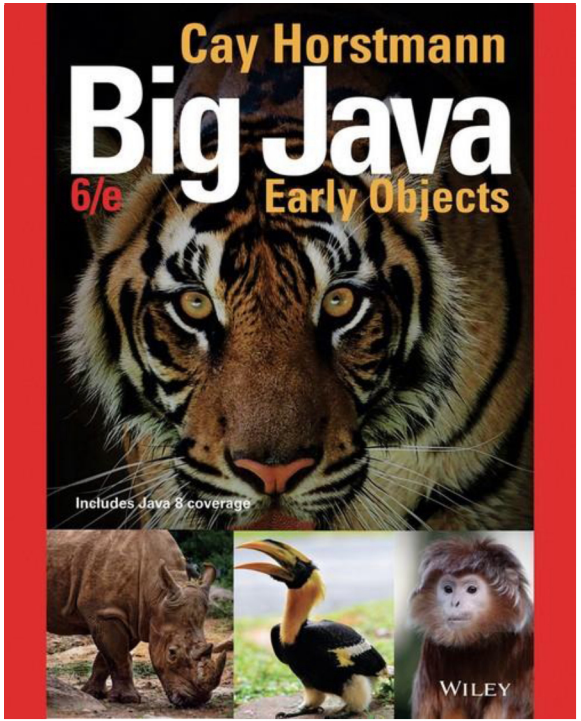


Chapter 10 - Interfaces and Polymorphism Part I

Chapter Goals

- To be able to declare and use Interface types
- To appreciate how interfaces can be used to decouple classes
- To learn how to implement helper classes as inner classes
- To implement event listeners in graphical applications

Using Interfaces for Algorithm Reuse

- **Interface types** are used to express common operations.
- Interfaces make it possible to make a service available to a wide set.
- This restaurant is willing to serve anyone who conforms to the `Customer` interface with `eat` and `pay` methods.

·£> Oxana Oltv nichsnkuTiS ıq-kiiitnu

Defining an Interface Type

- Example: a method to compute the average of an array of Objects
  - The algorithm for computing the average is the same in all cases
  - Details of measurement differ
- Goal: write one method that provides this service.
- We can't call `getBalance` in one case and `getArea` in another.
  - Solution: all object who want this service must agree on a `getMeasure` method
  - `BankAccount's getMeasure` will return the balance
  - `Country's getMeasure` will return the area
- Now we implement a single average method that computes the sum:

```
sum = sum + obj.getMeasure();
```

Defining an Interface Type

- Problem: we need to declare a type for `ob j`
- Need to invent a new type that describes any class whose objects can be measured.
- An interface type is used to specify required operations (like `getMeasure`) :

```
public interface Measurable
{
   double getMeasure();
}
```

- A Java interface type declares methods but does not provide their implementations.

Syntax 10.1 Declaring an Interface

Syntax public interface *InterfaceName*
{
method headers
}

```
public interface Measurable
{
   double getMeasure() ;
}
```

No implementation is provided.

The methods of an interface are automatically public.

Defining an Interface Type

- An interface type is similar to a class.
- Differences between classes and interfaces:
  - An interface type does not have instance variables.
  - All methods in an interface type are `abstract` (or in Java 8, `static` or `default`)
    - They have a name, parameters, and a return type, but no implementation.
  - All methods in an interface type are automatically public.
  - An interface type has no constructor.
    - You cannot construct objects of an interface type.

Defining an Interface Type

- Implementing a reusable average method:

```
public static double average(Measurable[] objects)
{
  double sum = 0;
  for (Measurable obj : objects)
  {
    sum = sum + obj.getMeasure();
  1
  if (objects.length > 0) { returnsum
  / objects.length; } else { return 0;
  1
}
```

- This method is can be used for objects of any class that conforms to the `Measurable` type.

- This stand-mixer provides the “rotation” service to any attachment that conforms to a common interface. Similarly, the average method at the end of this section works with any class that implements a common interface.

Implementing an Interface Type

- Use `implements` reserved word to indicate that a class implements an interface type:

```
public class BankAccount implements Measurable
{
   ...
   public double getMeasure()
   {
      return balance;
   }
}
```

- `BankAccount` objects are instances of the `Measurable` type:

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

Implementing an Interface Type

- A variable of type `Measurable` holds a reference to an object of some class that implements the `Measurable` interface.
- `Country` class can also implement the `Measurable` interface:

```
public class Country implements Measurable
{
  public double getMeasure ( )
  {
    return area;
  }
  . . .
}
```

- Use interface types to make code more reusable.

Implementing an Interface Type

- Put the `average` method in a class - say `Data`

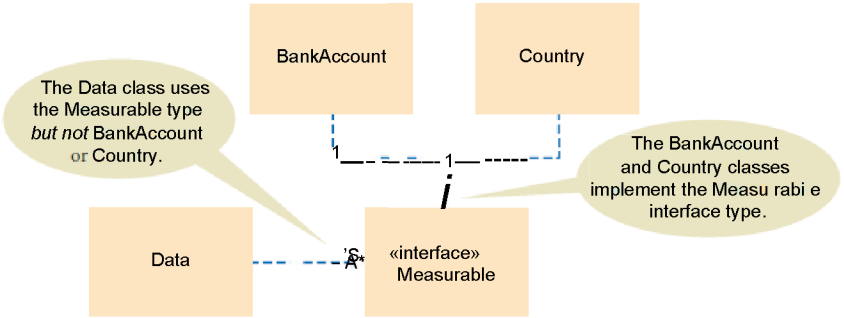


Figure 1 UML Diagram of the Data Class and the Classes that Implement the `Measurable` Interface

- `Data` class is decoupled from the `BankAccount` and `Country` classes.

Syntax 10.2 Implementing an Interface

Syntax public class *ClassName* implements *InterfaceName*, *InterfaceName*, . . .
{
instance variables
methods
}

```
public class BankAccount implements Measurable
{
   . . .
   public double getMeasure() -
   {
      return balance;
   }
   . . .
}
```

List all interface types that this class implements.

BankAccount instance variables

This method provides the implementation for the method declared in the interface.

Other BankAccount methods

sectionjl / Data.java

```
1 public class Liata
2 1
3     /■+■+
4         Computes the average of the measures of the given objects.
5         rparam ob j ects an array          of Measurable objects
í         t return the average of the measures
7     V
8     public    static   double  average(Measurable[] objects)
9     {
10        double sum =     0;
11        for (Measurable obj :       objects)
12        {
13            sum = sum + obj.getMeasure();
14        }
15        if (objects.length > 0) { return sum / objects.length ;            }
16        else { return 0; }'
17    }
18 }
```

section_1 /MeasurableTester.java

```
1  /**
2                  This program demonstrates the measurable BankAccount and Country classes.
3  */
4  public class MeasurableTester
5  {
6             public static void main(String[] args)
7     {
B        Measurable   []   accounts   =   new   Measurable[3];
9        accounts  [0]  =  new  BankAccount(0) ;
10       accounts[1]    =    new    BankAccount(10000);
11       accounts [2] = new BankAccount(2000) ;
12
13
14       double    averageBalance    =    Data.average(accounts);
15       System.out.printin("Average    balance:    "    +
16       averageBalance); System.out.printin("Expected : 4000"          );
17
18       Measurable   []   countries   =   new   Measurable[3];
19       countries[0]    =    new    Country("Uruguay",    176220);
20       countries[1] = new Country("Thailand",
21       5131     20); countries [0] = new
22           Country("Belgium", 30510) ;
23       double averageArea = Data.average(countries);
24       System.out.printin("Average    area:    "    +
25       averageArea) ; System.out.printin("Expected :
26 } } 2 3 9 9 5 0");
```

Program Run:

```
Average balance: 4000
Expected: 4000
Average area: 239950
Expected: 239950
```

Comparing Interfaces and Inheritance

- Here is a different interface: `Named`

```
public interface Named
{
   String getName ();
}
```

- A class can implement more than one interface:

```
public class Country implements Measurable, Named
```

- A class can only extend (inherit from) a single superclass.
- An interface specifies the behavior that an implementing class should supply
  Java 8, an interface can now supply a *default* implementation).
- A superclass provides some implementation that a subclass inherits.
- Develop interfaces when you have code that processes objects of different classes in a common way.

(in

Self Check 10.1

Suppose you want to use the `average` method to find the average salary of an array of `Employee` objects. What condition must the `Employee` class fulfill?

Answer: It must implement the `Measurable` interface, and its `getMeasure` method must return the salary.

Self Check 10.2

Why can't the `average` method have a parameter variable of type `object []` ?

Answer: The `object` class doesn't have a `getMeasure` method, and the `average` method invokes the `getMeasure` method.

Self Check 10.3

Why can't you use the `average` method to find the average length of `string` objects?

Answer: You cannot modify the `string` class to implement `Measurable`—`string` is a library class.

Self Check 10.4

What is wrong with this code?

```
Measurable meas = new Measurable();
System.out.printin(meas.getMeasure());
```

Answer: `Measurable` is not a class. You cannot construct objects of type `Measurable`.

Self Check 10.5

What is wrong with this code?

```
Measurable meas = new Country("Uruguay", 176220);
System.out.printin(meas.getName());
```

Answer: The variable `meas` is of type `Measurable`, and that type has no `getName` method.

Converting From Classes to Interfaces

- You can convert from a class type to an interface type, provided the class implements the interface.
- A `Measurable` variable can referto an object of the `BankAccount` class because `BankAccount` implements the `Measurable` interface:

```
BankAccount account = new BankAccount(1000);
Measurable meas = account; // OK
```

- A `Measurable` variable can referto an object of the `Country` class because that class also implements the `Measurable` interface:

```
Country Uruguay = new Country("Uruguay", 176220);
Measurable meas = Uruguay; // Also OK
```

- **A** `Measurable` variable **cannot** refer to an object of the `Rectangle` class because `Rectangle` doesn't implement `Measurable`:

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

Variables of Class and Interface Types

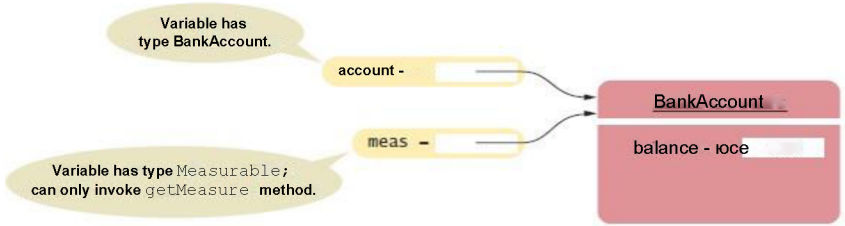


Figure 2 Two references to the same object

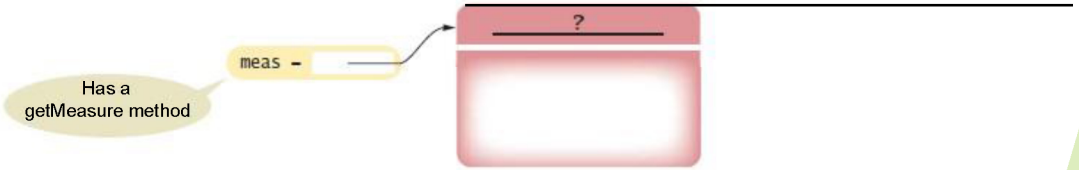


Figure 3 An Interface Reference Can Refer to an Object of Any Class that Implements the Interface

- Method calls on an interface reference are polymorphic. The appropriate method is determined at run time.

Casting from Interfaces to Classes

- Method to return the object with the largest measure:

```
public static Measurable larger(Measurable obj1, Measurable obj2)
{
  if (obj1.getMeasure() > obj2.getMeasure())
  {
    return obj1;
  }
  else
  {
    return obj2 ;
  }
}
```

- Returns the object with the larger measure, as a `Measurable` reference.

```
Country Uruguay = new Country("Uruguay",176220);

Country thailand = new Country("Thailand", 513120);

Measurable max = larger(uruguay, thailand);
```

Casting from Interfaces to Classes

- You know that `max` refers to a `Country` object, but the compiler does not.
- Solution: cast

```
Country maxCountry = (Country) max;
String name = maxCountry.getName();
```

- ► **You need a cast to convert from an interface type to a clas** s
- **type.**
- ► **If you are wrong and** `max` **doesn't refer to a** `Country` **objec** t,
- **the program throws an exception at runtime.**
- ► **If a** `Person` **object is actually a** `Superhero`, **you need a cast before you can apply any** `Superhero` **methods.**

Self Check 10.6

Can you US6 a cast `(BankAccount) meas` to convert a `Measurable` variable `meas` to a `BankAccount` reference?

Answer: Only if `meas` actually refers to a `BankAccount` object.

Self Check 10.7

If both `BankAccount` and `Country` implement the `Measurable` interface, can a `Country` reference be converted to a `BankAccount` reference?

Answer: No — a `Country` reference can be converted to a `Measurable` reference, but if you attempt to cast that reference to a `BankAccount`, an exception occurs.

Self Check 10.8

Why is it impossible to construct a `Measurable` object?

Answer: `Measurable` is an interface. Interfaces have no instance variables and no method implementations.

Self Check 10.9

Why can you nevertheless declare a variable whose type is `Measurable`?

Answer: That variable never refers to a `Measurable` object. It refers to an object of some class—a class that implements the `Measurable` interface.

Self Check 10.10

What does this code fragment print? Why is this an example of polymorphism?

```
Measurable[] data = { new BankAccount(10000), new Country("Belgium", 30510) };
System.out.println(average(data));
```

- **Answer: The code fragment prints 20255. The average metř calls `getMeasure` on each object in the array. In the first V call, the object is a `BankAccount`. In the second call, the object is a `Country`. A different `getMeasure` method is called in each case. The first call returns the account balance, the second one the area, which are then averaged.**

The Comparable Interface

- `Comparable` interface is in the standard Java library.
- `Comparable` interface has a single method:

```
public interface Comparable
{
   int compareTo(Object otherObject);
J__
```

- The call to the method:

```
a.compareTo(b)
```

- The `compareTo` method returns:

  a negative number if a should come before b,

  zero if a and b are the same

  a positive number if b should come before a.

- Implement the `Comparable` interface so that objects of your class can be compared, for example, in a sort method.

The Comparable Interface

- `BankAccount` class' implementation of `Comparable`:

```
public class BankAccount implements Comparable
{
   . . .
   public int compareTo(Object otherObject)
   {
      BankAccount other = (BankAccount) otherObject;
      if (balance < other.balance) { return -1; }
      if (balance > other.balance) { return 1; }
      return 0;
   }
   . . .
}
```

- `compareTo` method has a parameter of reference type `ob j ect`
- To get a `BankAccount` reference:

```
BankAccount other = (BankAccount) otherObject;
```

The Comparable Interface

- Because the `BankAccount` class implements the `Comparable` interface, you can sort an array of bank accounts with the `Arrays . sort` method:

```
BankAccount[] accounts = new BankAccount [3];
accounts[0] = new BankAccount(10000);
accounts[1] = new BankAccount(0);
accounts[2] = new BankAccount(2000);
Arrays.sort(accounts);
```

- Now the `accounts` array is sorted by increasing balance.
- The `compareTo` method checks whether another object is larger or smaller.

Self Check 10.11

How can you sort an array of `Country` objects by increasing area?

Answer: Have the `Country` class implement the `Comparable` interface, as shown below, and call `Arrays .sort`.

```
public class Country implements Comparable
{
   . . .
   public int compareTo(Object otherObject)
   {
      Country other = (Country) otherObject;
      if (area < other.area) {
       return -1;
      }
      if (area > other.area) {
       return 1;
      }
      return 0 ;
   }
}
```

Self Check 10.12

Can you use the `Arrays . sort` method to sort an array of `string` objects? Check the API documentation for the `string` class.

Answer: Yes, you can, because `string` implements the `Comparable` interface type.

Self Check 10.13

Can you use the `Arrays.sort` method to sort an array of `Rectangle` objects? Check the API documentation for the `Rectangle` class.

Answer: No. The `Rectangle` class does not implement the `Comparable` interface.

Self Check 10.14

Write a method max that finds the larger of any two `Comparable` objects.

Answer:

```
public static Comparable max(Comparable a, Comparable b)
{
   if (a.compareTo(b) >0) { return a; }
   else { return b; }
}
```

Self Check 10.15

Write a call to the method of Self Check 14 that computes the larger of two bank accounts, then prints its balance.

Answer:

```
BankAccount larger = (BankAccount) max(first, second);
System.out.println(larger.getBalance());
```

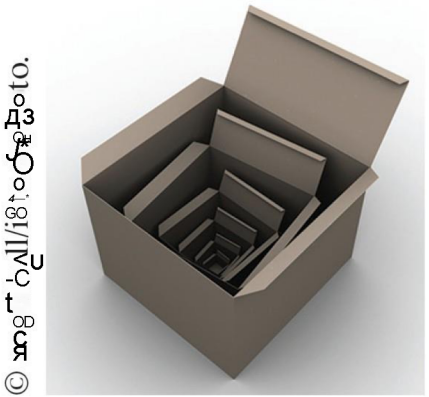
Note that the result must be cast from `Comparable` to `BankAccount` so that you can invoke the `getBalance` method.

Inner Classes

- Trivial class can be declared inside a method:

```
public class MeasurerTester
{
   public static void main(String [] args)
   {
      class AreaMeasurer implements Measurer
      {
         . . .
      }
      . . .
      Measurer areaMeas = new AreaMeasurer();
      double averageArea = Data.average(rects, areaMeas);
      . . .
   }
}
```

- An inner class is a class that is declared inside another class.



Inner Classes

- You can declare inner class inside an enclosing class, but outside its methods
- It is available to all methods of enclosing class:
- Compiler turns an inner class into a regular class file with a strange name:
- Inner classes are commonly used for utility classes that should not be visible elsewhere in a program.

```
oublie class MeasurerTester
{
class AreaMeasurer implements Measurer
{
  . . .
}
c'Ublic static void main (Strina_[ ] args)
{
  Measurer areaMeas = new AreaMeasurer();
  double averageArea = Data.average(rests, areaMeas);
  . . .
}
}
```

Self Check 10.21

Why would you use an inner class instead of a regular class?

Answer: Inner classes are convenient for insignificant classes. Also, their methods can access local and instance variables from the surrounding scop e.

Self Check 10.22

When would you place an inner class inside a class but outside any methods?

Answer: When the inner class is needed by more than one method of the classes.

Self Check 10.23

How many class files are produced when you compile the `MeasurerTester` program from this section?

Answer: Four: one for the outer class, one for the inner class, and two for the `Data` and `Measurer` classes.

Mock Objects

- Problem: Want to test a class before the entire program has been complet ed.
- **A mock object** provides the same services as another object, but in a simpl ified manner.
- If you just want to practice arranging the Christmas decorations, you don't ne ed a real tree. Similarly, when you develop a computer program, you can use mock objects to test parts of your program.

■L' Don Nirfiob/iSloctyholo.

Mock Objects

- **Example:** a grade book application, `GradingProgram`, manages quiz scores using class `GradeBook` with methods:

```
public void addScore(int studentId, double score)
public double getAverageScore(int studentId)
public void save(String filename)
```

- **Want to test `GradingProgram` without having a fully functional `GradeBook` class.**
- **Declare an interface type with the same methods that the `GradeBook` class provides**

  Convention: use the letter `I` as a prefix for the interface name

```
public interface IGradeBook
{
   void addScore(int studentId, double score);
   double getAverageScore(int studentId);
   void save(String filename);
   . . .
}
```

Mock Objects

- Both the mock class and the actual class implement the same interface. The `GradingProgram` Class Should ОП/y USe this interface, never the `GradeBook` class which implements this interface.
- Meanwhile, provide a simplified mock implementation, restricted to the case of one student and without saving functionality:

```
public class MockGradeBook implements IGradeBook
{
  private ArrayList<Double> scores;

  public MockGradeBook() { scores = new ArrayList<Double>(); }

  public void addScore(int studentId, double score)
  {
    // Ignore studentId scores.add(score);
  }

  public double getAverageScore(int studentId)
  {
    double total = 0;
    for (double x : scores) { total = total + x; } return total / scores.size ();
  }

  public void save(String filename)
  {
    // Do nothing
  }
  . . .
}
```

Mock Objects

- Now construct an instance of `MockGradeBook` and use it immediately to test the `GradingProgram` class.
- When you are ready to test the actual class, simply use a `GradeBook` instance instead.
- Don't erase the mock class — it will still come in handy for regression testing.

Self Check 10.24

Why is it necessary that the real class and the mock class implement the same interface type ?

Answer: You want to implement the `GradingProgram` class in terms of t hat interface so that it doesn't have to change when you switch between the mo ck class and the actual class.

Self Check 10.25

Why is the technique `Of` mock objects particularly effective when the `GradeBook` and `GradingProgram` class are developed by two programmers?

Answer: Because the developer of `GradingProgram` doesn't have to wait for the `GradeBook` class to be complete.