**Further Object Oriented Programming**

U08026

David Sutton

Department of Computing and Communication Technologies

# Interfaces and Abstract Classes

# Introduction

After you have completed this session, you will (hopefully!):

* Understand how you can use interfaces and abstract classes to define what methods a class will have, without saying how those methods will be implemented.
* Understand why you would want to do that.

# Exercise 1

Create a new NetBeans project. Add the SimpleStats class from the lecture slides (set out in ). Add the main class set out in . Build and run the project and check that it behaves in the way that you would expect it to.

Table The SimpleStats class

|  |
| --- |
| public class SimpleStats {  private double sum, lastElement;  private int count;  public double getSum() {  return sum;  }  public int getCount() {  return count;  }  public double getAverage() {  return getSum() / getCount();  }  public void add(double element) {  lastElement = element;  count++;  sum += element;  }  public double getLastElement() {  return lastElement;  }  } |

Table Main Class for Exercise 1

|  |
| --- |
| import java.util.Scanner;  public class Main {  private static SimpleStats myStats = new SimpleStats();  private static Scanner scan = new Scanner(System.in);  public static void main(String[] args) {  String option;  do {  System.out.println("Options are:");  System.out.println("A: Add a value");  System.out.println("P: Print out the statistics of the collection");  System.out.print("Enter your option > ");  option = scan.nextLine();  if (option.equalsIgnoreCase("A")) {  addValue(myStats);  }  if (option.equalsIgnoreCase("P")) {  printStats(myStats);  }  System.out.println();  } while (!option.equalsIgnoreCase("Q"));  }  private static void addValue(SimpleStats stats) {  System.out.print("Enter value to be added > ");  double d = scan.nextDouble();  stats.add(d);  }  private static void printStats(SimpleStats stats) {  System.out.println("Count = " + stats.getCount());  System.out.println("Sum = " + stats.getSum());  if (stats.getCount() > 0) {  System.out.println("Average = " + stats.getAverage());  }  }  } |

# Exercise 2

Add to your project the ArrayStats class, as set out in the lectures and in . Modify the Main class so that it produces the same output as before, but creates and uses and ArrayStats object instead of a SimpleStats one. Make a note of the changes that you need to make.

Table ArrayStats Class

|  |
| --- |
| public class ArrayStats {  public static final int MAX\_ELEMENTS = 1000;  protected double[] elements = new double[MAX\_ELEMENTS];  private int count;  public double getElement(int i) {  return elements[i];  }  public double getSum() {  double sum = 0;  for (int i = 0; i < count; i++) {  sum += getElement(i);  }  return sum;  }  public int getCount() {  return count;  }  public double getAverage() {  return getSum() / getCount();  }  public void add(double element) {  elements[count] = element;  count++;  }  } |

# Exercise 3

Add to your project the Stats interface from the lecture slides (). You can create an interface in NetBeans by right clicking on the package it will belong to and selecting New->Java Interface.

Modify the ArrayStats and SimpleStats classes so that they implement the Stats interface. To do this you simply modify the first line of the class so that the words “implements Stats” appear after its name.

Now modify the main class of your project to contain the code shown in Table 5. Run the project and check that it does what you would expect.

Notice that the parameter of the addValue and printStats method is of type Stats. This allows us to use the same method to handle SimpleStats and ArrayStats objects. Notice that Stats is also used as the type of the fields mySimpleStats and myArrayStats,

Table The Stats Interface

|  |
| --- |
| public interface Stats {  public double getSum();  public int getCount();  public double getAverage();  public void add(double element);  } |

Table Main Class for Exercise 3

|  |
| --- |
| import java.util.Scanner;  public class Main {  private static **Stats** mySimpleStats = new SimpleStats();  private static **Stats** myArrayStats = new ArrayStats();  private static Scanner scan = new Scanner(System.in);  public static void main(String[] args) {  String option;  do {  System.out.println("Options are:");  System.out.println("AS: Add a value to the SimpleStats Collection");  System.out.println("PS: Print out SimpleStats statistics");  System.out.println("AA: Add a value to the ArrayStats Collection");  System.out.println("PA: Print out ArrayStats statistics");  System.out.print("Enter your option > ");  option = scan.nextLine();  if (option.equalsIgnoreCase("AS")) {  addValue(mySimpleStats);  }  if (option.equalsIgnoreCase("PS")) {  printStats(mySimpleStats);  }  if (option.equalsIgnoreCase("AA")) {  addValue(myArrayStats);  }  if (option.equalsIgnoreCase("PA")) {  printStats(myArrayStats);  }  System.out.println();  } while (!option.equalsIgnoreCase("Q"));  }  private static void addValue(**Stats** stats) {  System.out.print("Enter value to be added > ");  double d = scan.nextDouble();  stats.add(d);  }  private static void printStats(**Stats** stats) {  System.out.println("Count = " + stats.getCount());  System.out.println("Sum = " + stats.getSum());  System.out.println("Average = " + stats.getAverage());  }  } |

# Exercise 4

Add the AbstractStats class set out in Table 6 to your project. Then change the ArrayStats and SimpleStats classes as follows:

* Replace the words “implements Stats” with “extends AbstractStats”.
* Remove the implementation of the getAverage() method.

Once you have finished the two classes should be as set out in Table 7 and Table 8.

Note that ArrayStats and SimpleStats still implement the Stats interface. We do not need to say so explicitly because they extend a class that implements that interface.

Note also that you can still call the getAverage() method on an ArrayStats or SimpleStats object. It inherits an implementation of that method from its superclass.

You do **not** need to modify the Main class. You should find that your project will compile correctly and behave in exactly the same way as before. Check that.

Table AbstractStats Class

|  |
| --- |
| public abstract class AbstractStats implements Stats {  public abstract double getSum();  public abstract int getCount();  public double getAverage() {return getSum()/getCount();};  public abstract void add(double element);  } |

Table SimpleStats Class for Exercise 4

|  |
| --- |
| public class SimpleStats extends AbstractStats {  private double sum, lastElement;  private int count;  public double getSum() {  return sum;  }  public int getCount() {  return count;  }  public void add(double element) {  lastElement = element;  count++;  sum += element;  }  public double getLastElement() {  return lastElement;  }  } |

Table ArrayStats Class for Exercise 4

|  |
| --- |
| public class ArrayStats extends AbstractStats {  public static final int MAX\_ELEMENTS = 1000;  protected double[] elements = new double[MAX\_ELEMENTS];  private int count;  public double getElement(int i) {  return elements[i];  }  public double getSum() {  double sum = 0;  for (int i = 0; i < count; i++) {  sum += getElement(i);  }  return sum;  }  public int getCount() {  return count;  }  public void add(double element) {  elements[count] = element;  count++;  }  } |

# Exercise 5

In this exercise we will add the Bounds interface to our project and create two classes that implement it. The first will just implement the Bounds interface, and the second will implement the Stats interface as well.

1. Add the Bounds Interface to your project (Table 9).
2. Add the SimpleBounds class shown in Table 10. This class implements only the Bounds interface.
3. Add the BStats class shown in the lecture notes and Table 11. This class implements both the Bounds and Stats interfaces.
4. Add the TestBounds class shown in Table 12. Note that the Bounds interface is used as the parameter type for its showBounds method. This allows us to pass the method any object that implements the Bounds interface.
5. Run the TestBounds class. You can do this by right clicking on it in the projects window and then selecting “Run File”. Check that the results are as you would expect.

Table The Bounds Interface

|  |
| --- |
| public interface Bounds {  public double getMin();  public double getMax();  public void add(double element);  } |

Table The SimpleBounds Class

|  |
| --- |
| public class SimpleBounds implements Bounds {  private double min = Double.MAX\_VALUE;  private double max = Double.MIN\_VALUE;  public double getMin() {  return min;  }  public double getMax() {  return max;  }  public void add(double element) {  if (element > max) {  max = element;  }  if (element < min) {  min = element;  }  }  } |

Table The BStats Class

|  |
| --- |
| public class BStats implements Stats, Bounds {  private Stats stats;  private double min = Double.MAX\_VALUE;  private double max = Double.MIN\_VALUE;  public BStats(Stats stats) {  this.stats = stats;  }  public double getMin() {  return min;  }  public double getMax() {  return max;  }  public void add(double element) {  if (element > max) {  max = element;  }  if (element < min) {  min = element;  }  stats.add(element);  }  public double getSum() {  return stats.getSum();  }  public int getCount() {  return stats.getCount();  }  public double getAverage() {  return stats.getAverage();  }  } |

Table A Test Class

|  |
| --- |
| public class TestBounds {  public static void main(String[] args) {  Bounds simple = new SimpleBounds();  Bounds bstats = new BStats(new ArrayStats());  simple.add(-1);  simple.add(450);  simple.add(1);  bstats.add(56);  bstats.add(34);  bstats.add(1);  System.out.println("SimpleBounds");  showBounds(simple);  System.out.println();  System.out.println("BStats");  showBounds(bstats);  }  private static void showBounds(Bounds bounds) {  System.out.println("Min = " + bounds.getMin());  System.out.println("Max = " + bounds.getMax());  }  } |

# Exercise 6

1. Add the Addable and BoundableStats interfaces to your project.
2. Refactor the Bounds and Stats interfaces so that they extend Addable. Remove the definition of the add method from those interfaces. You don’t need it to be there anymore because it is inherited from Addable.
3. Modify the BStats class so that it implements the single interface BoundableStats.
4. Modify the Main class so that it creates two BStats objects, one of which is associated with a SimpleStats object, and the other with an ArrayStats object
5. Modify the printStats method of the main class so that its argument is a BoundableStats object and it prints out the minimum and maximum values that have been added, as well as the sum, count, and average.
6. Modify the addValue method of the Main class so that the type of its parameter is Addable. This type will work because the only method we need to call on the parameter is the add method. You should always try to give variables the most general type that you can.

The modified classes are set out in Table 13 to Table 18.

Table The Addable Interface

|  |
| --- |
| public interface Addable {  public void add(double element);  } |

Table First Line of BStats Class (remainder of class is unchanged)

|  |
| --- |
| public class BStats implements BoundableStats { |

Table Boundable Stats Interface

|  |
| --- |
| public interface BoundableStats extends Bounds, Stats {  } |

Table Modified Bounds Interface. Note removal of add method.

|  |
| --- |
| public interface Bounds extends Addable {  public double getMin();  public double getMax();  } |

Table Modified Stats Interface. Note removal of add method.

|  |
| --- |
| public interface Stats extends Addable {  public double getSum();  public int getCount();  public double getAverage();  } |

Table Modified Main Class. Note parameter of addValue method.

|  |
| --- |
| import java.util.Scanner;  public class Main {  private static BoundableStats simpleBStats = new BStats(new SimpleStats());  private static BoundableStats arrayBStats = new BStats(new ArrayStats());  private static Scanner scan = new Scanner(System.in);  public static void main(String[] args) {  String option;  do {  System.out.println("Options are:");  System.out.println("AS: Add a value to the SimpleStats Collection");  System.out.println("PS: Print out SimpleStats statistics");  System.out.println("AA: Add a value to the ArrayStats Collection");  System.out.println("PA: Print out ArrayStats statistics");  System.out.print("Enter your option > ");  option = scan.nextLine();  if (option.equalsIgnoreCase("AS")) {  addValue(simpleBStats);  }  if (option.equalsIgnoreCase("PS")) {  printStats(simpleBStats);  }  if (option.equalsIgnoreCase("AA")) {  addValue(arrayBStats);  }  if (option.equalsIgnoreCase("PA")) {  printStats(arrayBStats);  }  System.out.println();  } while (!option.equalsIgnoreCase("Q"));  }  private static void addValue(Addable addable) {  System.out.print("Enter value to be added > ");  double d = scan.nextDouble();  addable.add(d);  }  private static void printStats(BoundableStats stats) {  System.out.println("Count = " + stats.getCount());  System.out.println("Sum = " + stats.getSum());  System.out.println("Average = " + stats.getAverage());  System.out.println("Min " + stats.getMin());  System.out.println("Max = " + stats.getMax());  }  } |

# Exercise 7

1. Define a new interface Trend, which extends the Addable interface and adds one new method

int getTrend()

This method should return one of the following values:

+1 if the last value to be added was greater than the penultimate (next to last) one

-1 if the last value to be added was less than the penultimate one

0 if the last value to be added was equal to the penultimate one

-2 if less than 2 values have so far been added.

1. Define an interface TrendBoundableStats that extends both the Trend and BoundableStats interfaces.
2. Modify the BStats class so that it implements the TrendBoundableStats interface. You will need to implement the getTrend() method. The UML Diagram for the completed exercise is shown overleaf.
3. Modify the Main class so that it is testing the newly added method, as well as all of the existing ones.

# What you should submit

Please submit two files:

1. A zip file containing all of the Java files that you have created in this session (including your answer to Exercise 7).
2. A word file into which you have pasted all of the Java source you have created (including your answer to Exercise 7).