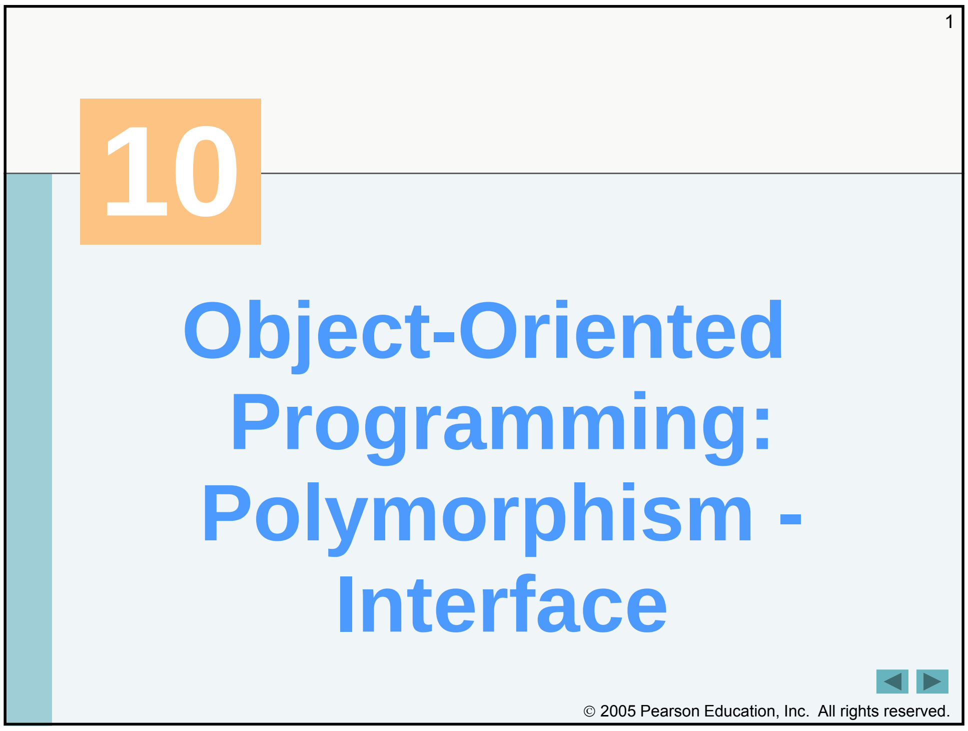
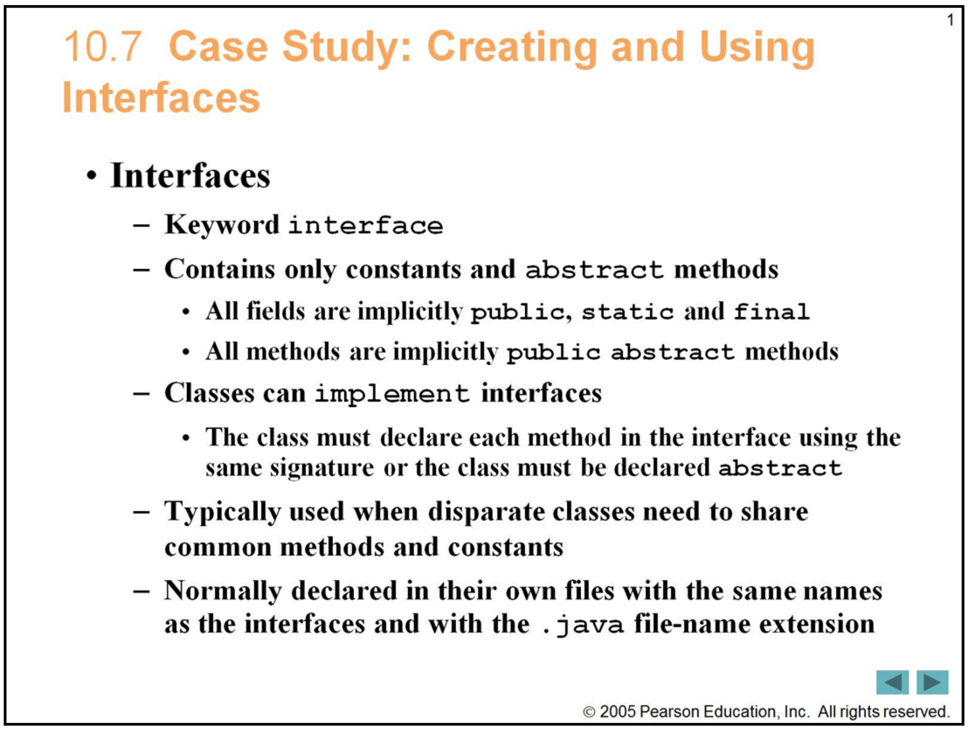
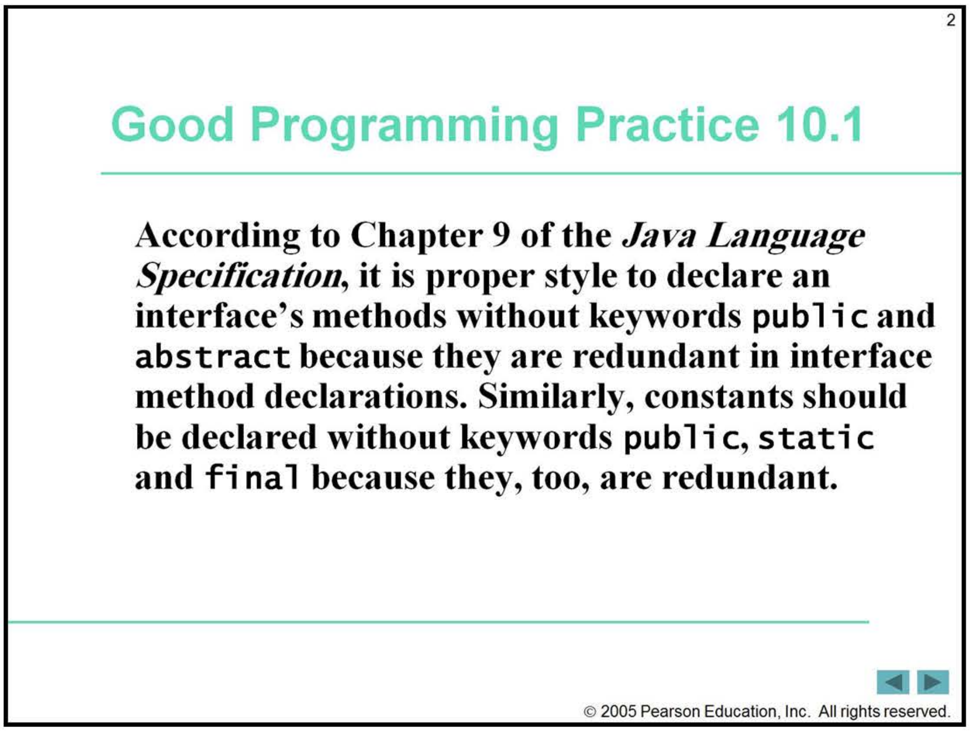


10 Object-Oriented Programming: Polymorphism - Interface



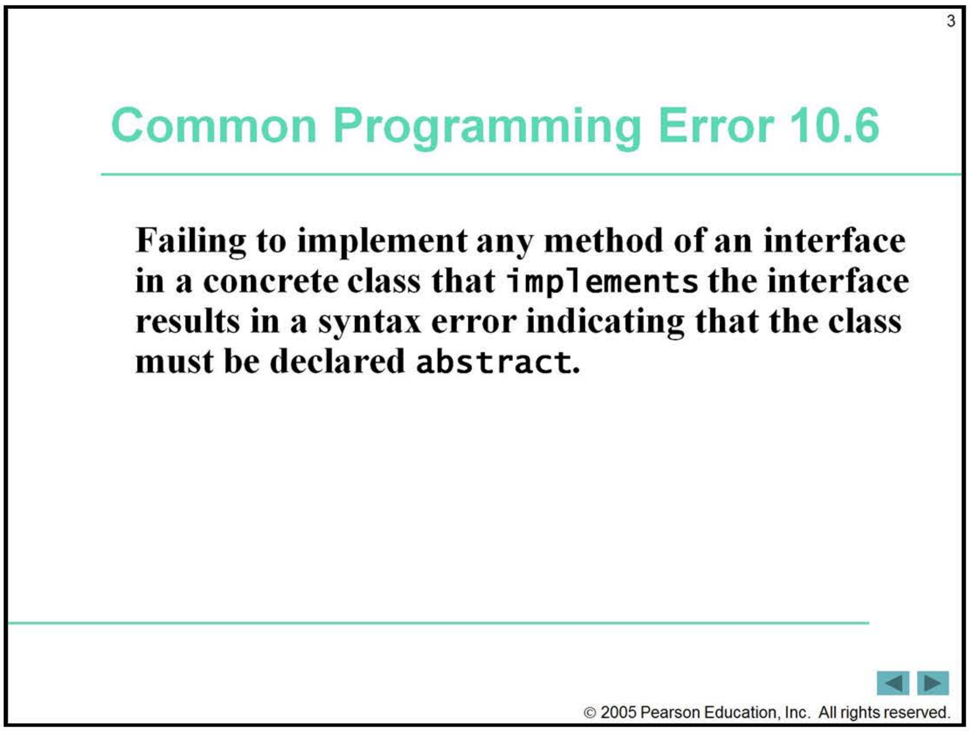
10.7 Case Study: Creating and Using Interfaces

- **Interfaces**
  - Keyword `interface`
  - Contains only constants and `abstract` methods
    - All fields are implicitly `public`, `static` and `final`
    - All methods are implicitly `public abstract` methods
  - Classes can `implement` interfaces
    - The class must declare each method in the interface using the same signature or the class must be declared `abstract`
  - Typically used when disparate classes need to share common methods and constants
  - Normally declared in their own files with the same names as the interfaces and with the `.java` file-name extension



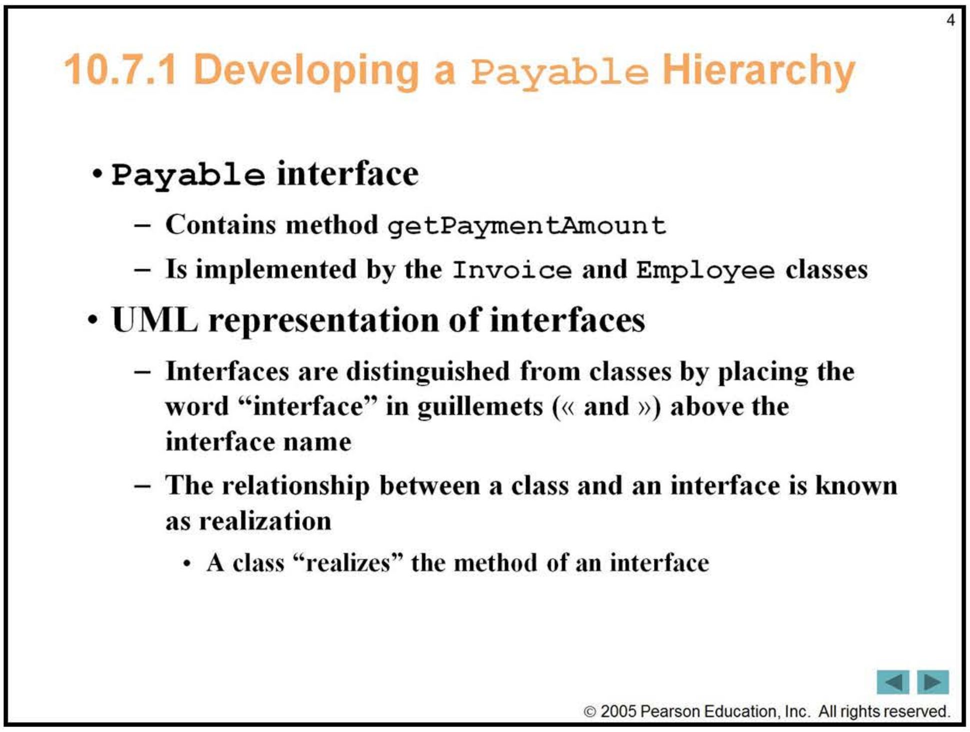
Good Programming Practice 10.1

According to Chapter 9 of the *Java Language Specification*, it is proper style to declare an interface's methods without keywords `public` and `abstract` because they are redundant in interface method declarations. Similarly, constants should be declared without keywords `public`, `static` and `final` because they, too, are redundant.



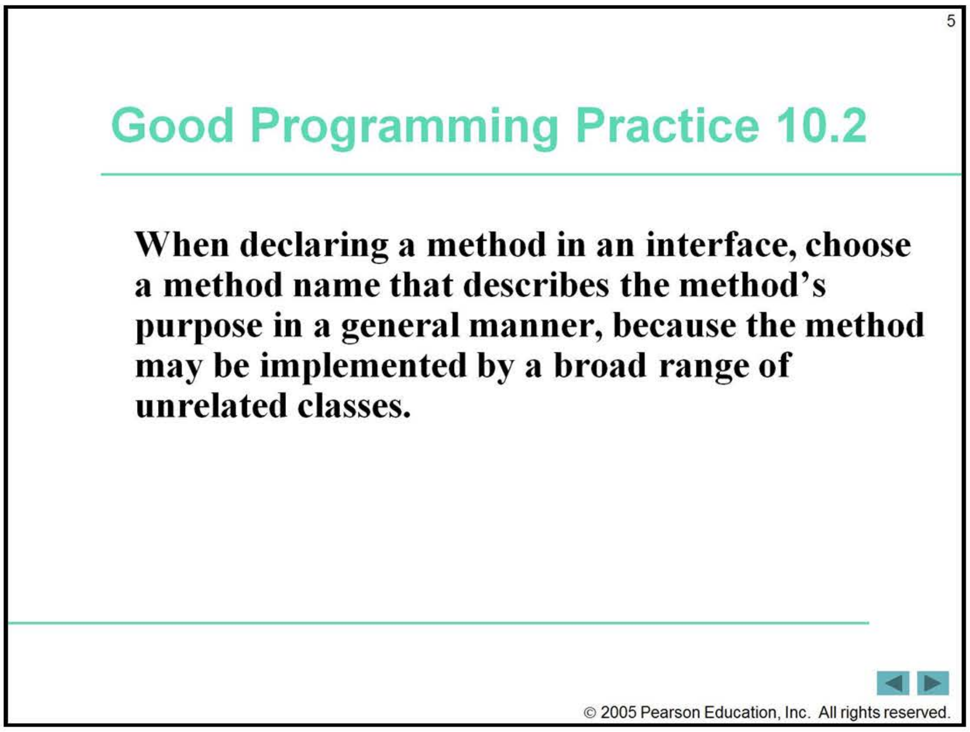
Common Programming Error 10.6

Failing to implement any method of an interface in a concrete class that `implements` the interface results in a syntax error indicating that the class must be declared `abstract`.



10.7.1 Developing a `Payable` Hierarchy

- `Payable` interface
  - Contains method `getPaymentAmount`
  - Is implemented by the `Invoice` and `Employee` classes
- UML representation of interfaces
  - Interfaces are distinguished from classes by placing the word “interface” in guillemets (« and ») above the interface name
  - The relationship between a class and an interface is known as realization
    - A class “realizes” the method of an interface



Good Programming Practice 10.2

When declaring a method in an interface, choose a method name that describes the method’s purpose in a general manner, because the method may be implemented by a broad range of unrelated classes.

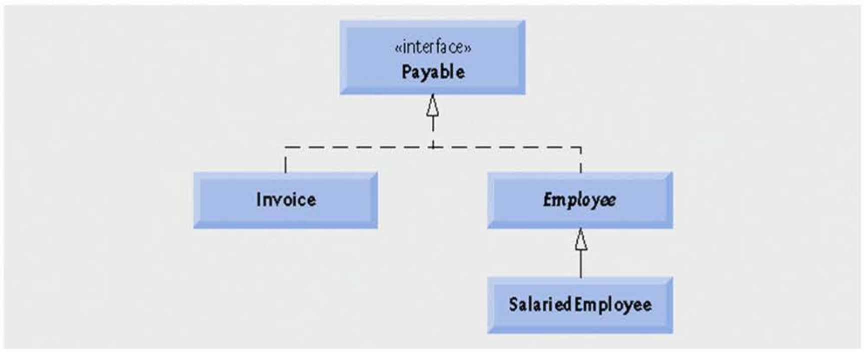


Fig. 10.10 | `Payable` **interface hierarchy UML class diagram.**

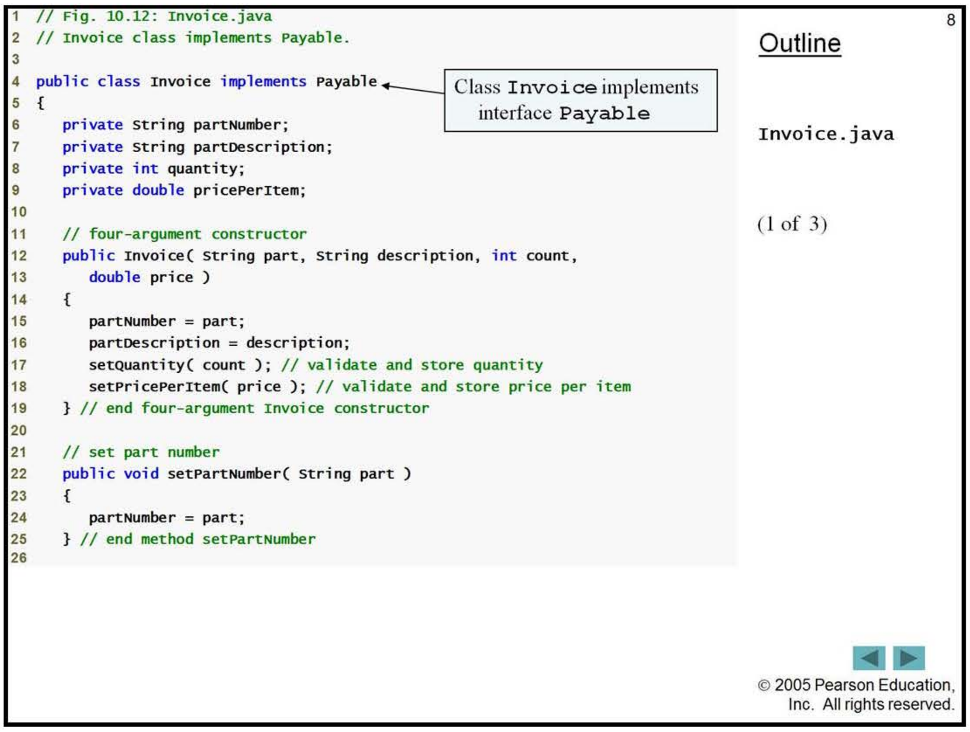
Outline

`Payable.java`

```java
// Fig. 10.11: Payable.java
// Payable interface declaration.

public interface Payable
{
   double getPaymentAmount(); // calculate payment; no implementation
} // end interface Payable
```

Declare interface `Payable`

Declare `getPaymentAmount` method which is implicitly `public` and `abstract`



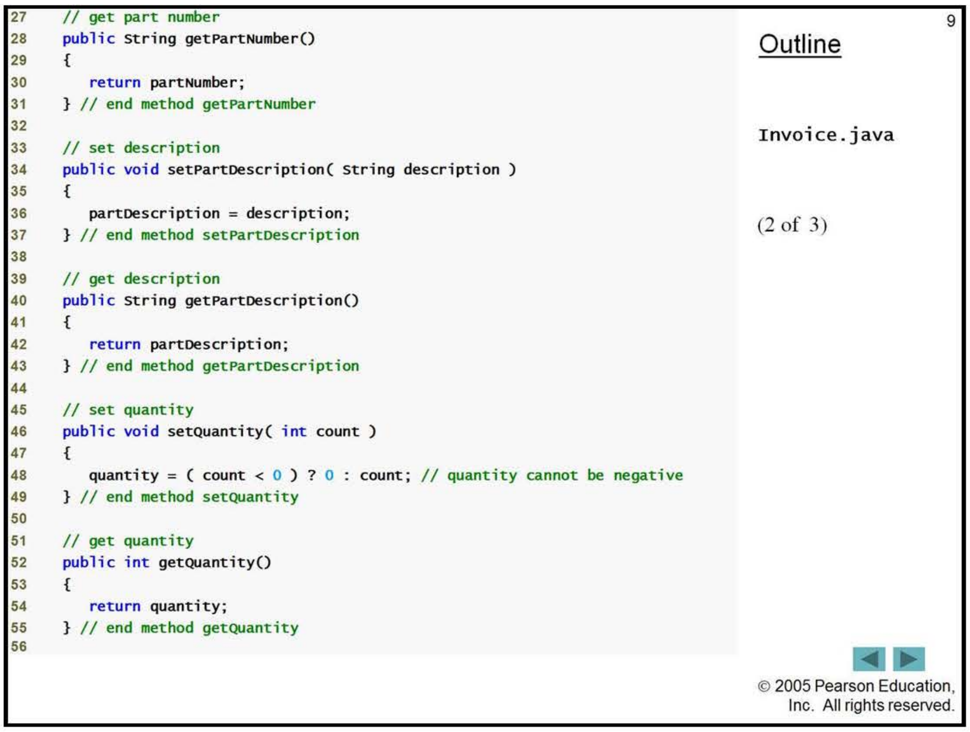
Outline

Invoice.java

(1 of 3)

```java
// Fig. 10.12: Invoice.java
// Invoice class implements Payable.

public class Invoice implements Payable
{
   private String partNumber;
   private String partDescription;
   private int quantity;
   private double pricePerItem;

   // four-argument constructor
   public Invoice( String part, String description, int count,
      double price )
   {
      partNumber = part;
      partDescription = description;
      setQuantity( count ); // validate and store quantity
      setPricePerItem( price ); // validate and store price per item
   } // end four-argument Invoice constructor

   // set part number
   public void setPartNumber( String part )
   {
      partNumber = part;
   } // end method setPartNumber

```

Class `Invoice` implements interface `Payable`

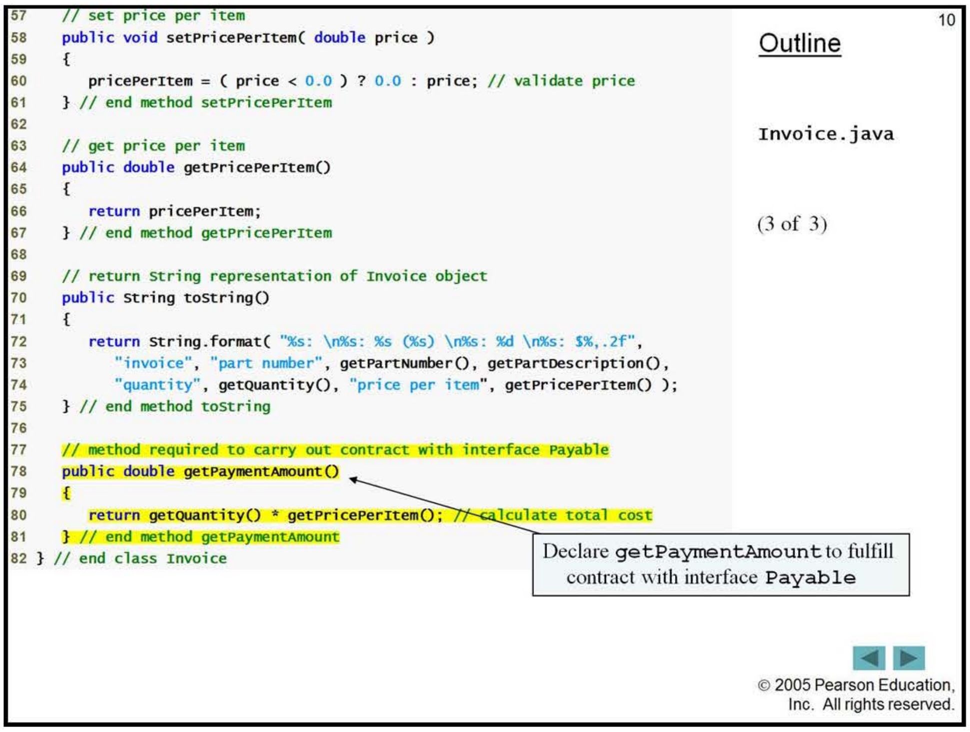


Outline

Invoice.java

(2 of 3)

```java
   // get part number
   public String getPartNumber()
   {
      return partNumber;
   } // end method getPartNumber

   // set description
   public void setPartDescription( String description )
   {
      partDescription = description;
   } // end method setPartDescription

   // get description
   public String getPartDescription()
   {
      return partDescription;
   } // end method getPartDescription

   // set quantity
   public void setQuantity( int count )
   {
      quantity = ( count < 0 ) ? 0 : count; // quantity cannot be negative
   } // end method setQuantity

   // get quantity
   public int getQuantity()
   {
      return quantity;
   } // end method getQuantity

```



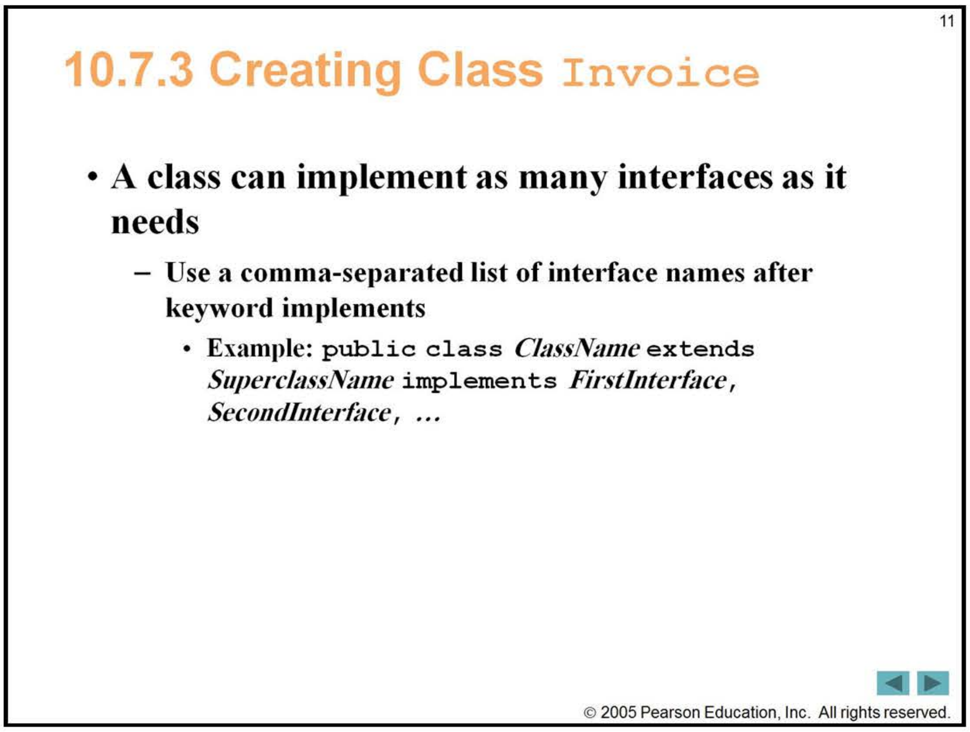
```
   // set price per item
   public void setPricePerItem( double price )
   {
      pricePerItem = ( price < 0.0 ) ? 0.0 : price; // validate price
   } // end method setPricePerItem

   // get price per item
   public double getPricePerItem()
   {
      return pricePerItem;
   } // end method getPricePerItem

   // return String representation of Invoice object
   public String toString()
   {
      return String.format( "%s: \n%s: %s (%s) \n%s: %d \n%s: $%,.2f",
         "invoice", "part number", getPartNumber(), getPartDescription(),
         "quantity", getQuantity(), "price per item", getPricePerItem() );
   } // end method toString

   // method required to carry out contract with interface Payable
   public double getPaymentAmount()
   {
      return getQuantity() * getPricePerItem(); // calculate total cost
   } // end method getPaymentAmount
} // end class Invoice
```

Outline

Invoice.java

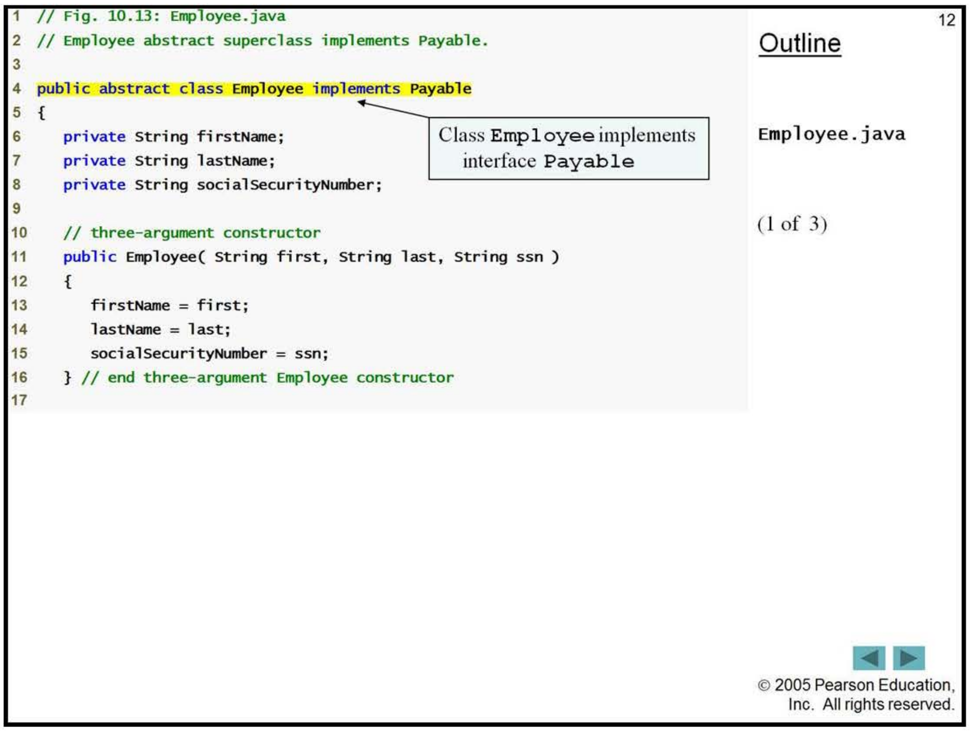
(3 of 3)

Declare getPaymentAmount to fulfill contract with interface Payable



10.7.3 Creating Class Invoice

- A class can implement as many interfaces as it needs
  - Use a comma-separated list of interface names after keyword implements
    - Example: public class *ClassName* extends *SuperclassName* implements *FirstInterface*, *SecondInterface*, …



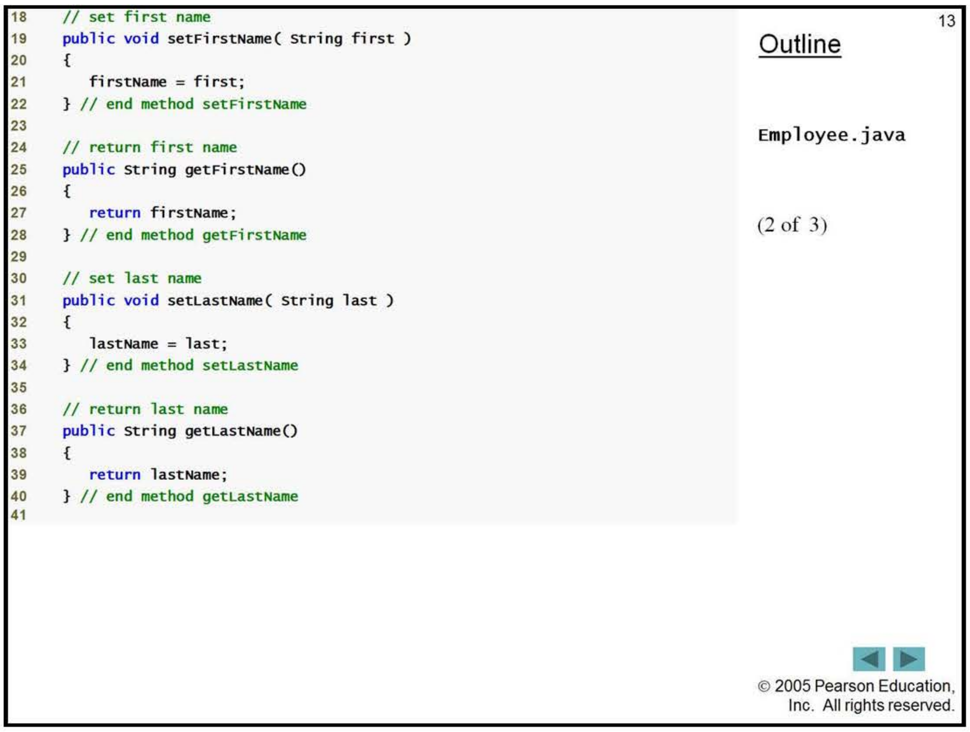
Outline

```java
// Fig. 10.13: Employee.java
// Employee abstract superclass implements Payable.

public abstract class Employee implements Payable
{
   private String firstName;
   private String lastName;
   private String socialSecurityNumber;

   // three-argument constructor
   public Employee( String first, String last, String ssn )
   {
      firstName = first;
      lastName = last;
      socialSecurityNumber = ssn;
   } // end three-argument Employee constructor

```

Class `Employee` implements interface `Payable`

`Employee.java`

(1 of 3)

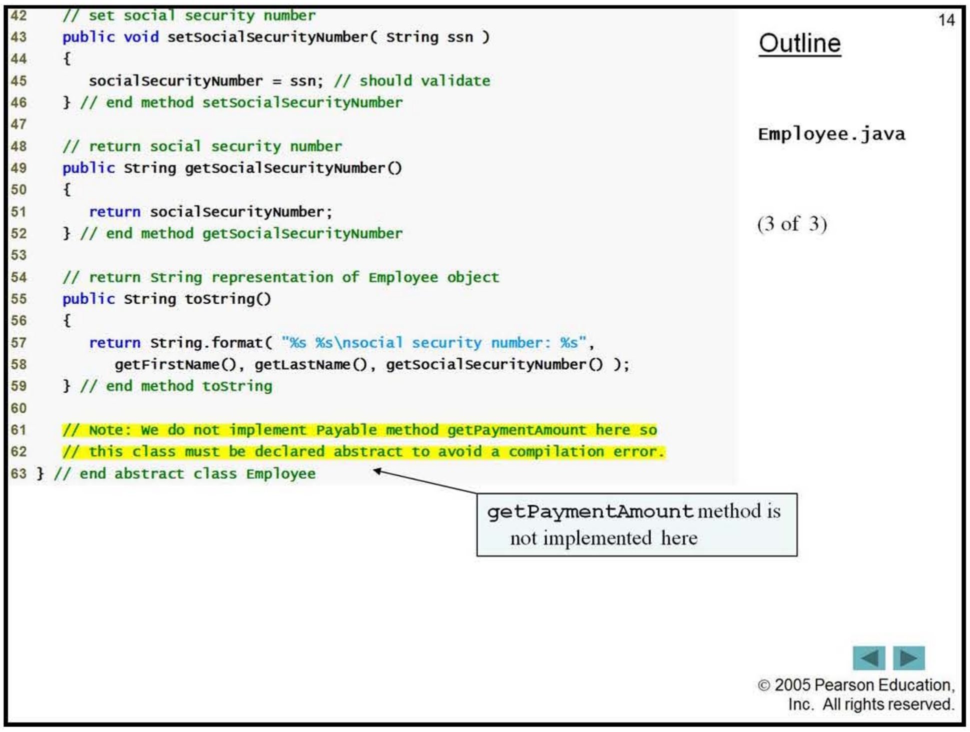


Outline

```java
   // set first name
   public void setFirstName( String first )
   {
      firstName = first;
   } // end method setFirstName

   // return first name
   public String getFirstName()
   {
      return firstName;
   } // end method getFirstName

   // set last name
   public void setLastName( String last )
   {
      lastName = last;
   } // end method setLastName

   // return last name
   public String getLastName()
   {
      return lastName;
   } // end method getLastName

```

`Employee.java`

(2 of 3)



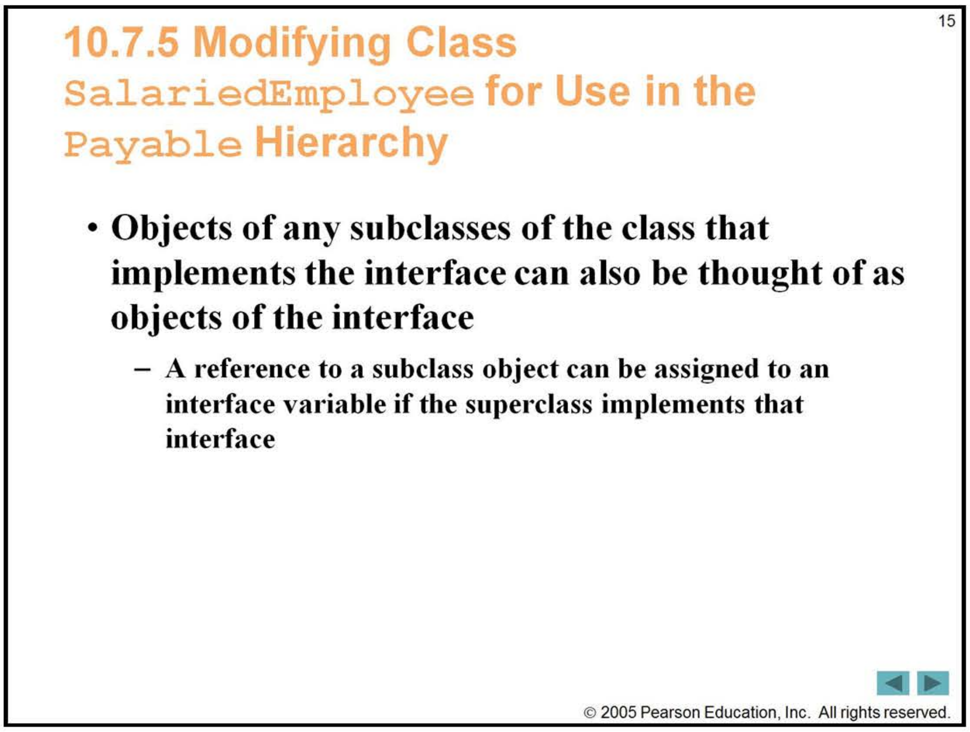
Outline

```
   // set social security number
   public void setSocialSecurityNumber( String ssn )
   {
      socialSecurityNumber = ssn; // should validate
   } // end method setSocialSecurityNumber

   // return social security number
   public String getSocialSecurityNumber()
   {
      return socialSecurityNumber;
   } // end method getSocialSecurityNumber

   // return String representation of Employee object
   public String toString()
   {
      return String.format( "%s %s\nsocial security number: %s",
         getFirstName(), getLastName(), getSocialSecurityNumber() );
   } // end method toString

   // Note: We do not implement Payable method getPaymentAmount here so
   // this class must be declared abstract to avoid a compilation error.
} // end abstract class Employee
```

`Employee.java`

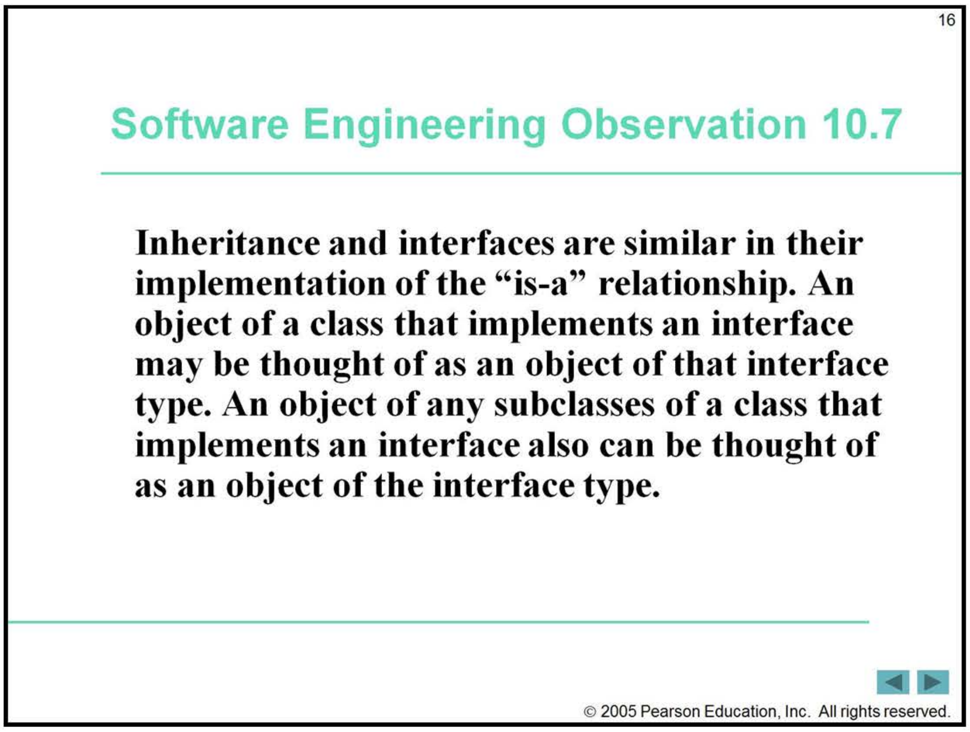
(3 of 3)

`getPaymentAmount` method is not implemented here



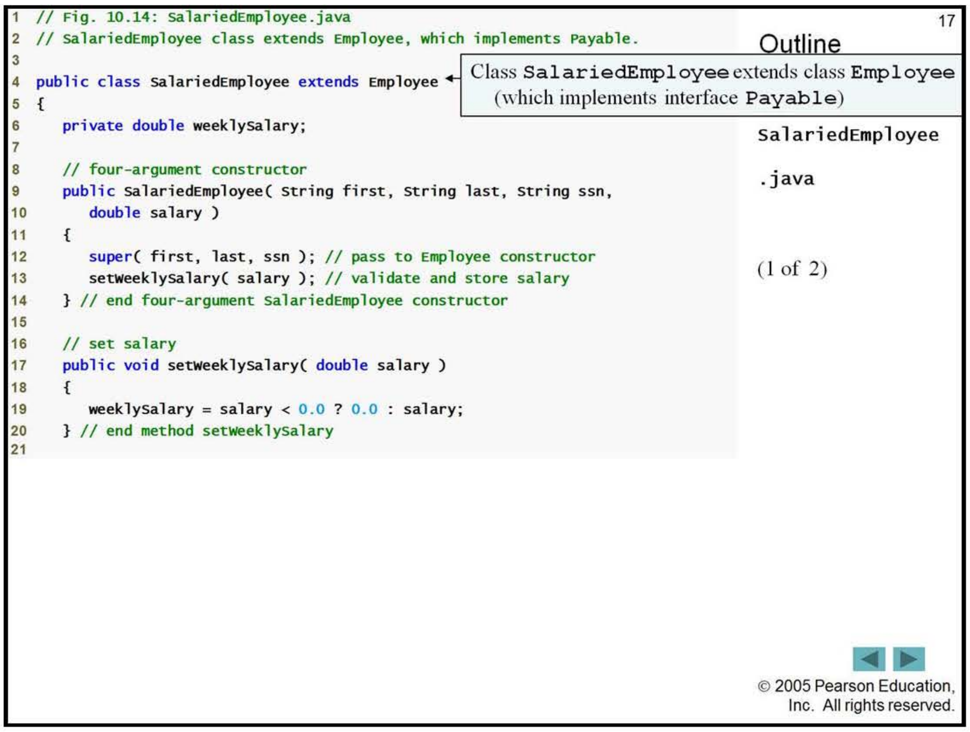
10.7.5 Modifying Class `SalariedEmployee` for Use in the `Payable` Hierarchy

- **Objects of any subclasses of the class that implements the interface can also be thought of as objects of the interface**
  - **A reference to a subclass object can be assigned to an interface variable if the superclass implements that interface**



Software Engineering Observation 10.7

Inheritance and interfaces are similar in their implementation of the "is-a" relationship. An object of a class that implements an interface may be thought of as an object of that interface type. An object of any subclasses of a class that implements an interface also can be thought of as an object of the interface type.



```java
// Fig. 10.14: SalariedEmployee.java
// SalariedEmployee class extends Employee, which implements Payable.

public class SalariedEmployee extends Employee
{
   private double weeklySalary;

   // four-argument constructor
   public SalariedEmployee( String first, String last, String ssn,
      double salary )
   {
      super( first, last, ssn ); // pass to Employee constructor
      setWeeklySalary( salary ); // validate and store salary
   } // end four-argument SalariedEmployee constructor

   // set salary
   public void setWeeklySalary( double salary )
   {
      weeklySalary = salary < 0.0 ? 0.0 : salary;
   } // end method setWeeklySalary

```

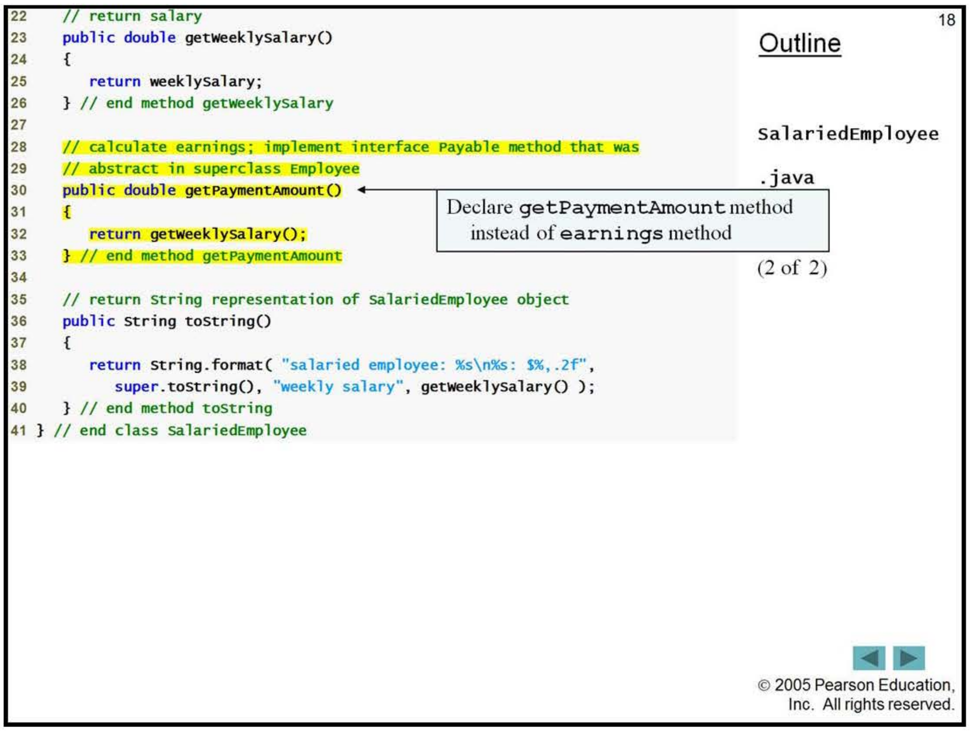
Outline

Class `SalariedEmployee` extends class `Employee` (which implements interface `Payable`)

`SalariedEmployee`

`.java`

(1 of 2)



Outline

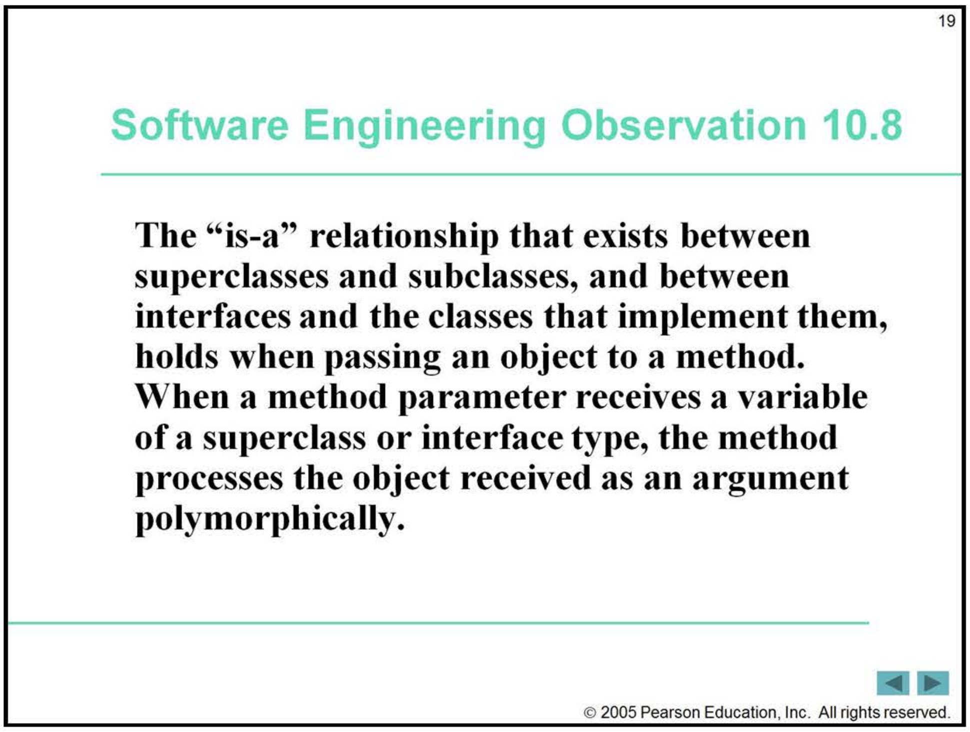
```
   // return salary
   public double getWeeklySalary()
   {
      return weeklySalary;
   } // end method getWeeklySalary

   // calculate earnings; implement interface Payable method that was
   // abstract in superclass Employee
   public double getPaymentAmount()
   {
      return getWeeklySalary();
   } // end method getPaymentAmount

   // return String representation of SalariedEmployee object
   public String toString()
   {
      return String.format( "salaried employee: %s\n%s: $%,.2f",
         super.toString(), "weekly salary", getWeeklySalary() );
   } // end method toString
} // end class SalariedEmployee
```

Declare `getPaymentAmount` method instead of `earnings` method

`SalariedEmployee`

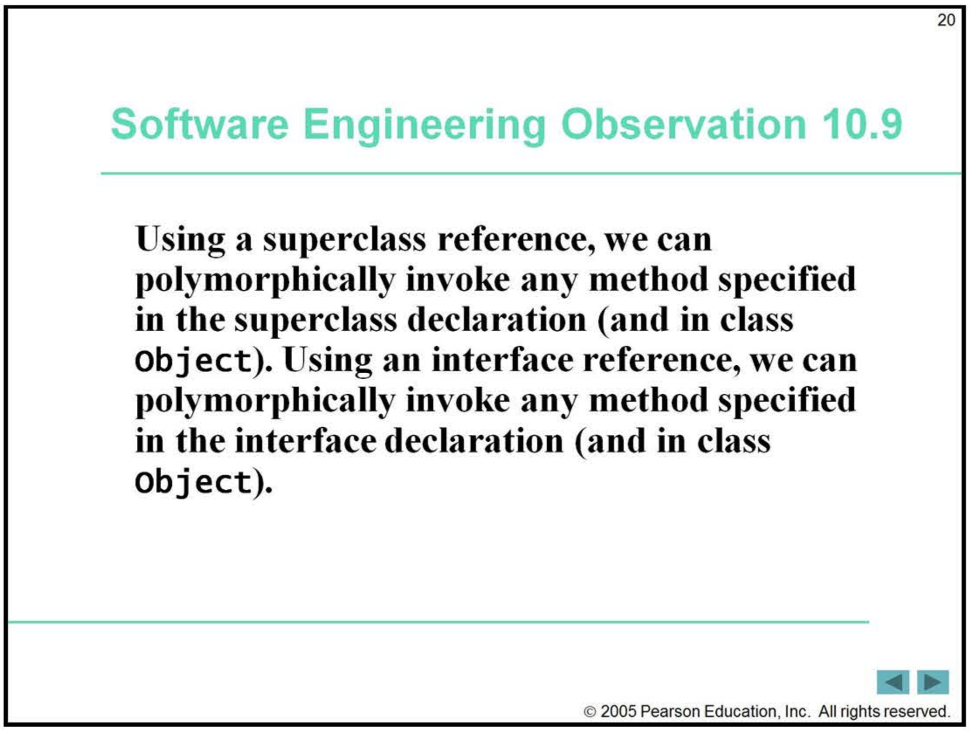
`.java`

(2 of 2)



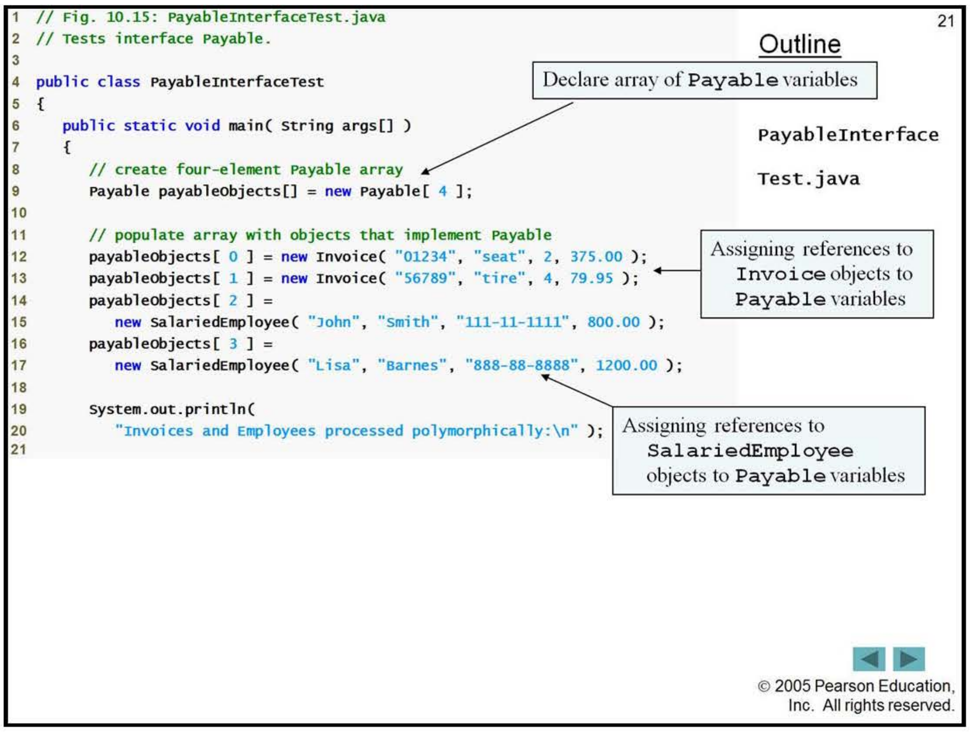
Software Engineering Observation 10.8

The “is-a” relationship that exists between superclasses and subclasses, and between interfaces and the classes that implement them, holds when passing an object to a method. When a method parameter receives a variable of a superclass or interface type, the method processes the object received as an argument polymorphically.



Software Engineering Observation 10.9

Using a superclass reference, we can polymorphically invoke any method specified in the superclass declaration (and in class `Object`). Using an interface reference, we can polymorphically invoke any method specified in the interface declaration (and in class `Object`).



Outline

`PayableInterface`

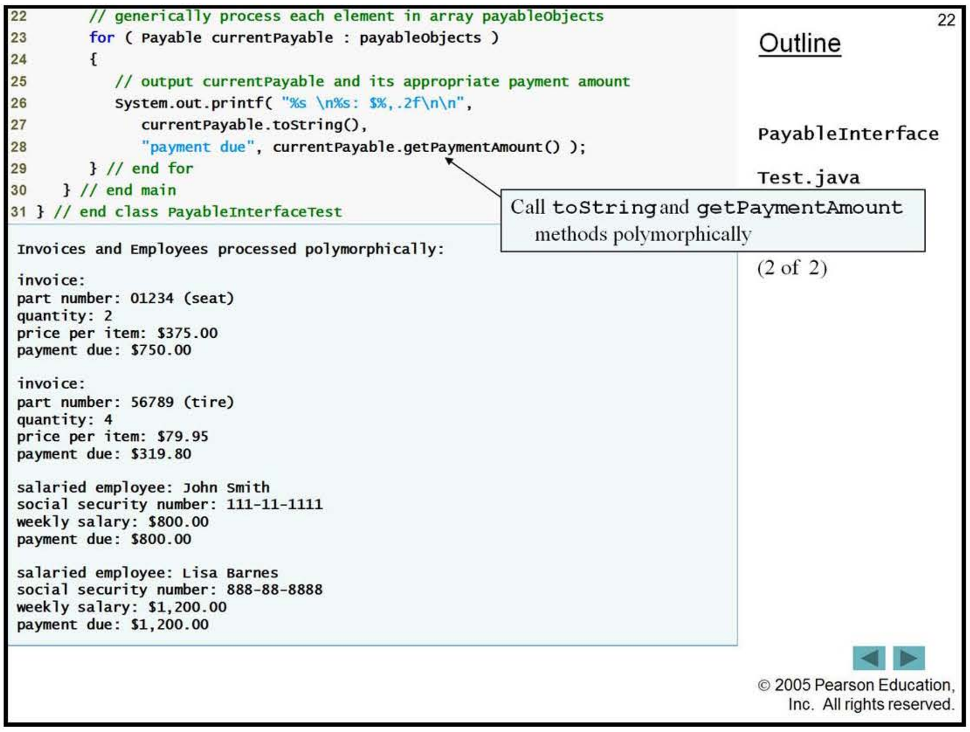
`Test.java`

```java
// Fig. 10.15: PayableInterfaceTest.java
// Tests interface Payable.

public class PayableInterfaceTest
{
   public static void main( String args[] )
   {
      // create four-element Payable array
      Payable payableObjects[] = new Payable[ 4 ];

      // populate array with objects that implement Payable
      payableObjects[ 0 ] = new Invoice( "01234", "seat", 2, 375.00 );
      payableObjects[ 1 ] = new Invoice( "56789", "tire", 4, 79.95 );
      payableObjects[ 2 ] =
         new SalariedEmployee( "John", "Smith", "111-11-1111", 800.00 );
      payableObjects[ 3 ] =
         new SalariedEmployee( "Lisa", "Barnes", "888-88-8888", 1200.00 );

      System.out.println(
         "Invoices and Employees processed polymorphically:\n" );

```

Declare array of `Payable` variables

Assigning references to `Invoice` objects to `Payable` variables

Assigning references to `SalariedEmployee` objects to `Payable` variables



Outline

```
      // generically process each element in array payableObjects
      for ( Payable currentPayable : payableObjects )
      {
         // output currentPayable and its appropriate payment amount
         System.out.printf( "%s \n%s: $%,.2f\n\n",
            currentPayable.toString(),
            "payment due", currentPayable.getPaymentAmount() );
      } // end for
   } // end main
} // end class PayableInterfaceTest
```

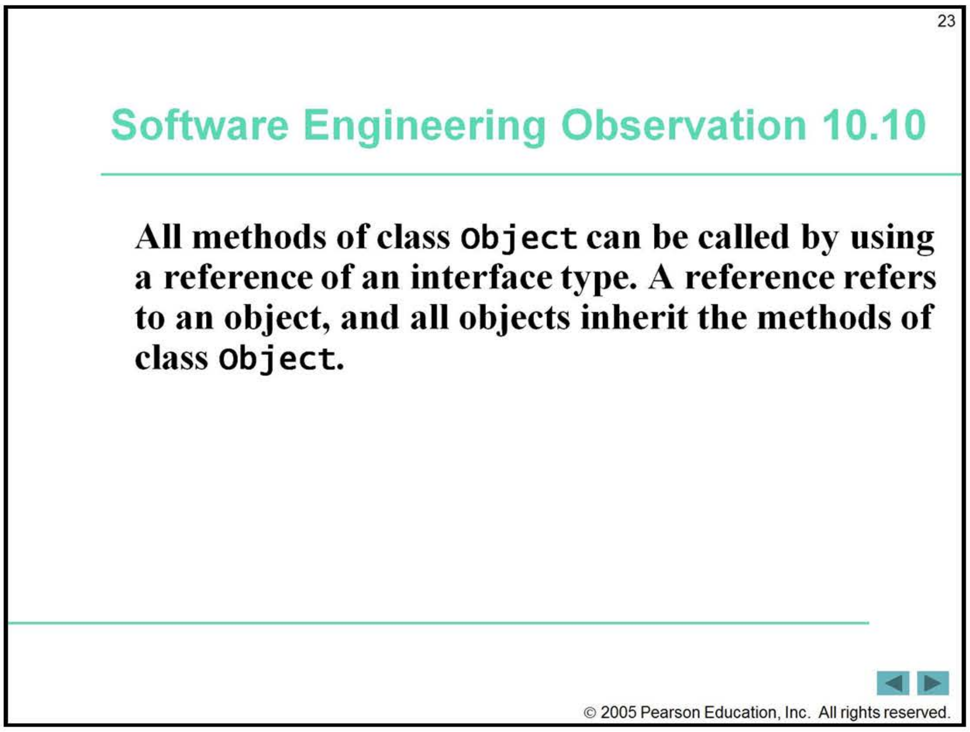
Call `toString` and `getPaymentAmount` methods polymorphically

```
Invoices and Employees processed polymorphically:

invoice:
part number: 01234 (seat)
quantity: 2
price per item: $375.00
payment due: $750.00

invoice:
part number: 56789 (tire)
quantity: 4
price per item: $79.95
payment due: $319.80

salaried employee: John Smith
social security number: 111-11-1111
weekly salary: $800.00
payment due: $800.00

salaried employee: Lisa Barnes
social security number: 888-88-8888
weekly salary: $1,200.00
payment due: $1,200.00
```

`PayableInterface`

`Test.java`

(2 of 2)



Software Engineering Observation 10.10

All methods of class `Object` can be called by using a reference of an interface type. A reference refers to an object, and all objects inherit the methods of class `Object`.

10.7.7 Declaring Constants with Interfaces

- **Interfaces can be used to declare constants used in many class declarations**
  - **These constants are implicitly `public`, `static` and `final`**
  - **Using a `static import` declaration allows clients to use these constants with just their names**

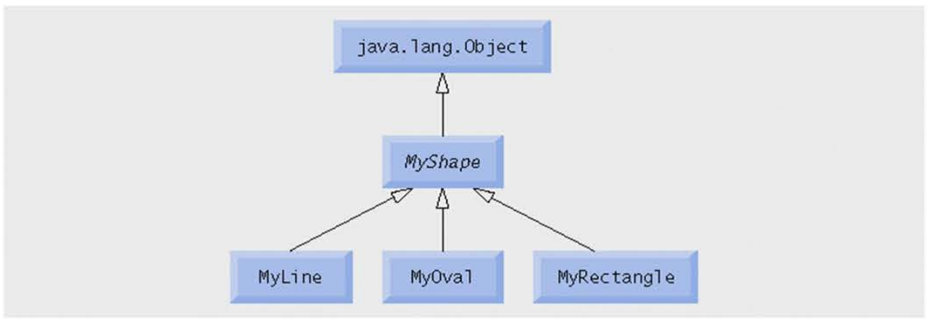


Fig. 10.17 | MyShape hierarchy.

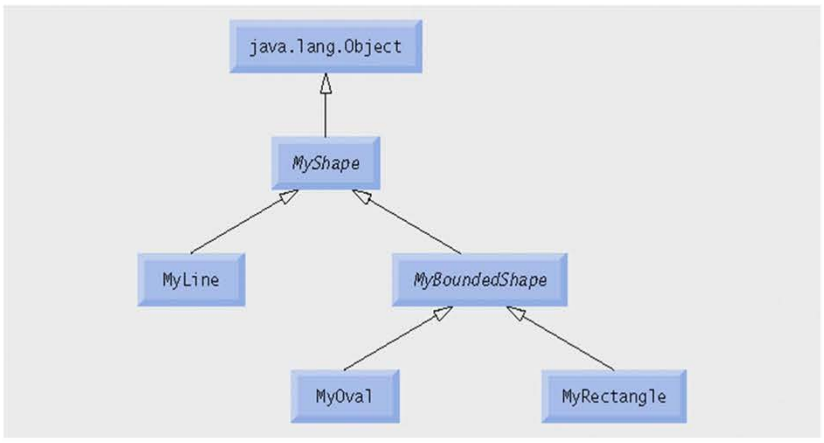


Fig. 10.18 | `MyShape` **hierarchy with** `MyBoundedShape`**.**

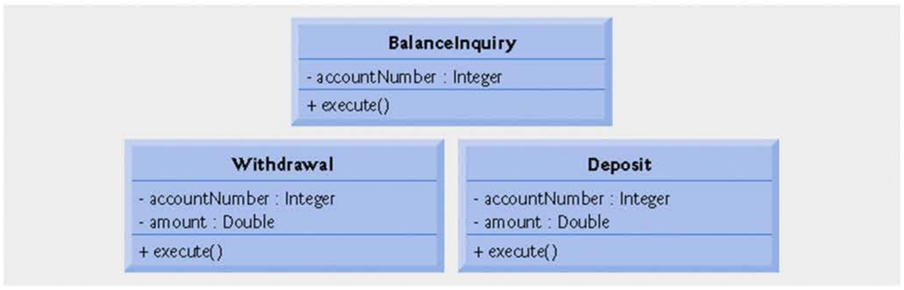


Fig. 10.19 | **Attributes and operations of classes** `BalanceInquiry`**,** `Withdrawal` **and** `Deposit`**.**