The “contract” between designer and implementer

 Interfaces

What is an Interface?

- A Java interface is composed of a collection of abstract methods and constants

```java
public interface Doable {
    public void doThis();
    public int doThat(int num);
    public boolean doTheOther();
}
```

Since all methods in an interface are abstract, the keyword `abstract` is left off.

None of the methods in an interface are given a definition (body).

A semicolon immediately follows each method header.
Interfaces

- Why may an interface not be instantiated?
- Why are interface methods public by default?
- Why must a class implementing an interface, define all methods in the interface?
- Why may a class implementing an interface also implement other methods?

```
public class CanDo implements Doable {
    public void doThis () {
        // code to do this
    }
    public void doThat (int num) {
        // code to do that
    }
    public boolean doTheOther () {
        // whatever
    }
}
```

Encryptable.java

```java
/**
 * Represents the interface for an object that can be encrypted and decrypted.
 * @author Java Foundations
 */
public interface Encryptable {
    public void encrypt();
    public String decrypt();
}
```
If Caesar had anything confidential to say, he wrote it in cipher, that is, by so changing the order of the letters of the alphabet, that not a word could be made out. If anyone wishes to decipher these, and get at their meaning, he must substitute the fourth letter of the alphabet, namely D, for A, and so with the others.

Caesar cipher

* Replace each letter by the letter that comes some fixed distance before or after it in the alphabet.

Shift = 3

GALLIA EST OMNIS DIVISA IN PARTES TRES

JDOOLD HVW RPQLV GLYLVD LQ SDUWHV WUHV
Still in use by some...

Mafioso Bernardo Provenzano Captured
by Roberto Paglia

Tuesday, 11 April 2006 will become infamous for two things. It was the day that Italy's right-centre government, with Silvio Berlusconi at the helm (serving longer than any other prime minister in post-war history), was marginally defeated by a shaky leftist coalition, and it was the day that Bernardo Provenzano, supreme head of Sicily's Mafia, was finally captured following forty-three years evading police "underground" in Italy, France and elsewhere. He was captured in Sicily, just a few miles from Corleone, centre of his clan's power. The "boss of bosses" has moved frequently—almost constantly—over the years, maintaining contact with his underlings through the use of scribbled notes. He reached his place at the pinnacle of Sicilian organised crime only in 1993, following the capture of Salvatore "Toto" Riina.

SecretTest.java

```java
/**
 * Demonstrates the use of a formal interface.
 * @author Java Foundations
 */
public class SecretTest {

    /**
     * Creates a Secret object and exercises its encryption.
     */
    public static void main(String[] args) {
        Secret hush = new Secret("GALLIA EST OMNIS DIVISA IN PARTES TRES");
        System.out.println(hush);

        hush.encrypt();
        System.out.println(hush);
        // prints JAVAQKPKVXPTQLV#GLYLV#LQ#SDUMV#WJHV

        hush.decrypt();
        System.out.println(hush);
        // prints GALLIA EST OMNIS DIVISA IN PARTES TRES
    }
}
```
File: Secret.java

```java
import java.util.Random;

/**
 * Represents a secret message that can be encrypted and decrypted.
 * @author Java Foundations
 */
public class Secret implements Encryptable {
    private String message;
    private boolean encrypted;
    private int shift;
    private Random generator;

    /**
     * Constructor: Stores the original message and establishes
     * a value for the encryption shift.
     */
    public Secret (String msg) {
        message = msg;
        encrypted = false;
        generator = new Random();
        shift = generator.nextInt(10) + 5;
    }

    /**
     * Encrypts this secret using a Caesar cipher. Has no effect if
     * this secret is already encrypted.
     */
    public void encrypt () {
        if (!encrypted) {
            String masked = "";
            for (int index=0; index < message.length(); index++)
                masked = masked + (char)(message.charAt(index)+shift);
            message = masked;
            encrypted = true;
        }
    }

    /**
     * Decrypts and returns this secret. Has no effect if this
     * secret is not currently encrypted.
     */
    public String decrypt() {
        if (encrypted) {
            String unmasked = "";
            for (int index=0; index < message.length(); index++)
                unmasked = unmasked + (char)(message.charAt(index)-shift);
            message = unmasked;
            encrypted = false;
        }
        return message;
    }
}
```
Polymorphism via Interfaces

- An interface name can be used as the type of an object reference variable
  ```java
  Speaker current;
  ```
- The current reference can be used to point to any object of any class that implements the Speaker interface
- The version of speak that the following line invokes depends on the type of object that current is referencing
  ```java
  current.speak();
  ```

Suppose two classes, Philosopher and Politician, both implement the Speaker interface, providing distinct versions of the speak method

- In the following code, the first call to speak invokes one version and the second invokes another
  ```java
  Speaker guest = new Philosopher();
  guest.speak();
  guest = new Politician();
  guest.speak();
  ```
Java Interfaces

* The Java standard class library has many helpful interfaces

```java
java.lang
Interface Comparable<T>
```

**Method Summary**

```java
int compareTo(T o)

// Compares this object with the specified object for order.
```

* The `Comparable` interface contains one abstract method called `compareTo`, which is used to compare two objects

* The `String` class implements `Comparable`, giving us the ability to put strings in lexicographic order

The Comparable Interface

* Any class can implement `Comparable` to provide a mechanism for comparing objects of that type

```java
if (obj1.compareTo(obj2) < 0)
    System.out.println("obj1 is less than obj2");
```

* It’s up to the programmer to determine what makes one object < than another

* You may define the `compareTo` method of an `Employee` class to order employees by name (alphabetically) or by employee number

* The implementation of the method can be as straightforward or as complex as needed for the situation
**String implements Comparable**

- The String class contains a method called `compareTo` to determine if one string comes before another

- A call to `name1.compareTo(name2)`
  - returns 0 if `name1` and `name2` are equal (contain the same characters)
  - returns a negative value if `name1` is less than `name2`
  - returns a positive value if `name1` is greater than `name2`

```java
if (name1.compareTo(name2) < 0)
    System.out.println(name1 + " comes first");
else
    if (name1.compareTo(name2) == 0)
        System.out.println("Same name");
    else
        System.out.println(name2 + " comes first");
```

**The Iterator Interface**

- An iterator is an object that provides a means of processing a collection of objects, one at a time

- It is created formally by implementing the Iterator interface’s 3 methods
  - The `hasNext` method returns a boolean – true if there are items left to process
  - The `next` method returns the next object in the iteration
  - The `remove` method removes the object most recently returned by the `next` method

```java
Scanner
Iterator
```

- By implementing the Iterator interface, a class formally establishes that objects of that type are iterators

- Once established, the for-each version of the for loop can be used to process the items in the iterator