Principles of Computer Science II

Prof. Nadeem Abdul Hamid

CSC 121 – Spring 2006

Lecture Unit 2 - Interfaces and Polymorphism

1

Lecture Outline

- Interfaces
- Polymorphism
- Inner classes
- Event-listeners

2

```
/** Computes various statistics of a set of data values */

public class DataSet {

   private double sum;
   private double maximum;
   private int count;

   /** Constructs an empty data set. */
   public DataSet() {
      sum = 0;
      count = 0;
      maximum = 0;
   }

   /** Adds a data value to the data set
       @param x a data value */
   public void add(double x) {
      sum = sum + x;
      if (count == 0 || maximum < x) maximum = x;
      count++;
   }

   /** Gets the average of the added data.
       @return the average or 0 if no data has been added */
   public double getAverage() {
      if (count == 0) return 0;
      else return sum / count;
   }

   /** Gets the largest of the added data.
       @return the maximum or 0 if no data has been added */
   public double getMaximum() {
      return maximum;
   }
}
```

3

```
// modified for handling a set of BankAccounts
public class BankDataSet {

   private double sum;
   private BankAccount maximum;
   private int count;

   ...

   public void add(BankAccount x) {
      sum = sum + x.getBalance();
      if (count == 0 || maximum.getBalance() < x.getBalance())
            maximum = x;
      count++;
   }

   public BankAccount getMaximum() {
      return maximum;
   }

}
```

4

```
// modified for handling a set of Coins
public class CoinDataSet {

   private double sum;
   private Coin maximum;
   private int count;

   ...

   public void add(Coin x) {
      sum = sum + x.getValue();
      if (count == 0 || maximum.getValue() < x.getValue())
            maximum = x;
      count++;
   }

   public Coin getMaximum() {
      return maximum;
   }

}
```

5

DataSet Issues

- The fundamental method of analyzing the data is the same
- Details of how to determine the value of data (bank accounts/coins) differ
- Suppose classes agreed on a common method for returning the *measure* needed for data analysis
  - `getMeasure()` - returns balance for bank accounts
  - `getMeasure()` - returns value for coins
- *add* method looks like:

```
sum = sum + x.getMeasure();
if (count == 0 || maximum.getMeasure() < x.getMeasure())
      maximum = x;
count++;
```

6

Interface Types

- What is the type of x in the `add` method?
  - Should be any class that has a `getMeasure` method
- In Java, use *interface* types to specify required operations that must be supported by a set of classes

```
public interface Measurable {

   double getMeasure();

   // other required methods ...
}
```

7

Interfaces vs. Classes

- All methods in interface type are *abstract* -- i.e. have header (name,params,return) but no implementation (body)
  - Just a semicolon following the method header
- All methods in interface are automatically public
  - `public double getMeasure();`
- Interface cannot have instance fields

8

```
/** Computes various statistics of a set of data values */

public class DataSet {
   private double sum;
   private Measurable maximum;
   private int count;

   /** Constructs an empty data set. */
   public DataSet() {
      sum = 0;
      count = 0;
      maximum = null;
   }

   /** Adds a data value to the data set
       @param x a data value */
   public void add(Measurable x) {
      sum = sum + x.getMeasure();
      if (count == 0 || maximum.getMeasure() < x.getMeasure())
               maximum = x;
      count++;
   }

   /** Gets the average of the added data.
       @return the average or 0 if no data has been added */
   public double getAverage() {
      if (count == 0) return 0;
      else return sum / count;
   }

   /** Gets the largest of the added data.
       @return the maximum or 0 if no data has been added */
   public Measurable getMaximum() {
      return maximum;
   }
}
```

9

```
public class BankAccount implements Measurable {
   private double balance;

   public BankAccount( double bal ) {
      balance = bal;
   }

   // other BankAccount methods: withdraw, deposit, getBalance, ...

   public double getMeasure() { return balance; }
}



public class Coin implements Measurable {
   private double value;
   private String name;

   public Coin( double val, String n ) {
      value = val;
            name = n;
   }

   // other Coin methods ...

   public double getMeasure() { return value; }
}
```

10

```
/** This program tests the DataSet class. */
public class DataSetTester {

   public static void main(String[] args) {

      DataSet bankData = new DataSet();

      bankData.add(new BankAccount(0));
      bankData.add(new BankAccount(10000));
      bankData.add(new BankAccount(2000));

      System.out.println("Average balance = " + bankData.getAverage());
      Measurable max = bankData.getMaximum();
      System.out.println("Highest balance = " + max.getMeasure());


      DataSet coinData = new DataSet();

      coinData.add(new Coin(0.25, "quarter"));
      coinData.add(new Coin(0.1, "dime"));
      coinData.add(new Coin(0.05, "nickel"));

      System.out.println("Average coin value = " + coinData.getAverage());
      max = coinData.getMaximum();
      System.out.println("Highest coin value = " + max.getMeasure());

   }

}
```

11

Benefit of Interfaces

- `Measurable` interface expresses what all measurable objects have in common
  - Allows for reusable version of DataSet class - can analyze collections of objects of any class that supports Measurable interface
- Interfaces can reduce coupling (dependencies) between classes
- Use the `implements` keyword to indicate that a class supports all operations of an interface type

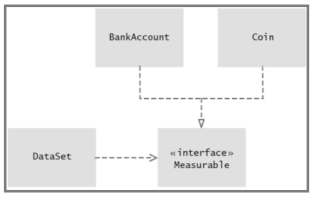
12

Why Use Interfaces?

- Because interfaces “institutionalize” a standard set of method prototypes in a software product, requiring teams of programmers to follow an established template or pattern in developing a group of related classes.
- Standardization is good – it can **improve product quality**, **reduce development time**, and **simplify maintenance of code** throughout its lifecycle, especially for a very large, complex program!
- This provides project managers and senior code developers with some control over a team of programmers developing many classes

13

Interfaces: UML Diagram



- Interfaces are tagged with a “stereotype” indicator «interface»
- A dotted arrow with a triangular tip denotes the “is-a” relationship between a class and an interface
- A dotted line with an open v-shaped arrow tip denotes the “uses” relationship or dependency
- Note: DataSet is decoupled from BankAccount and Coin

14

Syntax: Interfaces

- Syntax figures 11.1 and 11.2 (page 414)

// defining…

```
public interface InterfaceName
{
   method signatures
}
```

// implementing…

```
public class ClassName
       implements InterfaceName, InterfaceName, ...
{
   methods and fields
}
```

15

Constants in Interfaces

- Interfaces cannot have instance fields, but may define *constants*
- Constants defined in interfaces automatically defined as `public static final`
- Example:

```
public interface SwingConstants {

   public static final int NORTH = 1;
   public static final int NORTHEAST = 2;
   public static final int EAST = 3;
   . . .
}
```

16

Converting Between Class and Interface Types

- You can convert from a class type to an interface type, provided the class implements the interface

```
BankAccount account = new BankAccount(10000);
Measurable x = account; // OK

Coin dime = new Coin(0.1, "dime");
Measurable x = dime; // Also OK
```

- Cannot convert between unrelated types

```
Measurable x = new Rectangle(5, 10, 20, 30); // ERROR
```

17

Converting from Interface to Class Type

```
DataSet coinData = new DataSet();
coinData.add( new Coin( 0.25, "quarter" ) );
coinData.add( new Coin( 0.1, "dime" ) );
coinData.add( new Coin( 0.05, "nickel" ) );
Measurable max = coinData.getMaximum();

Coin maxCoin = max; // ERROR
String maxCoinName = maxCoin.getName();

Coin maxCoin = (Coin) max; // OK
String maxCoinName = maxCoin.getName();
```

- Compiler can’t tell for certain that `max` refers to a Coin object - although you can
- Need a cast to convert from interface type to class type
  - If you are wrong and max *isn’t* a Coin object, the program throws an exception when you run it

18

Casts: Numbers and Objects

- Casting is used to convert between types
- When casting number types you agree to the information loss

```
double average = 6.7 / 3.4;
int avg = (int) average;
```

- When casting object types you agree to the risk of causing an exception

19

Common Error: Instantiating Interfaces

- You can defined variables of an interface type:

```
Measurable x;
```

- But you can *never* construct an interface 'object':

```
Measurable x = new Measurable(); // ERROR
```

- You can only construct objects of a class that implements the interface:

```
Measurable x = new BankAccount(); // OK
```

20

Polymorphism

- ***Definition:*** Principle that behavior of some code varies depending on the actual type of an object
- Interface variable holds reference to object of a class that implements the interface

```
Measurable x;
...
x = new Coin( 0.1, "dime" );
...
x = new BankAccount( 10000.0 );
```

- When you call a method of the interface,

```
double m = x.getMeasure();
```

which method is actually called?

21

Polymorphism (cont.)

- Method called depends on the actual object
  - If x refers to a BankAccount, calls BankAccount class' implementation of getMeasure()
  - If x refers to a Coin, calls Coin class' implementation of getMeasure()
- Also referred to as *late-binding*: selection of the actual method takes place at run-time
- Overloaded methods resolved using *early-binding*: i.e. determined at compile-time

22

Making DataSet More Reusable

- Limitations of `DataSet` handling `Measurable` objects
  - Can only add `Measurable` interface to classes under your control (e.g. can't redefine the standard API `Rectangle` class)
  - Can only measure an object in one way -- i.e. each class provides a single implementation of `getMeasure()` method
    - e.g. Sometimes may want to analyze BankAccounts based on balance; sometimes based on interest rate

23

Callbacks

- Mechanism for one component (e.g. DataSet) to invoke a method (e.g. getMeasure()) on another component (e.g. Rectangle) without having been written in terms of, or with knowledge of, the other component's type.
- Section 11.4 (in-class exercise)

24