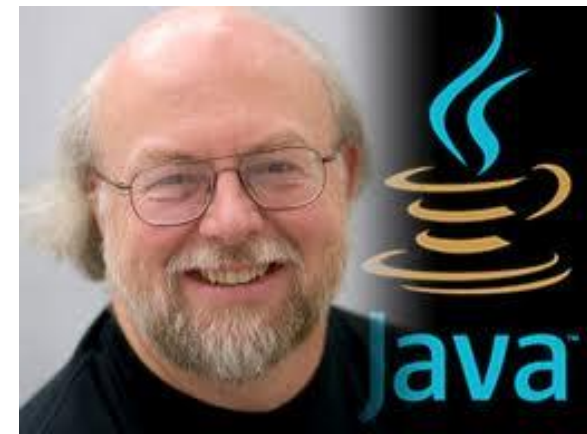


# Topic 7

## Interfaces



I once attended a Java user group meeting where James Gosling (one of Java's creators) was the featured speaker. During the memorable Q&A session, someone asked him: "If you could do Java over again, what would you change?" **"I'd leave out classes,"** he replied. After the laughter died down, he explained that the real problem wasn't classes per se, but rather implementation inheritance (the extends relationship). Interface inheritance (the implements relationship) is preferable.

- Allen Holub



# Clicker 1

► How many sorts do you want to have to write?

```
public static void selSort(double[] data) {  
    for(int i = 0; i < data.length; i++) {  
        int small = i;  
        for(int j = i + 1; j < data.length; j++) {  
            if (data[j] < data[small])  
                small = j;  
        }  
        double temp = data[i];  
        data[i] = data[small];  
        data[small] = temp;  
    }  
}
```

- A. 0
- B. 1
- C. 2
- D. 3
- E.  $\geq 4$

# Why interfaces?

- ▶ Interfaces allow the creation of *abstract types*
  - "A set of data values and associated operations that are precisely specified independent of any particular implementation. "
  - multiple implementations allowed
- ▶ Interfaces allow a data type to be specified without worrying about the implementation
  - do design first
  - What will this data type do?
  - Don't worry about implementation until design is done.
  - separation of concerns.
  - allow us to create *generic algorithms*

# Interfaces

```
public interface List<E> {
```

- ▶ No constructors
- ▶ No instance variables
- ▶ abstract methods
- ▶ default methods
- ▶ static methods
- ▶ class constants (prefer enums)

```
    public static final int DEFAULT_CAP = 10;  
    public void add(E val);
```

# Implementing Interfaces

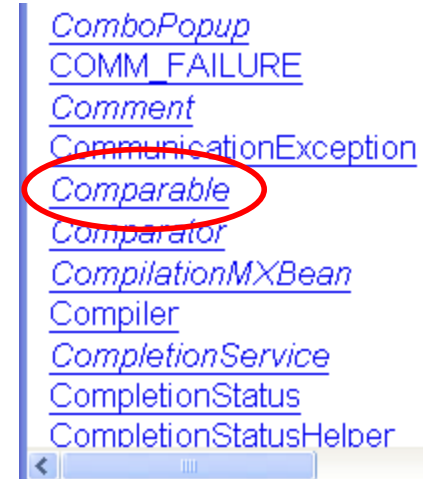
- ▶ In Java, a class inherits (extends) exactly one other class, but ...
- ▶ A class can *implement* as many interfaces as it likes

```
public class ArrayList implements List
```

- ▶ A class that implements an interface must provide implementations of all non default method declared in the interface or the class must be abstract
- ▶ interfaces can extend other interfaces

# The Comparable Interface

- ▶ The Java Standard Library contains a number of interfaces
  - names are italicized in the class listing
- ▶ One of the most important interfaces is the Comparable interface



# Comparable Interface

```
package java.lang;

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

- ▶ `compareTo` must return
  - an int  $< 0$  if the calling object is less than the parameter,
  - 0 if they are equal
  - an int  $> 0$  if the calling object is greater than the parameter
- ▶ `compareTo` should be *consistent with equals* but this isn't required.

# Interfaces

- ▶ "Use interfaces to ensure a class has methods that ***other*** classes or methods will use." (In other words, clients of your class.)
  - Anthony, Spring 2013
- ▶ The other classes or methods may already be written.
- ▶ The other methods or classes use interface type for the parameters of methods.
- ▶ POLYMORPHISM
  - old code using new code



# Clicker Question 2

► What is output by the following code?

```
Comparable c1 = new Comparable();  
Comparable c2 = new Comparable();  
System.out.println(c1.compareTo(c2));
```

- A. A value < 0
- B. 0
- C. A value > 0
- D. Unknown until program run
- E. Compile error

# Example compareTo

- ▶ Suppose we have a class to model playing cards
  - Ace of Spades, King of Hearts, Two of Clubs
- ▶ each card has a suit and a value, represented by ints
- ▶ this version of `compareTo` will compare values first and then break ties with suits



# compareTo in a Card class

```
public class Card implements Comparable<Card> {  
    public int compareTo(Card otherCard) {  
        return this.rank - other.rank;  
    }  
    // other methods not shown  
}
```

Assume ints for ranks (2, 3, 4, 5, 6,...) and suits (0 is clubs, 1 is diamonds, 2 is hearts, 3 is spades).

# Interfaces and Polymorphism

- ▶ Interfaces may be used as the data type for object variables
- ▶ Can't simply create objects of that type
- ▶ Can refer to any objects that implement the interface or descendants
- ▶ Assume `Card` implements `Comparable`

```
Card c = new Card();
```

```
Comparable comp1 = new Card();
```

```
Comparable comp2 = c;
```

# Clicker Question 3

- ▶ Which of the following lines of code causes a syntax error?

`Comparable c1; // A`

`c1 = "Ann"; // B`

`Comparable c2 = "Kelly"; // C`

`int x = c2.compareTo(c1); // D`

`// E No syntax errors.`

`// what is x after statement?`

# Why Make More Work?

- ▶ Why bother implementing an interface such as Comparable
  - objects can use method that expect an interface type
- ▶ Example if I implement Comparable:  
Arrays.sort(Object[] a)  
public static void sort(Object[] a)  
All elements in the array must implement the Comparable interface. Furthermore, all elements in the array must be *mutually comparable*
- ▶ objects of my type can be stored in data structures that accept Comparables

# A List Interface

- ▶ What if we wanted to specify the operations for a List, but no implementation?
- ▶ Allow for multiple, different implementations.
- ▶ Provides a way of creating *abstractions*.
  - a central idea of computer science and programming.
  - specify "what" without specifying "how"
  - "Abstraction is a mechanism and practice to reduce and factor out details so that one can focus on a few concepts at a time. "

# List Interface

```
public interface List <E> {  
    public void add(E val);  
    public int size();  
    public E get(int location);  
    public void insert(int location, E val);  
    public E remove(int location);  
}
```



# One Sort

```
public static void sort(Comparable[] data) {  
    final int LIMIT = data.length - 1;  
    for(int i = 0; i < LIMIT; i++) {  
        int small = i;  
        for(int j = i + 1; j < data.length; j++) {  
            int d = data[j].compareTo(data[small]);  
            if( d < 0)  
                small = j;  
        }  
        Comparable temp = data[i];  
        data[i] = data[small];  
        data[small] = temp;  
    } // end of i loop  
}
```