CS102

Introduction to data structures, algorithms, and object-oriented programming

April 13, 2016
Interfaces

• In order for objects to interact, they must "know" about the public methods each supports.

• Java application programming interface (API) requires classes to specify the interface they present to other objects.

• The major structural element in Java that supports an API is the interface

  ==> Collection of method declarations and/or constants with no variables and no method bodies. Class full of empty methods!
Iterable Interface

- The Iterable interface (java.lang.Iterable) is one of the root interfaces of the Java collection classes. The Collection interface extends Iterable, so all subtypes of Collection also implement the Iterable interface.

The Iterable interface has only one method:

```java
public interface Iterable<T> {
    public Iterator<T> iterator();
}
```
Iterable Interface

• A class that implements the Iterable can be used with the new for-each loop. Here is such an example:

```java
List list = new ArrayList();

for(Object o : list){
    //do something to o;
}
```
Multiple Inheritance

The ability to derive a class from more than one parent class is known as *multiple inheritance*.

```
class Person {...}
class Employee extends Person, Democrat {...}
class Democrat {...}
```

Multiple inheritance is NOT ALLOWED in Java (i.e., a class can't extend more than one other class)
A Java Provision for Multiple Inheritance...

**class** Person {...}

**interface** Manager {...}

**interface** Democrat {...}

**class** Employee **extends** Person **implements** Manager, Democrat {...}

The Employee class would be a subclass of Person and a subtype of Manager and Democrat. We could write a program that makes use of Employee objects anywhere Person, Manager, or Democrat objects are required!
Another Way Java Provides for Multiple Inheritance...

Interfaces can extend multiple interfaces

```
interface Democrat {...}
interface Farmer {...}
interface Senator extends Democrat, Farmer {...}
```

```
class Person {...}
```

```
class Employee extends Person implements Senator {...}
```

The Employee class would be a subclass of Person and a subtype of Senator.
Multiple Interfaces

When a class implements an interface that extends another interface, it must include all methods from each interface in hierarchy.

Faculty and Staff must include methods of both Retirement and StatePlan interfaces.
Dynamic Binding and Polymorphism

Even though you can't instantiate an object from an interface, you can assign an object of a subtype to a variable of its supertype (i.e., an interface the object implements).

```java
StatePlan s = new Faculty ( "Joe", 26, "1 Main St", "845-555-1212", 10000.0 );
StatePlan e = new Staff ( "Boss", 42, "4 Main St", "854-555-1212", 10000.0 );
StatePlan p = null;
if( ((int)(Math.random() * 10)) % 2 == 1 )
    p = s;
else
    p = e;
System.out.println( "Person is " + p.toString() );
```

Do not know until program runs whether to use `Faculty's toString` or `Staff's toString`
Casting in an Inheritance Hierarchy

- The "instanceof" operator allows us to test whether an object variable is referring to an object of a certain class, derived from a certain class, or implementing a certain interface.

```java
StatePlan s = new Faculty ( "Joe", 26, "1 Main St", "845-555-1212", 10000.0 );
StatePlan e = new Staff ( "Boss", 42, "4 Main St", "854-555-1212", 10000.0 );
StatePlan p = null;
if( ((int)(Math.random() * 10)) % 2 == 0 )
  p = s;
else
  p = e;
if (p instanceof Staff) Staff f = (Staff)p;
```
Abstract Classes

• Abstract classes lie between interfaces and complete classes.

  ==> Class that may contain empty method declarations as well as fully defined methods and instance variables.

  ➢ Not possible to instantiate an abstract class.
  ➢ Subclasses must provide an implementation for each abstract method in the parent class.
  ➢ "Partial" implementation of a class. Derived classes complete the definition.

```java
public abstract class Matrix implements Graph {...}
```
An Abstract Class

The purpose of an abstract class is to define inheritable, shared variables and methods and to impose requirements through abstract methods.

```java
public abstract class Attraction {
    public int minutes;
    public Attraction() {minutes = 75;}
    public Attraction(int m) {minutes = m;}
    public int getMinutes() {return minutes;}
    public void setMinutes(int m) {minutes = m;}
    public abstract int rating();
}
```

Any classes derived from Attraction would inherit the public members and would have to provide an implementation of the abstract method `rating`.
A Class Derived from Attraction

```java
public class Movie extends Attraction {
    public int script, acting, direction;

    public Movie() {script=5; acting=5; direction = 5;}

    public Movie(int m) {super(m);}

    public int rating() {
        return script+acting+direction+getMinutes();
    }
}
```

Any classes derived from Attraction would inherit the public members and would have to provide an implementation of the abstract method `rating`. 
```java
class GenericArray {
    public static void main (String[] args) {
        Object[] array = new Object[4];
        array[0] = "String 1";
        array[1] = new Integer(1);
        array[2] = new Person();
        array[3] = new Integer("57");
        for (int i = 0; i < array.length; i++) {
            if (array[i] instanceof String) {
                String temp = (String)array[i];
                System.out.println(temp);
            }
            else if (array[i] instanceof Integer) {
                int x = ((Integer)array[i]).intValue();
                System.out.println(x);
            }
            else if (array[i] instanceof Person) {
                int y = ((Person)array[i]).getAge();
                System.out.println(y);
            }
        }
    }
}
```

Example of creating array of Objects and testing and casting each before printing
File Input & Output (File I/O)

• File I/O in Java can be accomplished by using one of many built-in Java classes.
  – Reading input from a file:
    • BufferedReader inFile = new BufferedReader(new FileReader("input.dat"));
      Function: Reads text as a stream of characters.
  
  – Writing output to a file:
    • PrintWriter outFile = new PrintWriter(new FileWriter("output.dat"));
      Function: Prints formatted representations of objects to text output stream.
import java.io.*;

public class TestReadWrite {
    public static void main (String[] args) throws IOException{
        BufferedReader inFile = new BufferedReader(new FileReader("input.dat"));
        PrintWriter outFile = new PrintWriter(new FileWriter("output.dat"));
        StreamTokenizer token = new StreamTokenizer(inFile);

        token.nextToken();

        int number = ((int) token.nval);

        outFile.println("The number read is "+number);

        outFile.close();
        inFile.close();
    }
} // end of class
File I/O example using Command line arguments

```java
import java.io.*;

public class TestReadWrite {
    public static void main (String[] args) throws IOException{
        BufferedReader inFile = new BufferedReader
            (new FileReader (args[0]));
        PrintWriter outFile = new PrintWriter
            (new FileWriter (args[1]));
        StreamTokenizer token = new StreamTokenizer(inFile);
        token.nextToken();
        int number = ((int) token.nval);
        outFile.println("The number read is "+number);
        outFile.close();
        inFile.close();
    } // end of main
} // end of class
```
import java.io.*;

public class TestReadWrite {
    public static void main (String[] args) throws IOException{
        BufferedReader inFile = new BufferedReader
        (new FileReader (args[0]));
        PrintWriter outFile = new PrintWriter
        (new FileWriter (args[1]));
        StreamTokenizer token = new StreamTokenizer(inFile);
        token.nextToken();
        int number = ((int) token.nval);
        outFile.println("The number read is "+number);
        outFile.close();
        inFile.close();
    } // end of main
} // end of class

args[0] and args[1] are Strings read from the command line.

nextToken() looks for characters separated by white space.
token.nval returns the value of the token as a double (cast as an int here).
import java.io.*;

public class TestReadWrite {
    public static void main (String[] args) throws IOException{
        BufferedReader inFile = new BufferedReader(new FileReader(args[0]));

        PrintWriter outFile = new PrintWriter(new FileWriter(args[1]));

        StreamTokenizer token = new StreamTokenizer(inFile);
        token.nextToken();

        int number = ((int) token.nval);

        outFile.println("The number read is "+number);

        outFile.close();
        inFile.close();
    }
} // end of class
Reading Command-Line Arguments

- Command-line arguments are read through the main method's array of Strings parameter, args.

- For example, to run the TestReadWrite program shown on the 2 previous slides, we would type

  java TestReadWrite input.dat output.dat

**IMPORTANT:** You must have a non-empty file in the same directory as the TestReadWrite.class file called "input.dat" when you run this program!! Also, any file called "output.dat" in the current directory will be overwritten.

- In the TestReadWrite program, args[0] = "input.dat" and args[1] = "output.dat" during execution of the program.
Taking Input from the Console

```java
import java.io.*;

public class ChangeCoinsApp {
    public static void main (String args[]) throws IOException {
        BufferedReader stdin =
            new BufferedReader(new InputStreamReader(System.in));

        System.out.println("Enter a number: ");

        String line = stdin.readLine();
        int num = Integer.parseInt(line);
    }
}
```

- **Add 2 words to main method declaration**
- **Declare an object that takes input from the keyboard.**
- **Read the String and convert it to an int using the static parseInt method of the Integer class**
Reading & Writing Strings in a Loop

```java
import java.io.*;
public class TestReadWrite {
    public static void main (String[] args) throws IOException{
        BufferedReader fileIn = new BufferedReader
            (new FileReader("pal.txt"));
        PrintWriter outFile = new PrintWriter
            (new FileWriter("palindrome.txt"));

        String line, lcString;
        StringTokenizer str;
        while ((line = fileIn.readLine()) != null) {
            str = new StringTokenizer
                (line.toLowerCase()," ,:.;!?"");
            while (str.hasMoreTokens()) {
                lcString = str.nextToken();
                outFile.println(lcString);
            } // end inner while
        } // end outer while
        fileIn.close(); // close input file
        outFile.close(); // close output file
    } // end main
} //end class
```