Chapter 5

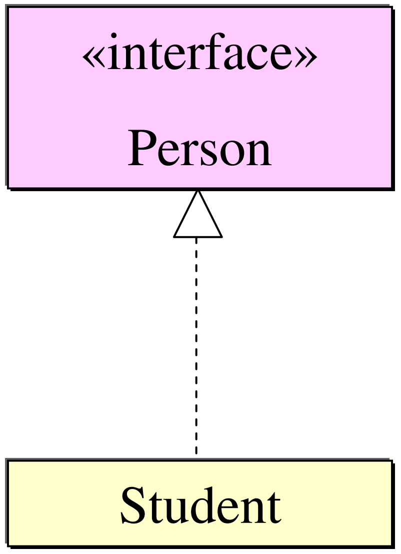
Interface

CSC 113
King Saud University
College of Computer and Information Sciences
Department of Computer Science

Interfaces

- An *interface* is something like an extreme case of an abstract class
  - However, *an interface is not a class*
  - *It is a type that can be satisfied by any class that implements the interface*
- The syntax for defining an interface is similar to that of defining a class
  - Except the word **`interface`** is used in place of **`class`**
  - **`public interface Person`**
- An interface specifies a set of methods that any class that implements the interface must have
  - It contains method headings and constant definitions only
  - It contains no instance variables nor any complete method definitions

The Person Interface



```
public interface Person
{
  public double getSalary(); // calculate salary, no implementation
} // end interface Person
```

Interfaces

- An interface serves a function similar to a base class, though it is not a base class
  - Some languages allow one class to be derived from two or more different base classes
  - This *multiple inheritance* is not allowed in Java
  - Instead, Java's way of approximating multiple inheritance is through interfaces

Interfaces

- An interface and all of its method headings should be declared public
  - They cannot be given private, protected
  - When a class implements an interface, it must make all the methods in the interface public
- Because an interface is a type, a method may be written with a parameter of an interface type
  - That parameter will accept as an argument any class that implements the interface

Interfaces

- To *implement an interface*, a concrete class must do two things:

1. It must include the phrase

   ```
   implements Interface_Name
   ```

   at the start of the class definition

   ```
   public class Student implements Person
   ```

   – If more than one interface is implemented, each is listed, separated by commas

2. The class must implement all the method headings listed in the definition(s) of the interface(s)

Implementation of an Interface

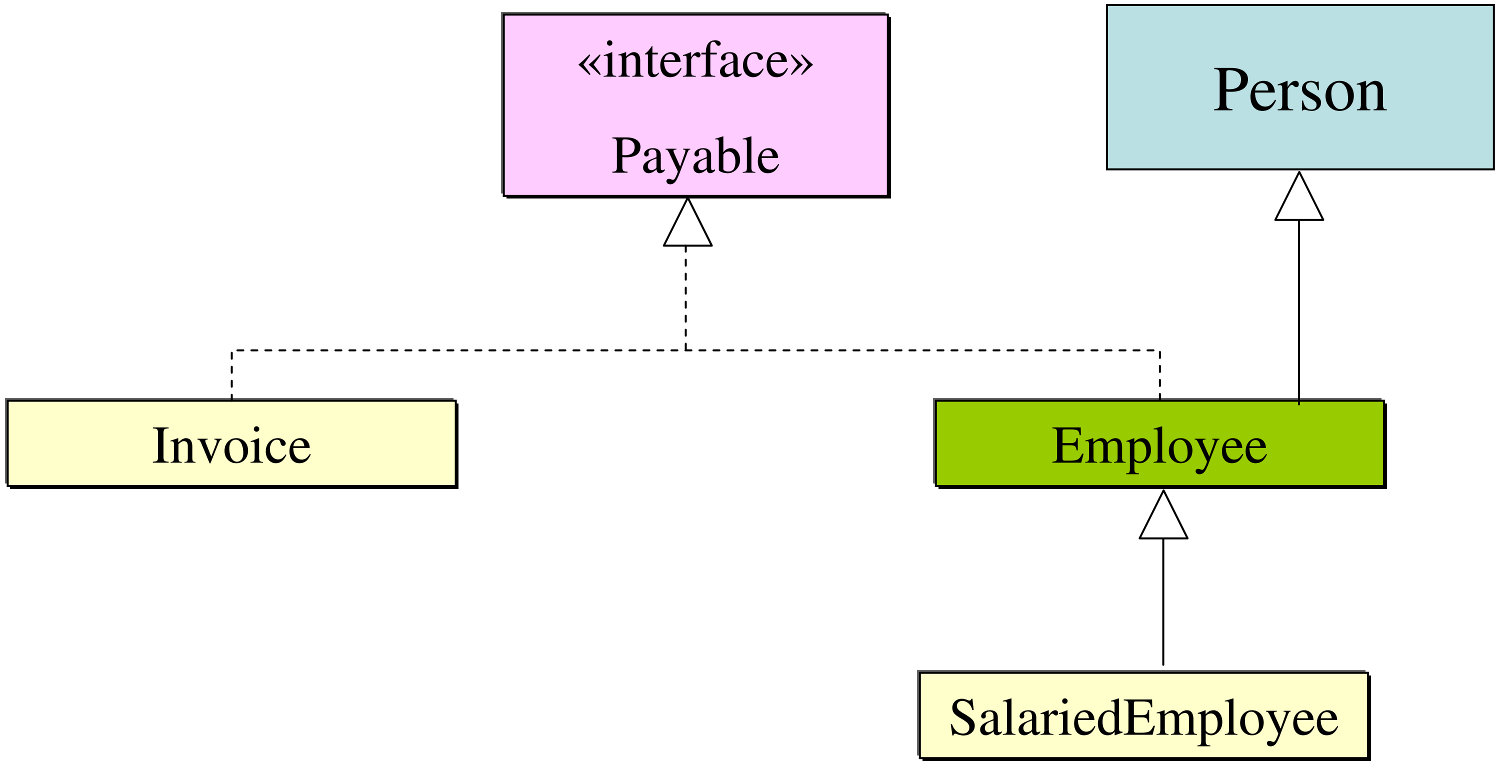
```
public class Student implements Person
{
   private int gpa;
   ……
   ……

   public double getSalary()
   {
      return (gpa * 200);
    }
}
```

Abstract Classes Implementing Interfaces

- Abstract classes may implement one or more interfaces
  - Any method headings given in the interface that are not given definitions are made into abstract methods
- A concrete class must give definitions for all the method headings given in the abstract class *and the interface*

An Abstract Class Implementing an Interface



Payable & Person Class implementation

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

```
// Person class.
public class Person
{  protected String  address;
    public Person (String ad)
    {
         address = new String (ad);
      }
} // end Person class
```

Invoice class implementation

```
// Invoice class implements Payable.
public class Invoice implements Payable
{ private String partNumber,
  private String partDescription;
  private int quantity;
  private double pricePerItem;
  // constructor
  public Invoice( String part, String description,
                          int count, double price )
  { partNumber = part;
    partDescription = description;
    setQuantity( count );
    setPricePerItem( price );
  }
// set part number
  public void setPartNumber( String part )
  {  partNumber = part;
  }
```

```
// get part number
  public String getPartNumber()
  { return partNumber; }
// set description
  public void setPartDescription( String description )
  {  partDescription = description; }
// get description
  public String getPartDescription()
  { return partDescription; }
// set quantity
  public void setQuantity( int count )
  { quantity = ( count < 0 ) ? 0 : count; }
  // get quantity
  public int getQuantity()
  { return quantity; }
// set price per item
  public void setPricePerItem( double price )
  {  pricePerItem = ( price < 0.0 ) ? 0.0 : price; }
```

Invoice class implementation: Cont

```
// get price per item
   public double getPricePerItem()
   {  return pricePerItem; }
   // 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() );
   }
   // method required to carry out contract with interface Payable
   public double getPaymentAmount()
   {  return getQuantity() * getPricePerItem();    }
} // end class Invoice
```

Employee Abstract class implementation

```
// Employee abstract superclass implements Payable.
public abstract class Employee extends Person implements Payable
{ private String firstName;
   private String lastName;
   private String socialSecurityNumber;
  // four-argument constructor
   public Employee( String first, String last, String ssn, String ad )
   { supper (ad);
      firstName = first;  lastName = last;
      socialSecurityNumber = ssn;
   } // end three-argument Employee constructor
   // set first name
   public void setFirstName( String first )
   { firstName = first; } // end method setFirstName
    // return first name
   public String getFirstName()
   { return firstName; } // end method getFirstName
```

Employee Abstract class implementation: Cont

```
  public void setLastName( String last )
   { lastName = last; } // end method setLastName
  public String getLastName()
   { return lastName; } // end method getLastName
   public void setSocialSecurityNumber( String ssn )
   { socialSecurityNumber = ssn;} // 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
```

SalariedEmployee Concrete class implementation

```
// SalariedEmployee class extends Employee, which implements Payable.
public class SalariedEmployee extends Employee
{    private double weeklySalary;
     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
   public void setWeeklySalary( double salary )
    {weeklySalary = salary < 0.0 ? 0.0 : salary; } // end method setWeeklySalary
   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
   public String toString()
   { return String.format( "salaried employee: %s\n%s: $%,.2f",
      super.toString(), "weekly salary", getWeeklySalary() ); } // end method toString
} // end class SalariedEmployee
```

PayableInterfaceTest

```
// 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( “Ali", “Yassin", "111-11-1111", 800.00, “Malaz” );
      payableObjects[ 3 ] = new SalariedEmployee( “Med", “Ahmed", "888-88-8888", 1200.00, “Makka” );
      System.out.println( "Invoices and Employees processed polymorphically:\n" );
      // generically process each element in array payableObjects
      for ( Payable currentPayable : payableObjects )
      { System.out.printf( "%s \n%s: $%,.2f\n\n", currentPayable.toString(), "payment due",
                                    currentPayable.getPaymentAmount() );
      } // end for
   } // end main
} // end class PayableInterfaceTest
```

Derived Interfaces (Extending an Interface)

- Like classes, an interface may be derived from a base interface
  - This is called *extending* the interface
  - The derived interface must include the phrase
    `extends` ***`BaseInterfaceName`***
- A concrete class that implements a derived interface must have definitions for any methods in the derived interface as well as any methods in the base interface

public interface X extends Y

Defined Constants in Interfaces

- An interface can contain defined constants in addition to or instead of method headings
  - Any variables defined in an interface must be public, static, and final
  - Because this is understood, Java allows these modifiers to be omitted
- Any class that implements the interface has access to these defined constants