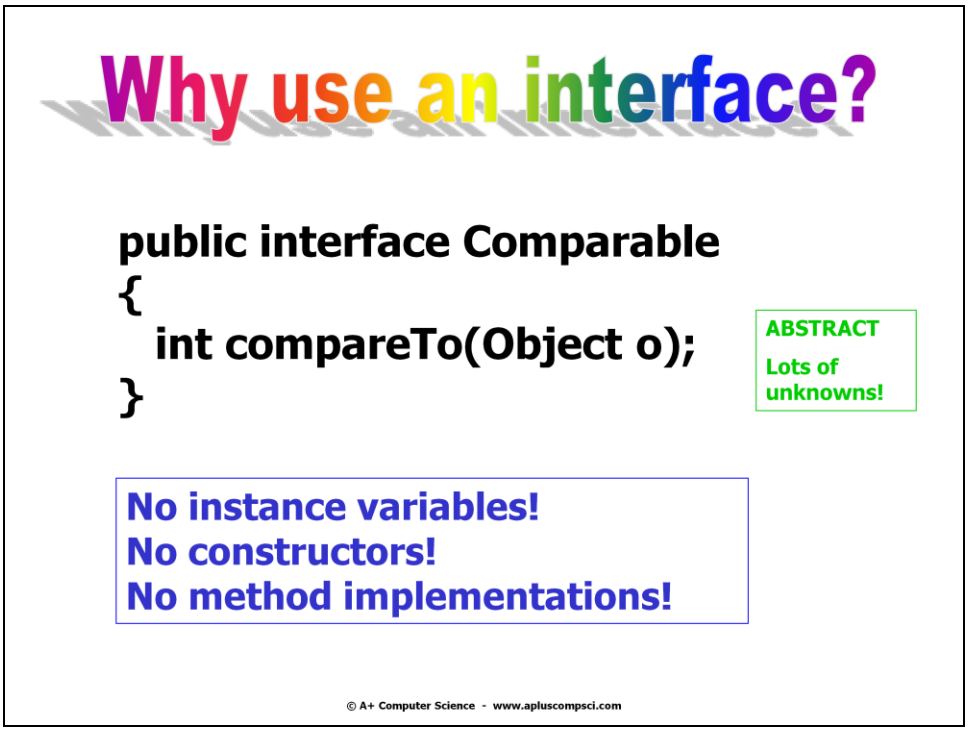
OUTPUT

6



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.

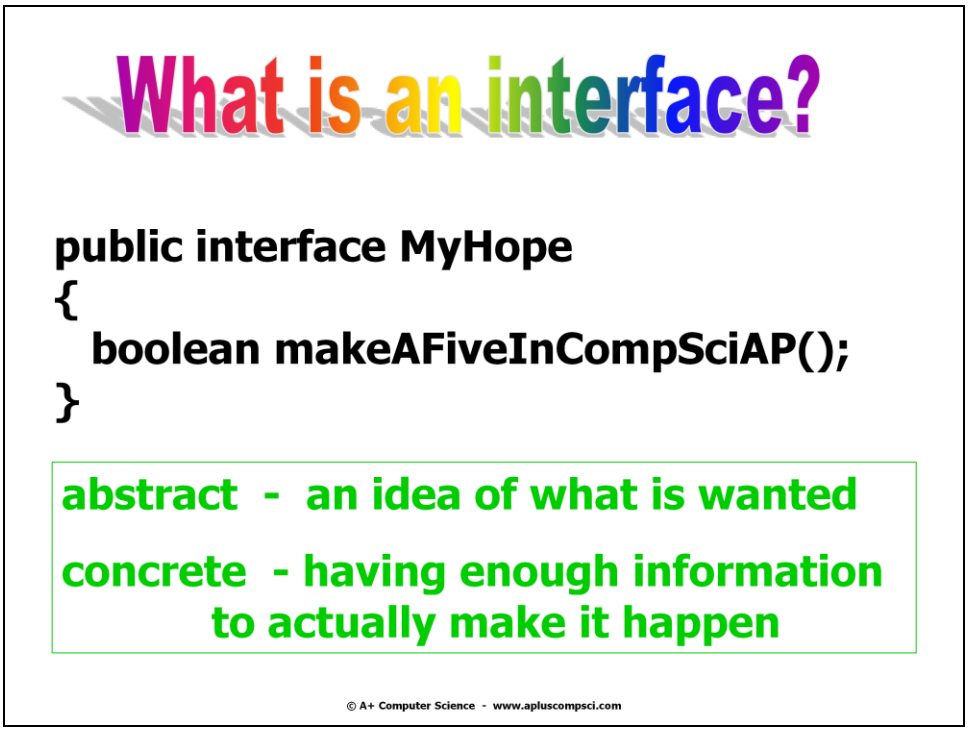


Comparable is an interface used by most classes that would need to be compared. Comparable has only one abstract method, `compareTo()`.

Open comparableone.java

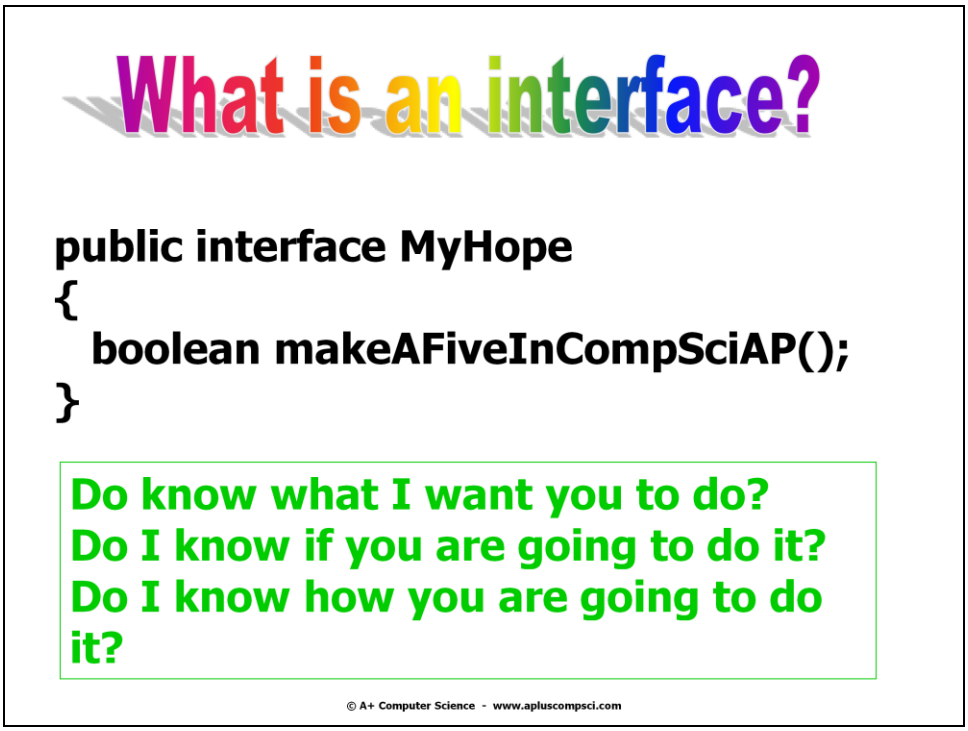
Open comparabletwo.java

Making the abstract concrete



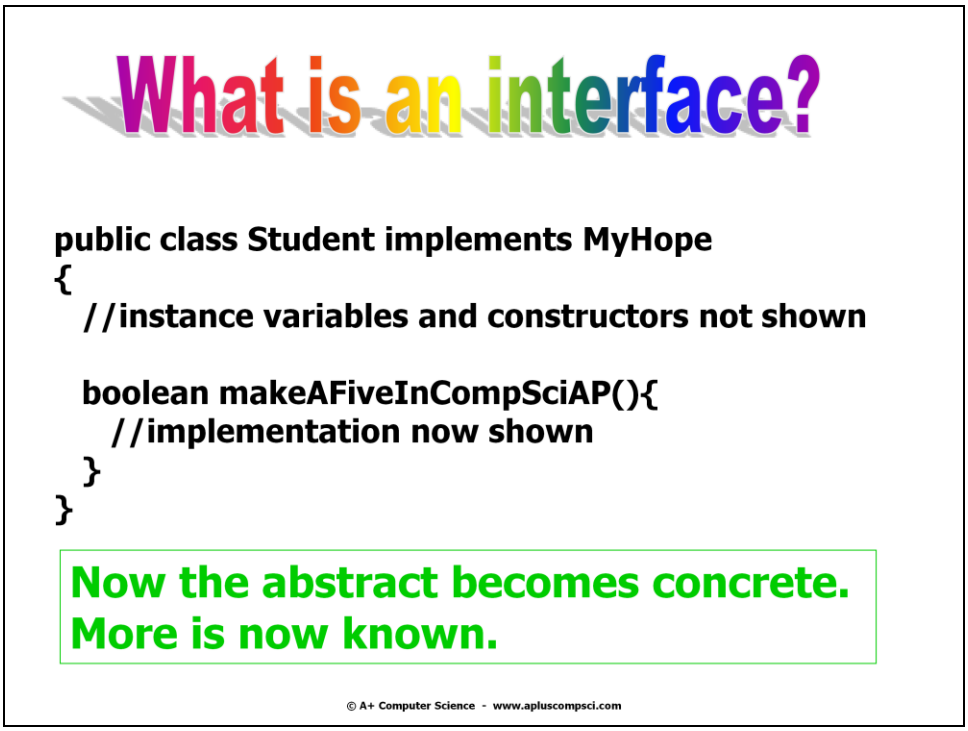
All methods listed in an interface are abstract. Abstract methods have no code. Abstract methods are simply the method signature with a semi-colon at the end.

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



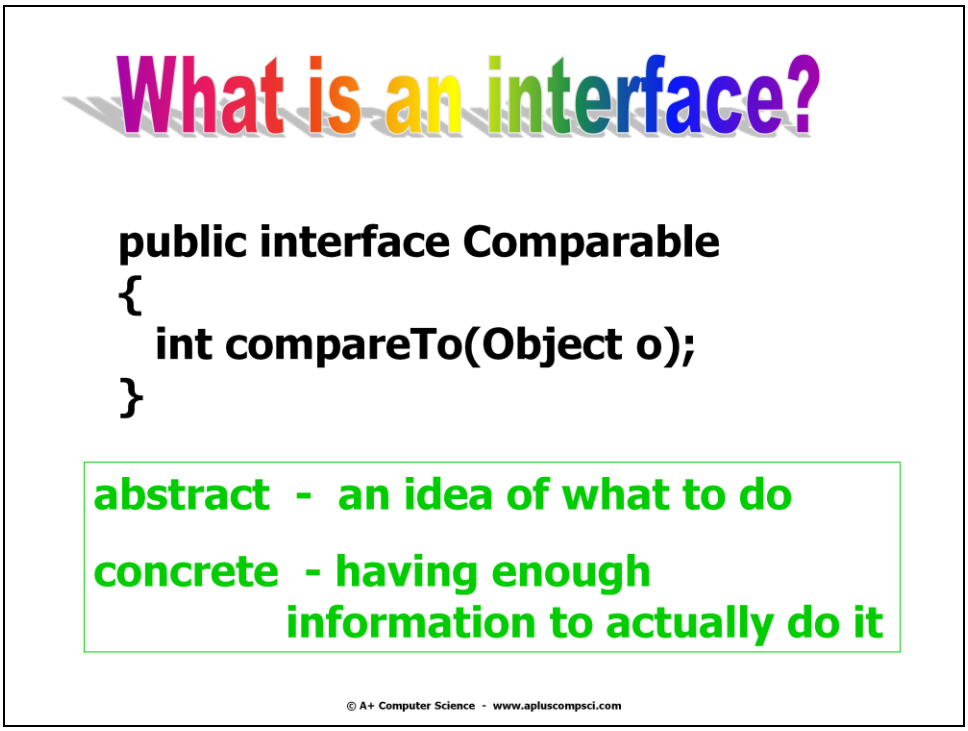
All methods listed in an interface are abstract. Abstract methods have no code. Abstract methods are simply the method signature with a semi-colon at the end.

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



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.



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.

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.

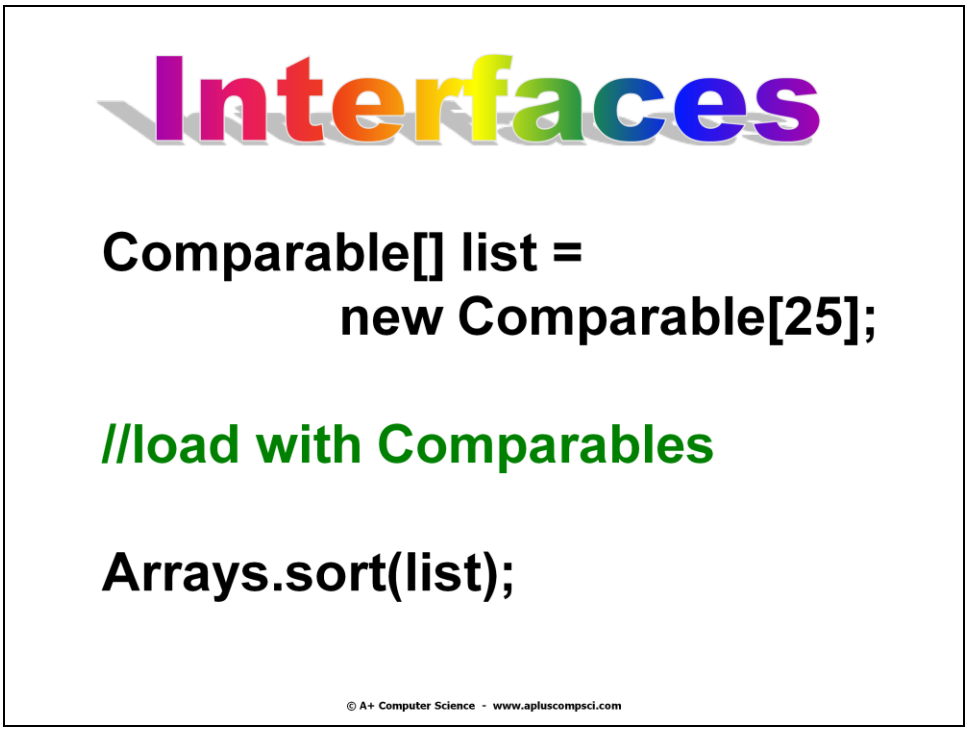
Open comparablethree.java

Open sort.java

How Java uses Comparable

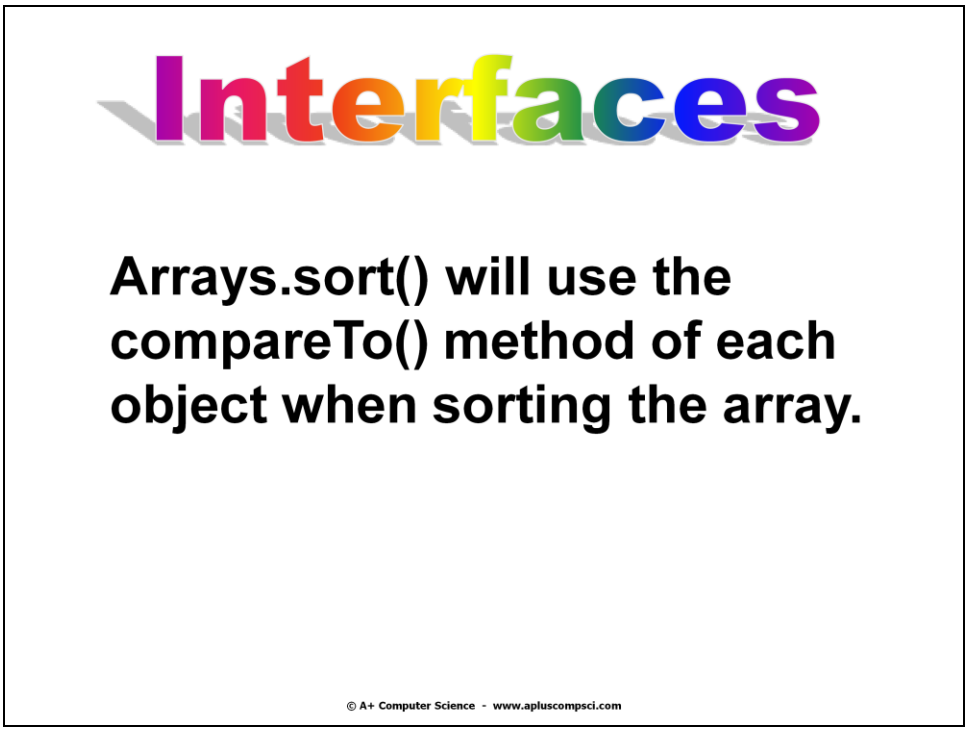
Why use an interface?

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

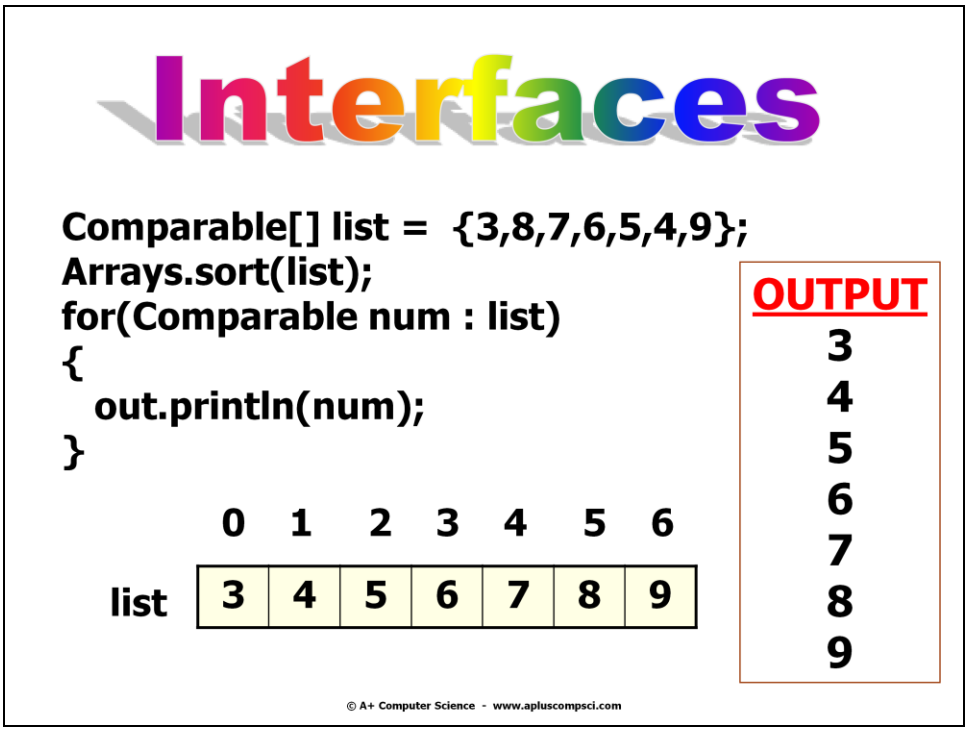


List is an array of `Comparable` references. Each spot in list stores the address of a `Comparable` object.

`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()`.



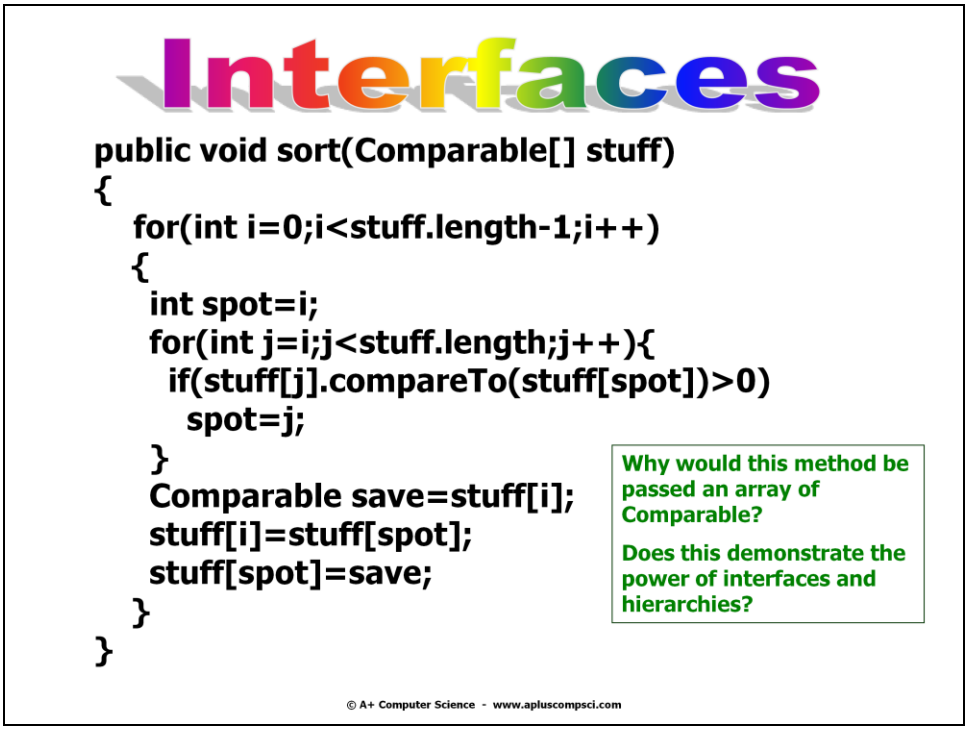
`Arrays.sort()` will use the `compareTo()` method to compare all `Comparable` references. This is a great example of polymorphic behavior. The `compareTo()` method calls are made dynamically at run-time by `Arrays.sort()`.



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

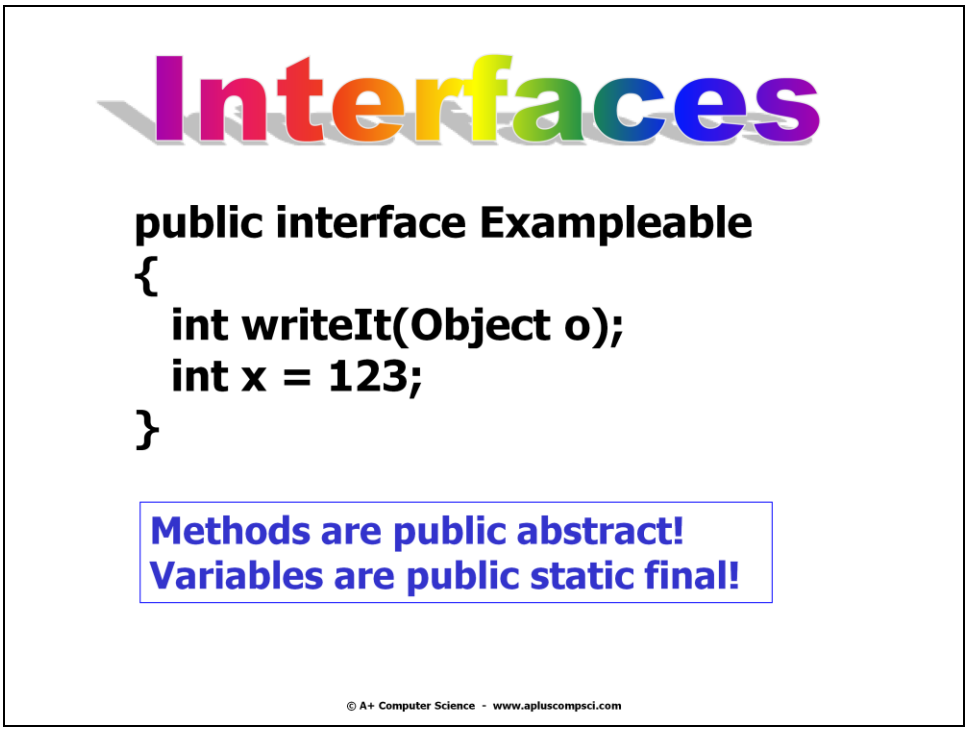


The selection sort sorts `Comparable` arrays. Because stuff is an array of `Comparable`, I can write very generic code based on the `compareTo()` method.

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

Open sortthree.java

More Interfaces

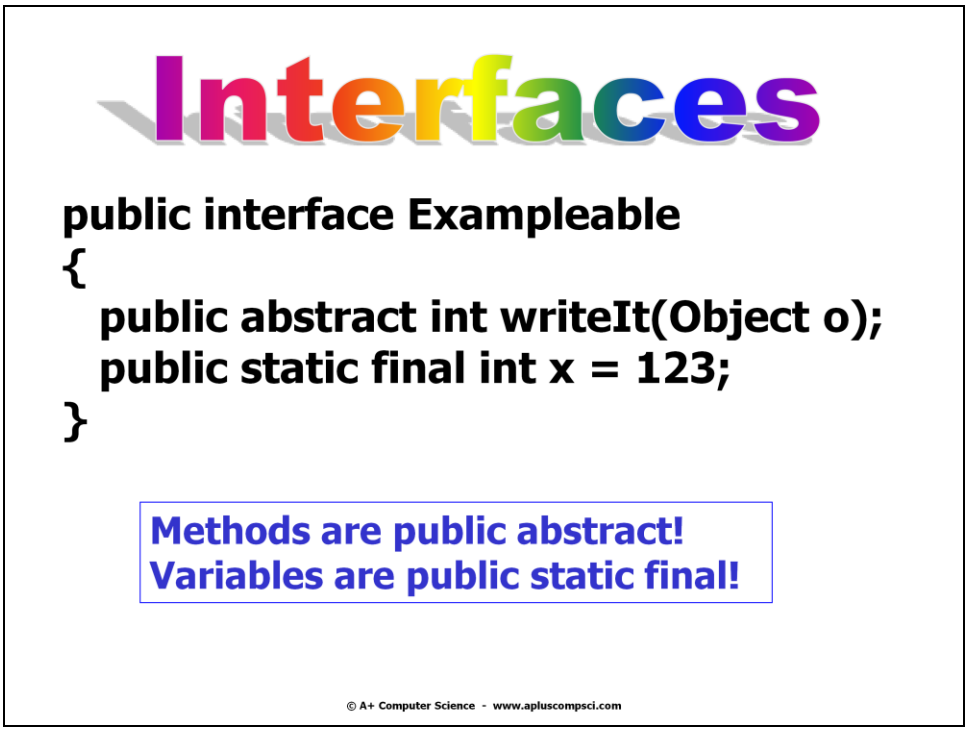


All methods listed in an interface are public abstract. Abstract methods have no code.

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.

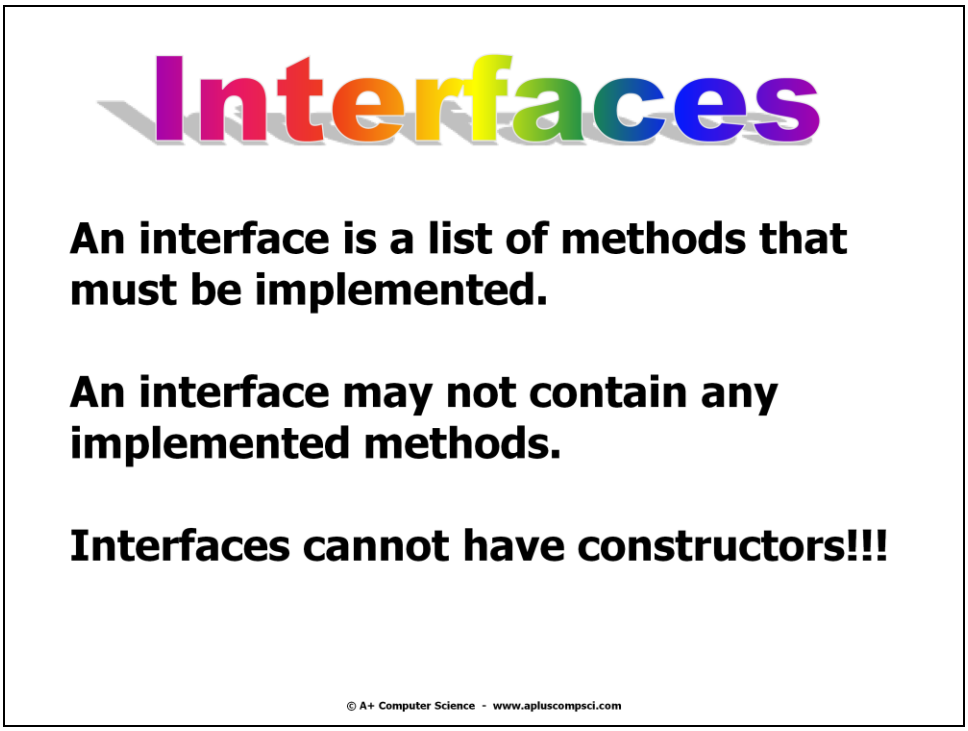


All methods listed in an interface are public abstract. Abstract methods have no code.

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.

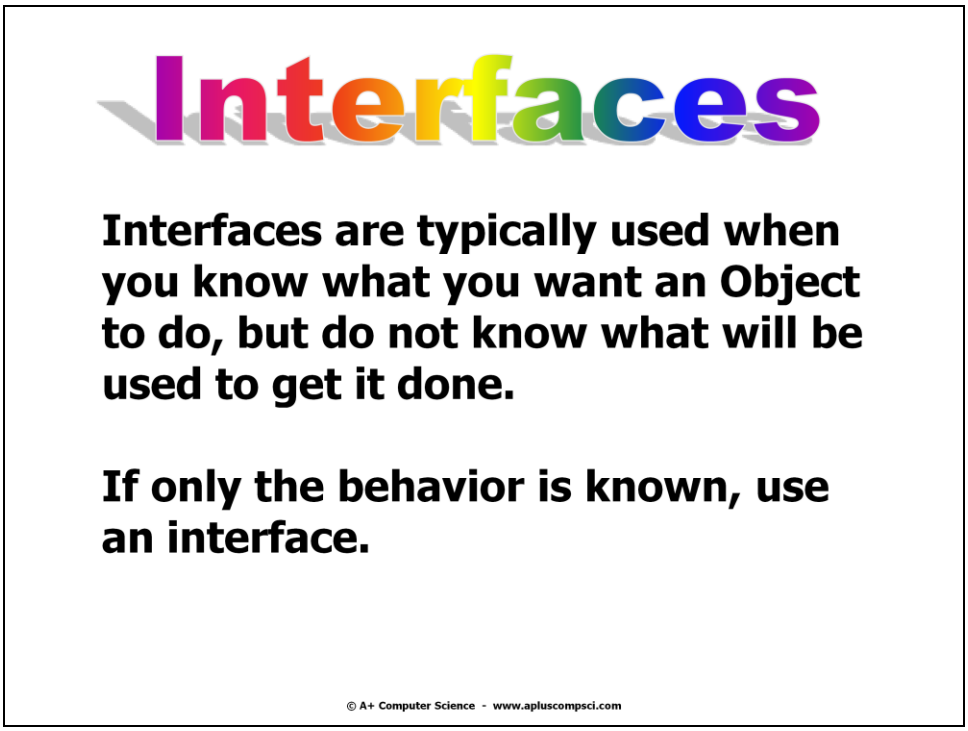


All methods listed in an interface are public abstract. Abstract methods have no code.

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 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 Obejcts of the same type in a certain way.

Interfaces

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

Interfaces

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

Interfaces

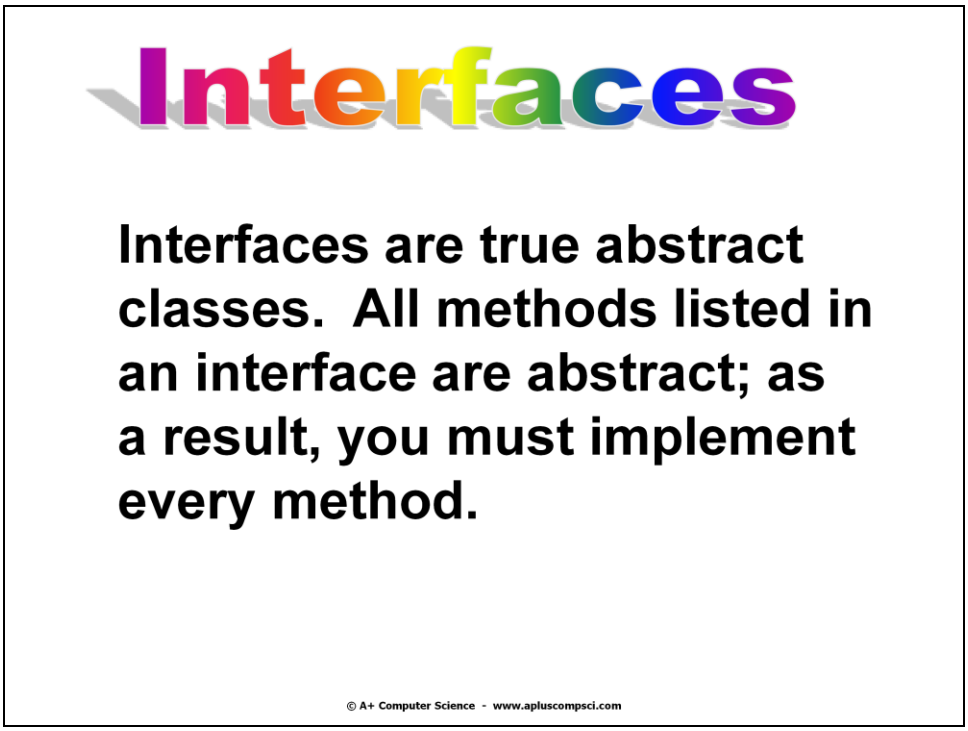
```
class Ship implements Locatable, Movable
{
      private int xPos, yPos;

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

Interfaces

class A implements B { }

A class can implement multiple interfaces.

class A implements B,C { } //legal



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