CS 171: Introduction to Computer Science II

Simple Sorting (cont.) + Interface

Li Xiong

Today

- Simple sorting algorithms (cont.)
  - Bubble sort
  - Selection sort
  - Insertion sort
- Interface

Sorting problem

Ex. Student records in a university.

item →

key →

Chen	3	A	991-878-4944	308 Blair
Rohde	2	A	232-343-5555	343 Forbes
Gazsi	4	B	766-093-9873	101 Brown
Furia	**1**	**A**	**766-093-9873**	**101 Brown**
Kanaga	3	B	898-122-9643	22 Brown
Andrews	3	A	664-480-0023	097 Little
Battle	4	C	874-088-1212	121 Whitman

Sort. Rearrange array of N items into ascending order.

Andrews	3	A	664-480-0023	097 Little
Battle	4	C	874-088-1212	121 Whitman
Chen	3	A	991-878-4944	308 Blair
Furia	1	A	766-093-9873	101 Brown
Gazsi	4	B	766-093-9873	101 Brown
Kanaga	3	B	898-122-9643	22 Brown
Rohde	2	A	232-343-5555	343 Forbes

Two useful sorting abstractions

Helper functions. Refer to data through compares and exchanges.

Less. Is item `v` less than `w` ?

```
private static boolean less(Comparable v, Comparable w)
{  return v.compareTo(w) < 0;  }
```

Exchange. Swap item in array `a[]` at index `i` with the one at index `j`.

```
private static void exch(Comparable[] a, int i, int j)
{
   Comparable swap = a[i];
   a[i] = a[j];
   a[j] = swap;
}
```

Analysis of Bubble Sort

- Number of comparisons?

$$\frac{N(N-1)}{2} = O(N^2)$$

- Number of swaps?

  best case: $O(1)$

  worst cast: $\frac{N(N-1)}{2} = O(N^2)$

  average: $\frac{N(N-1)}{4} = O(N^2)$

Selection Sort

1. Keep track of the **index** of the smallest number in each round.
2. Swap the smallest number towards the beginning of the array.
3. Repeat the above two steps.

Selection Sort Implementation

```
public class Selection
{
   public static void sort(Comparable[] a)
   {
      int N = a.length;
      for (int i = 0; i < N; i++)
      {
         int min = i;
         for (int j = i+1; j < N; j++)
            if (less(a[j], a[min]))
               min = j;
         exch(a, i, min);
      }
   }

   private static boolean less(Comparable v, Comparable w)
   {  /* as before */  }

   private static void exch(Comparable[] a, int i, int j)
   {  /* as before */  }
}
```

Selection Sort

- Number of comparisons?

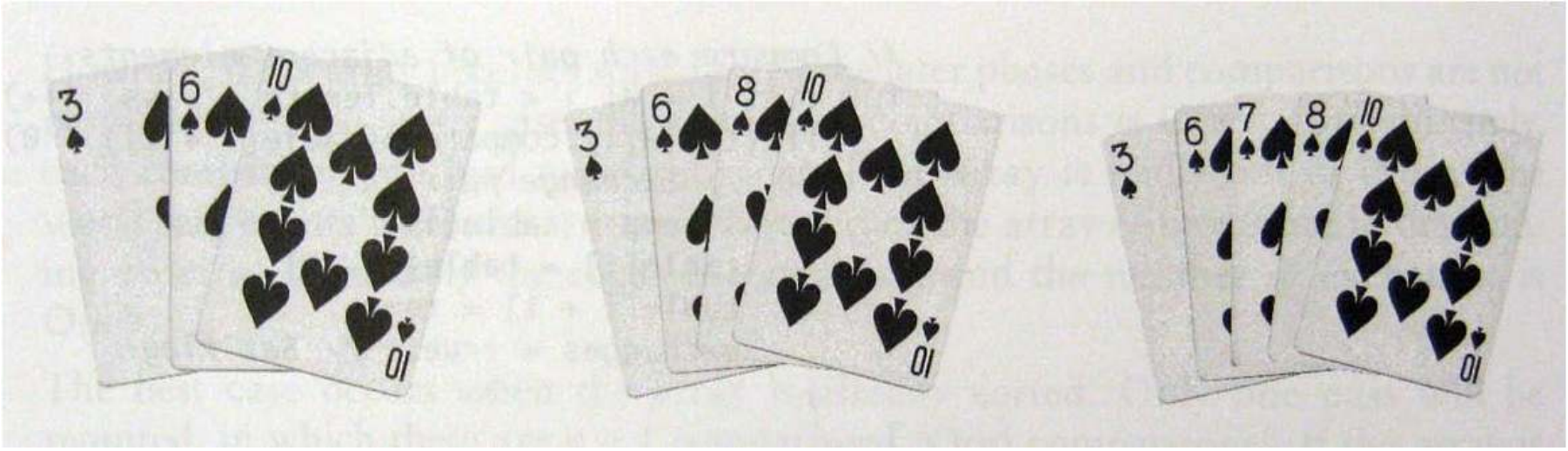
$$O(N^2)$$

- Number of swaps?

$$O(N)$$

Card Sorting Exercise

- How do you sort a hand of poker cards?



Insertion Sort

- Idea
  - Assume the left portion of the array is *partially sorted* (however, unlike selection sort, the elements are not necessarily in their final positions)
  - For each remaining element on the right portion, insert it to the left portion (similar to insertion in an ordered array).
  - Repeat until done.

Algorithm. ↑ scans from left to right.

Invariants.

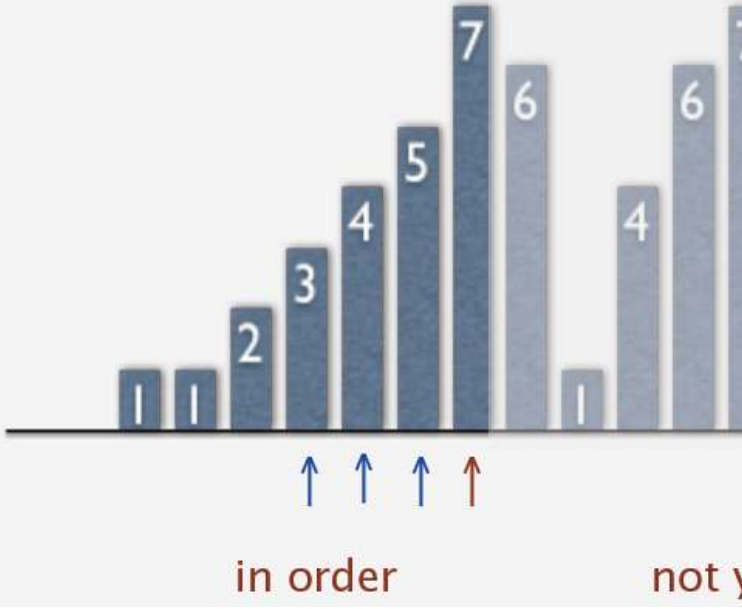
- Entries to the left of ↑ (including ↑) are in ascending order.
- Entries to the right of ↑ have not yet been seen.

- Move the pointer to the right.

```
i++;
```

- Moving from right to left, exchange `a[i]` with each larger entry to its left.

```
for (int j = i; j > 0; j--)
   if (less(a[j], a[j-1]))
         exch(a, j, j-1);
   else break;
```



Insertion Sort Implementation

```
public class Insertion
{
   public static void sort(Comparable[] a)
   {
      int N = a.length;
      for (int i = 0; i < N; i++)
         for (int j = i; j > 0; j--)
            if (less(a[j], a[j-1]))
               exch(a, j, j-1);
            else break;
   }

   private static boolean less(Comparable v, Comparable w)
   {  /* as before */  }

   private static void exch(Comparable[] a, int i, int j)
   {  /* as before */  }
}
```

Insertion Sort

- Online demo
  - http://www.sorting-algorithms.com/insertion-sort
- Romanian dance demo
  - http://www.youtube.com/watch?v=ROalU379l3U

Insertion Sort

- Number of comparisons?

- Number of exchanges?

Insertion sort

- Best case
  - N-1 comparisons
  - 0 exchanges
- Worst case
  - ~$N^2/2$ comparisons
  - ~$N^2/2$ exchanges
- Average case
  - ~$N^2/4$ comparisons
  - ~$N^2/4$ exchanges

Summary

- Both selection and insertion sort are comparison based.
- Both have an **average comparison cost** of $O(N^2)$

- Later we will learn several faster sorting algorithms, with a typical cost of $O(N \log N)$

Hw2

- Implement Bubble Sort
- Compare the runtime for bubble sort, selection sort, and insertion sort

Java's Sorting Methods

- **Primitive Type Arrays**:

```
Arrays.sort(int[]);
Arrays.sort(int[], int fromIdx, int toIdx);
Arrays.sort(float[]);
```

……

What sorting algorithm is used?

Java's Sorting Methods

- **Object Type Arrays**:

```
Arrays.sort(Object[]);
Arrays.sort(Object[], Comparator);
Arrays.sort(Object[], int fromIdx, int
toIdx, Comparator);
```

……

- **Comparator** is used to define how to compare two objects (i.e. which is bigger / smaller).

int **compare**(T o1, T o2)

boolean **equals**(Object obj)

What sorting algorithm is used?

Today

- Simple sorting algorithms (cont.)
  - Bubble sort
  - Selection sort
  - Insertion sort
- Interface

Insertion Sort Implementation

```
public class Insertion
{
   public static void sort(Comparable[] a)
   {
      int N = a.length;
      for (int i = 0; i < N; i++)
         for (int j = i; j > 0; j--)
            if (less(a[j], a[j-1]))
               exch(a, j, j-1);
            else break;
   }

   private static boolean less(Comparable v, Comparable w)
   {  /* as before */  }

   private static void exch(Comparable[] a, int i, int j)
   {  /* as before */  }
}
```

Goal. Sort any type of data.

Ex 1. Sort random real numbers in ascending order.

seems artificial, but stay tuned for an application

```
public class Experiment
{
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      Double[] a = new Double[N];
      for (int i = 0; i < N; i++)
         a[i] = StdRandom.uniform();
      Insertion.sort(a);
      for (int i = 0; i < N; i++)
         StdOut.println(a[i]);
   }
}
```

```
% java Experiment 10
0.08614716385210452
0.09054270895414829
0.10708746304898642
0.21166190071646818
0.363292849257276
0.460954145685913
0.5340026311350087
0.7216129793703496
0.9003500354411443
0.9293994908845686
```

Goal. Sort any type of data.

Ex 2. Sort strings from file in alphabetical order.

```
public class StringSorter
{
   public static void main(String[] args)
   {
      String[] a = In.readStrings(args[0]);
      Insertion.sort(a);
      for (int i = 0; i < a.length; i++)
         StdOut.println(a[i]);
   }
}
```

```
% more words3.txt
bed bug dad yet zoo ... all bad yes

% java StringSorter words3.txt
all bad bed bug dad ... yes yet zoo
```

Callback Mechanism: Interface

client

```
import java.io.File;
public class FileSorter
{
   public static void main(String[] args)
   {
      File directory = new File(args[0]);
      File[] files = directory.listFiles();
      Insertion.sort(files);
      for (int i = 0; i < files.length; i++)
         StdOut.println(files[i].getName());
   }
}
```

object implementation

```
public class File
implements Comparable<File>
{
   ...
   public int compareTo(File b)
   {
      ...
      return -1;
      ...
      return +1;
      ...
      return 0;
   }
}
```

Comparable interface (built in to Java)

```
public interface Comparable<Item>
{
   public int compareTo(Item that);
}
```

sort implementation

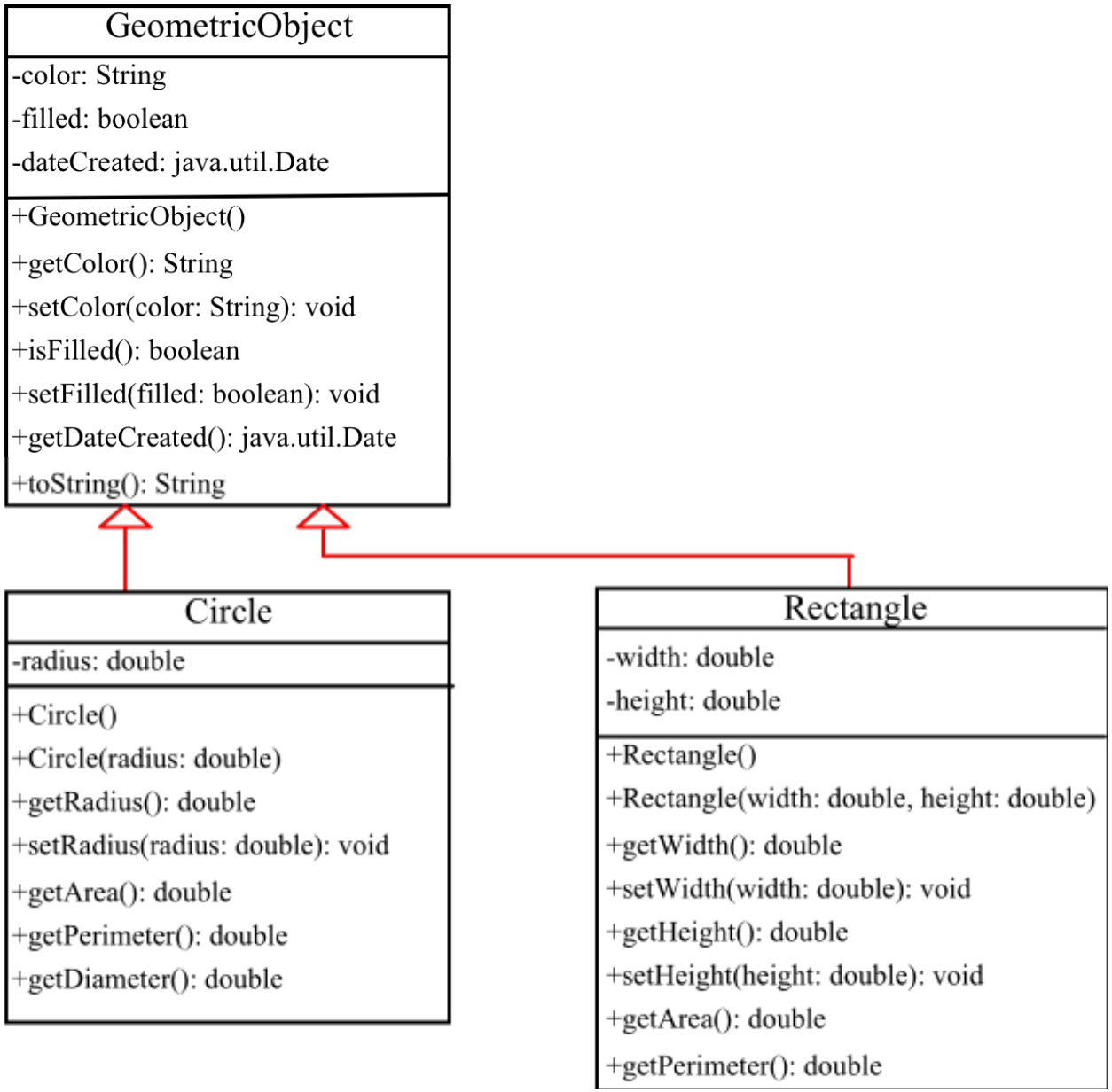
```
public static void sort(Comparable[] a)
{
   int N = a.length;
   for (int i = 0; i < N; i++)
      for (int j = i; j > 0; j--)
         if (a[j].compareTo(a[j-1]) < 0)
            exch(a, j, j-1);
         else break;
}
```

key point: no dependence on `File` data type

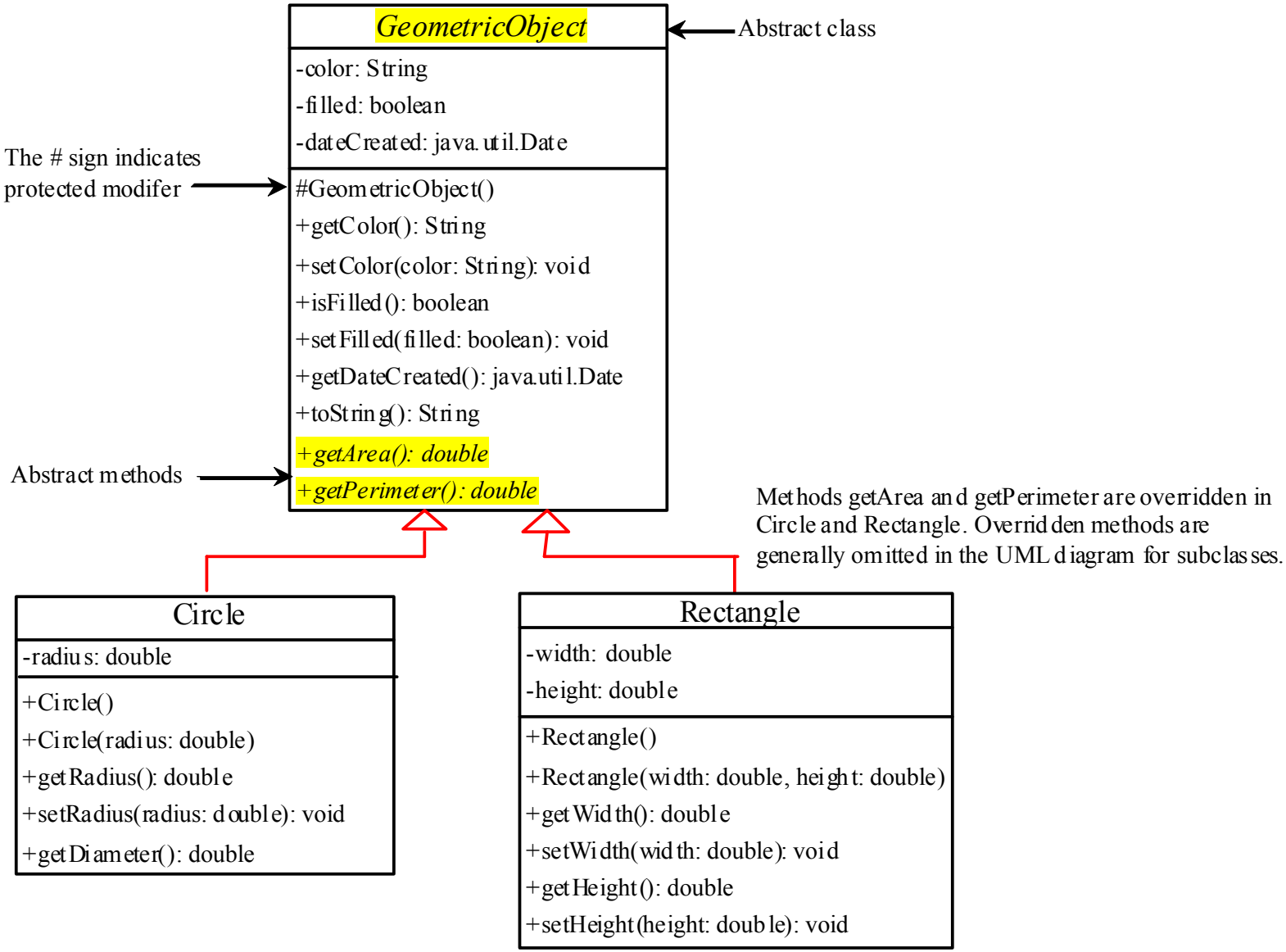
Abstract Classes and Interfaces

- Abstract class and abstract methods
- Interfaces

Superclasses and Subclasses



Abstract Classes and Abstract Methods



abstract method in abstract class

```
public abstract void method();
```

- If a class contains abstract methods, it must be declared abstract
- If a subclass of an abstract superclass does not implement all the abstract methods, the subclass must be declared abstract

Instance cannot be created from abstract class

- An abstract class cannot be instantiated using the new operator
- You can still define its constructors, which are invoked in the constructors of its subclasses
- For instance, the constructors of GeometricObject are invoked in the Circle class and the Rectangle class.

superclass of abstract class may be concrete

- A subclass can be abstract even if its superclass is concrete
- For example, the Object class is concrete, but its subclasses, such as GeometricObject, may be abstract

abstract class as type

- You cannot create an instance from an abstract class using the new operator, but an abstract class can be used as a data type

```
GeometricObject obj = new Circle(10);

GeometricObject[] geo = new GeometricObject[10];
```

Review questions

- Which of the following declares an abstract method in an abstract Java class?

A. public abstract method();

B. public abstract void method();

C. public void abstract Method();

D. public void method() {}

E. public abstract void method() {}

Review questions

Which of the following statements regarding abstract methods are true?

A. An abstract class can have instances created using the constructor of the abstract class.
B. An abstract class can be extended.
C. A subclass of a non-abstract superclass can be abstract.
D. An abstract class can be used as a data type.

Review questions

Suppose A is an abstract class, B is a concrete subclass of A, and both A and B have a default constructor. Which of the following is correct?

A a = new A();

A a = new B();

B b = new A();

B b = new B();

Interfaces

- What is an interface?
- Why is an interface useful?
- How do you define an interface?
- How do you use an interface?

What is an interface? Why is an interface useful?

- An interface is a classlike construct that contains only constants and abstract methods
- In many ways, an interface is similar to an abstract class, but the intent of an interface is to specify behavior for objects
  - Specify objects that are comparable, edible, cloneable using appropriate interfaces such as `Comparable`, `Edible`, and `Cloneable`
- A class that implements an interface need to implement all the abstract methods
  - define Orange and Chicken classes that implement `Edible` interface

Interface is a Special Class

- Like an abstract class, you cannot create an instance from an interface using the new operator
- You can create an instance from a class that implements an interface
- You can use an interface as a data type for a variable, as the result of casting, and so on.

Define an Interface

```
public interface InterfaceName {
  constant declarations;
  method signatures;
}
```

```
public interface Edible {
  /** Describe how to eat */
  public abstract String howToEat();
}
```

Omitting Modifiers in Interfaces

- All data fields are public final static (constants) in an interface
- All methods are public abstract in an interface

```
public interface T1 {
  public static final int K = 1;

  public abstract void p();
}
```

Equivalent

```
public interface T1 {
  int K = 1;

  void p();
}
```

The Comparable Interface

```
// This interface is defined in
// java.lang package
package java.lang;

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

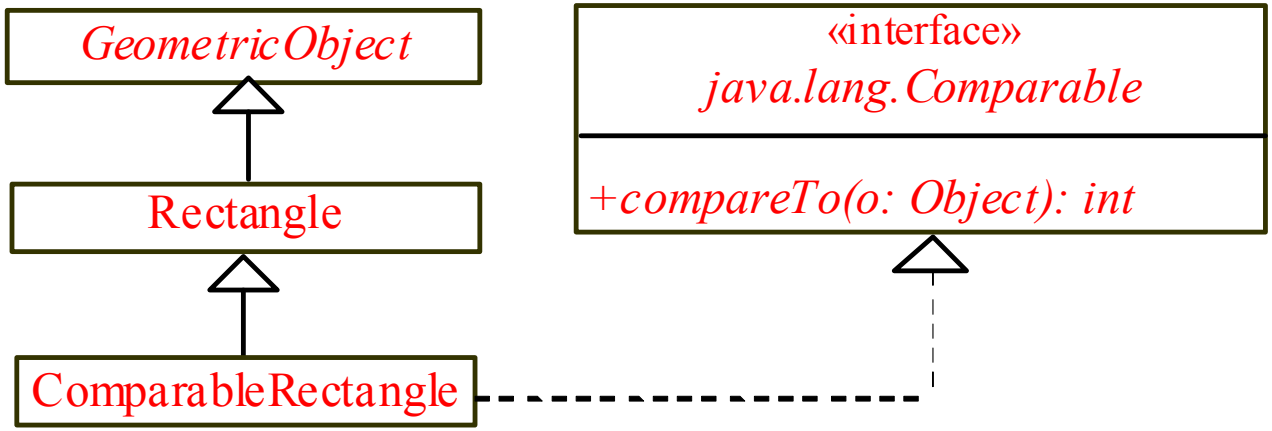
String and Date Classes

- Many classes (e.g., String and Date) in the Java library implement Comparable to define a natural order for the objects

```
public class String extends Object
    implements Comparable {
  // class body omitted
}
```

```
public class Date extends Object
    implements Comparable {
  // class body omitted
}
```

Declaring Classes to Implement Comparable




```
ComparableRectangle rectangle1 = new ComparableRectangle(4, 5);
ComparableRectangle rectangle2 = new ComparableRectangle(3, 6);
System.out.println(Max.max(rectangle1, rectangle2));
```

Callback Mechanism: Interface

client

```
import java.io.File;
public class FileSorter
{
   public static void main(String[] args)
   {
      File directory = new File(args[0]);
      File[] files = directory.listFiles();
      Insertion.sort(files);
      for (int i = 0; i < files.length; i++)
         StdOut.println(files[i].getName());
   }
}
```

object implementation

```
public class File
implements Comparable<File>
{
   ...
   public int compareTo(File b)
   {
      ...
      return -1;
      ...
      return +1;
      ...
      return 0;
   }
}
```

Comparable interface (built in to Java)

```
public interface Comparable<Item>
{
   public int compareTo(Item that);
}
```

sort implementation

```
public static void sort(Comparable[] a)
{
   int N = a.length;
   for (int i = 0; i < N; i++)
      for (int j = i; j > 0; j--)
         if (a[j].compareTo(a[j-1]) < 0)
            exch(a, j, j-1);
         else break;
}
```

key point: no dependence on `File` data type