Interface Types

- An interface is a special type of declaration that lists a set of methods and their signatures
  - A class that '*implements*' the interface must implement all of the methods of the interface
  - It is similar to a class, but there are differences:
    - All methods in an interface type are abstract:

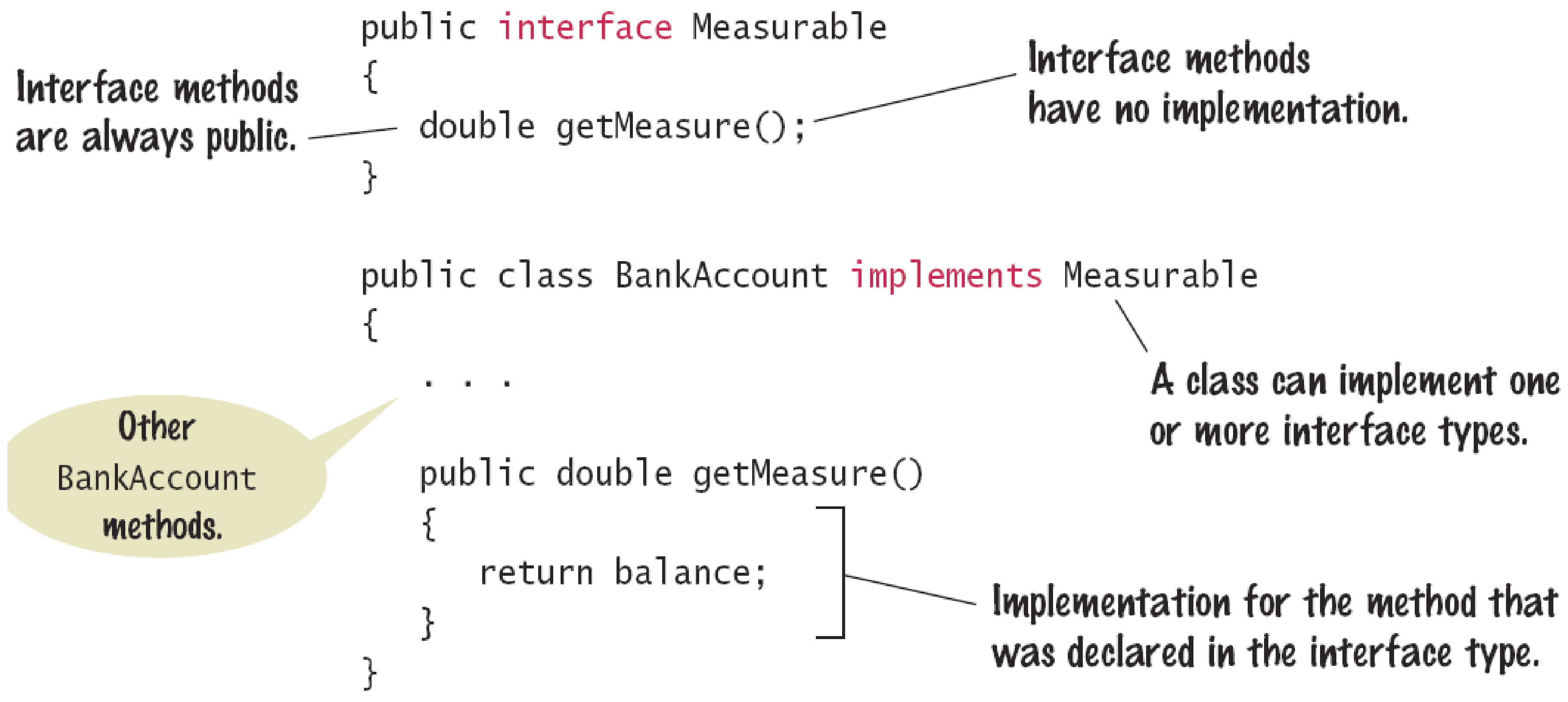
      They have a name, parameters, and a return type, but they don't have an implementation
    - All methods in an interface type are automatically public
    - An interface type cannot have instance variables
    - An interface type cannot have static methods

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

A Java `interface` type declares a set of methods and their signatures.

Interface Types

- An interface declaration and a class that `implements` the `interface`.



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

public class BankAccount implements Measurable
{
   . . .

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

Interface methods are always public.
Interface methods have no implementation.
A class can implement one or more interface types.
Other BankAccount methods.
Implementation for the method that was declared in the interface type.

Using Interface Types

- We can use the interface type `Measurable` to implement a "universal" static method for computing averages:

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

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

Implementing an Interface

- A class can be declared to `implement` an interface
  - The class must implement all methods of the interface

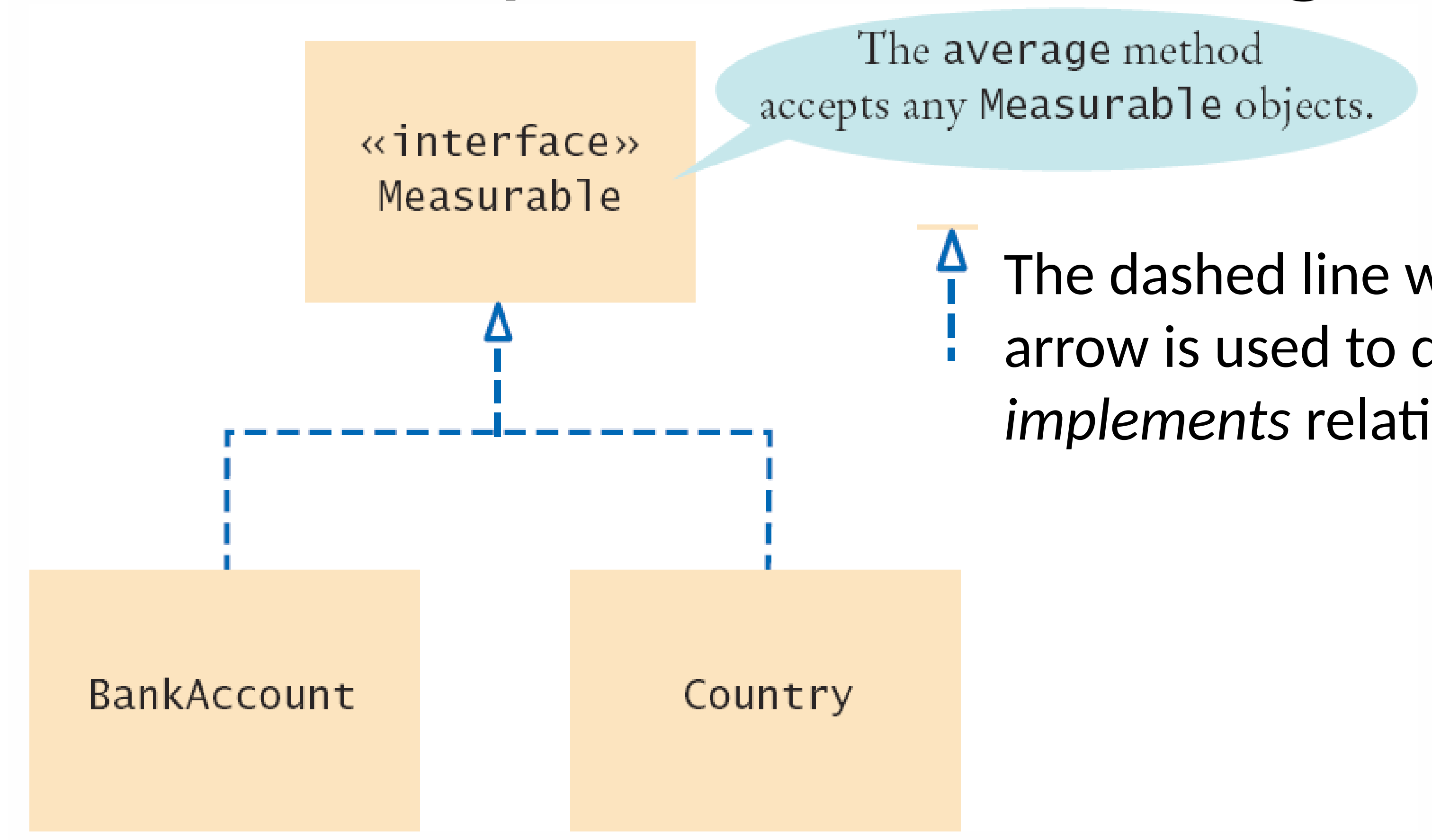
```
public class BankAccount implements Measurable
{
   public double getMeasure()
   {
      return balance;
   }
   . . .
}
```

Use the `implements` reserved word in the class declaration.

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

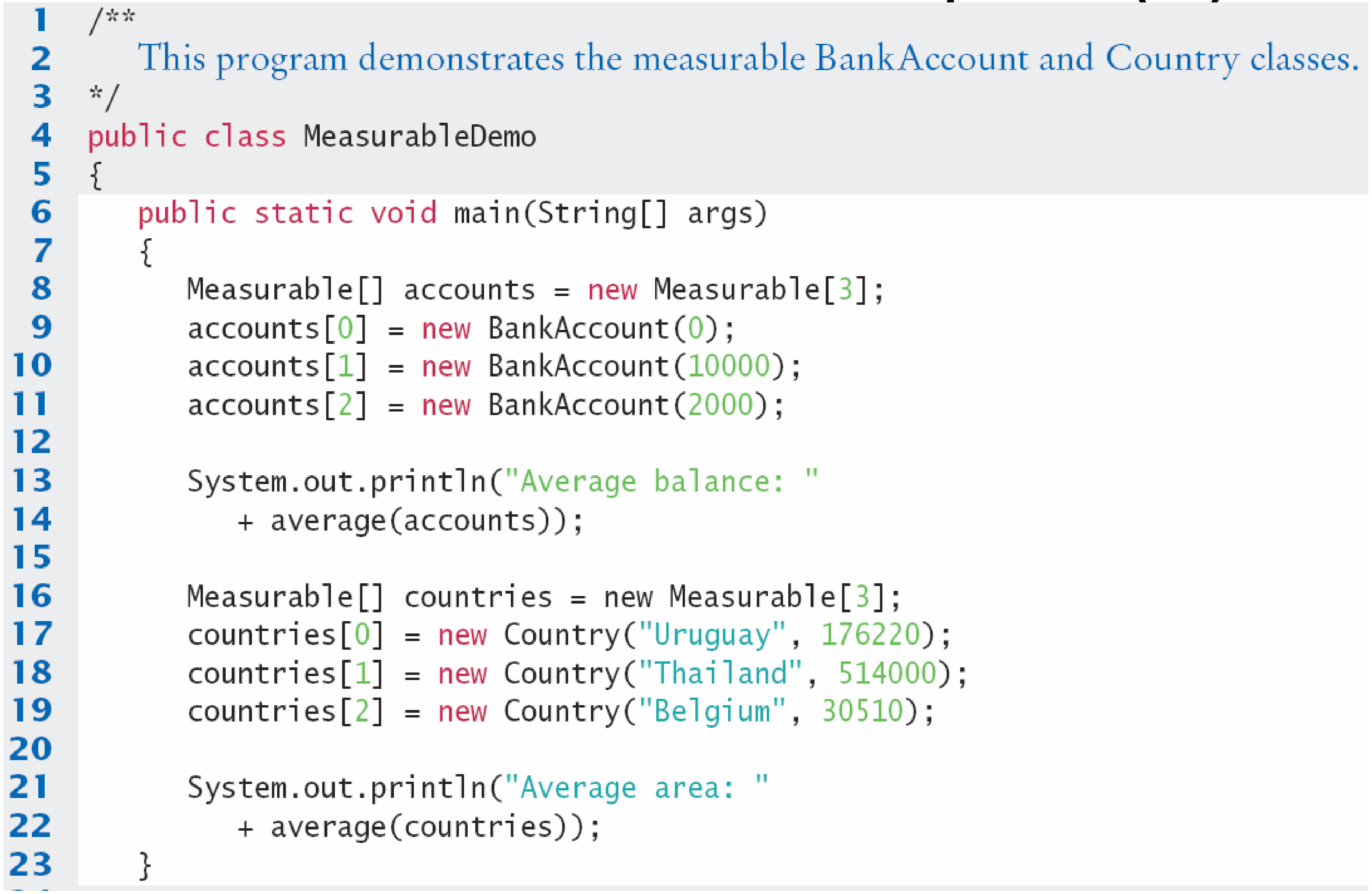
The methods of the interface must be declared as `public`

An Implementation Diagram



The dashed line with an arrow is used to denote *implements* relationships

MeasureableDemo.java (1)



```java
/**
   This program demonstrates the measurable BankAccount and Country classes.
*/
public class MeasurableDemo
{
   public static void main(String[] args)
   {
      Measurable[] accounts = new Measurable[3];
      accounts[0] = new BankAccount(0);
      accounts[1] = new BankAccount(10000);
      accounts[2] = new BankAccount(2000);

      System.out.println("Average balance: "
         + average(accounts));

      Measurable[] countries = new Measurable[3];
      countries[0] = new Country("Uruguay", 176220);
      countries[1] = new Country("Thailand", 514000);
      countries[2] = new Country("Belgium", 30510);

      System.out.println("Average area: "
         + average(countries));
   }
```

MeasureableDemo.java (2)

```java
    /**
       Computes the average of the measures of the given objects.
       @param objs an array of Measurable objects
       @return the average of the measures
    */
    public static double average(Measurable[] objs)
    {
       if (objs.length == 0) { return 0; }
       double sum = 0;
       for (Measurable obj : objs)
       {
          sum = sum + obj.getMeasure();
       }
       return sum / objs.length;
    }
}
```

Program Run

```
Average balance: 4000.0
Average area: 240243.33333333334
```

The `Comparable` Interface

- The Java library includes a number of important interfaces including Comparable
  - It requires implementing one method: `compareTo()`
  - It is used to compare two objects
  - It is implemented by many objects in the Java API
  - If may want to implement it in your classes to use powerful Java API tools such as sorting
- It is called on one object, and is passed another
  - Called on object a, return values include:
    - Negative: a comes before b
    - Positive: a comes after b
    - 0: a is the same as b

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

The `Comparable` Type parameter

- The `Comparable` interface uses a special type of parameter that allows it to work with any type:

```
public interface Comparable<T>
{
  int compareTo(T other);
}
```

  - The type <T> is a placeholder for an actual type of object
  - The class ArrayList class uses the same technique with the type surrounded by angle brackets < >

```
ArrayList<String> names = new
   ArrayList<String>();
```

Using the type inside angle braces will be covered further in the next chapter.

A `Comparable` Example

- The BankAccount `compareTo` method compares bank accounts by their balance.
  - It takes one parameter of it’s own class type (BankAccount)

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

The methods of the interface must be declared as public

Using compareTo to Sort

- The `Arrays.sort` method uses the `compareTo` method to sort the elements of the array
  - Once 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);
```

  - The array is now sorted by increasing balance

Implementing Java Library interfaces allows you to use the power of the Java Library with your classes.

Common Error

- Forgetting to Declare Implementing Methods as Public
  - The methods in an interface are not declared as public, because they are public by default.
  - However, the methods in a class are not public by default.
  - It is a common error to forget the public reserved word when declaring a method from an interface:

```
public class BankAccount implements Measurable
{
   double getMeasure()   // Oops—should be public
   {
      return balance;
   }
   . . .
}
```

Special Topic

- Interface Constants
  - Interfaces cannot have instance variables, but it is legal to specify constants
  - When declaring a constant in an interface, you can (and should) omit the reserved words `public static final`, because all variables in an interface are automatically `public static final`.

```
public interface SwingConstants
{
   int NORTH = 1;
   int NORTHEAST = 2;
   int EAST = 3;
   . . .
}
```

Interface vs. Abstract Class

- How does an Interface differ from an abstract class since both contain unimplemented and therefore abstract methods? The differences are significant:
  - An interface cannot implement any methods, whereas an abstract class can.
  - A class can implement many interfaces but can have only one superclass.
  - An interface is not part of the class hierarchy. Unrelated classes can implement the same interface.

Extending Interface

```
[public] interface InterfaceName [extends SuperInterfaces]
{
   // constants (optional)
   // method declarations without implementations
}
```

- Inheritance can also be applied to interfaces
- Define one interface based on another by using the keyword **`extends`** to identify the base interface name.
- A class can extend only one other class, an interface can extend any number of interfaces. The list of superi nterfaces is a comma-separated list of all the interfaces extended by the new interface.

Extending Interface (cont)

```
public interface Transportation{
    double travelTime();
    }
```

```
public interface Vehicle extends Transportation{
    int speedUp();
    int slowDown();
    }
```

- Any class implementing `Vehicle` will have access to the `Transportation` interface
- A class implementing the `Vehicle` interface should also implement `Transportation` interface

Extending Interface

```
public interface Truck extends Vehicle, Container{
    void load();
    void unload();
    }
```

- Unlike a class, which can extend only one other class, an interface can extend any number of other interfaces. This is called “***multiple inheritance***”.
- The `Truck` interface inherits all the methods and constants that are members of `Container` and `Vehicle` interfaces.
- A class implementing `Truck` interface should also implement `Container` and `Vehicle` interfaces.