Please answer the following questions using separate sheets of paper. Question point values are shown in parentheses.

1. (20) In class we discussed the KeyVector class that allows the user to retrieve an object based on its key field using its getObjByKey() method (see below).

```
public class KeyVector extends Vector
{
   public KeyVector()
   {
      super();
   }
   public Object getObjByKey(String x) throws NoSuchElementException,
                  ObjectNotManageableException
   {
      for (int i = 0; i < this.size(); i++)
      {
         if (this.elementAt(i) instanceof Manageable)
         {
           if (((Manageable)this.elementAt(i)).getKey().equals(x))
              return this.elementAt(i);
         }
         else
         {
           ObjectNotManageableException e =
             new ObjectNotManageableException(i);
           throw e;
         }
      }
      NoSuchElementException e =
         new NoSuchElementException("Object not in vector");
      throw e;
   }
}
```

Modify this class by adding a method that would allow the user to add an object to the vector so that the object is placed in the vector according to the value of its key field in ascending – smallest to largest – order. The API for this new method is:

```
public void addElementByKey(Object x) throws ObjectNotManageableException
```

In completing this new method, one or both of the following methods available in the Vector class might be helpful (see next page):

public final synchronized void **setElementAt**(Object obj, int index)

Sets the component at the specified index of this vector to be the specified object. The previous component at that position is discarded.

The index must be a value greater than or equal to 0 and less than the current size of the vector.

public final synchronized void **insertElementAt**(Object obj, int index)

Inserts the specified object as a component in this vector at the specified index. Each component in this vector with an index greater or equal to the specified index is shifted upward to have an index one greater than the value it had previously.

The index must be a value greater than or equal to 0 and less than or equal to the current size of the vector.

Assume that the user will not use the Vector class's regular addElement() method. That is, only the new addElementByKey() will be used which should insure that as new objects are added, they will be added to a vector that is already in ascending order.

First write pseudocode that outlines your strategy. Following this, write the correct Java to implement the pseudocode (the code for the addElementByKey() method).

2. (15) Are the concepts of an "abstract class" and "polymorphism" related? Explain.

3. (15) Assume you have two classes: Department and Employee. A department can be related to many employees and an employee can be related to a single department.

   How would you set up the two classes so that they can support these relationships. You do not have to write Java code but you need to provide a fairly detailed explanation.

4. (20) Consider the Account class and its Checking subclass from assignment 4:

```
public abstract class Account
{
   private String fAcctNo;
   private String fSSN;
   protected double fBalance;

   public Account(String acctNo, String ssn, double bal)
   {
      fAcctNo = acctNo;
      fSSN = ssn;
      fBalance = bal;
   }
   //Accessor Methods not shown here
   //Action method
   public void deposit(double amt)
   {
      fBalance = fBalance + amt;
   }
   //Abstract methods
   public abstract void doMonthEnd();
   public abstract void withdrawal(double amt);
}

public class Checking extends Account
{
   double fTransCharge;

   public Checking(String acctNo, String ssn, double bal)
   {
      super(acctNo, ssn, bal);
   }
  // A number of methods not shown …

   public void withdrawal(double amt)
   {
      fBalance = fBalance - (amt + fTransCharge);
   }
}
```

Assume that you want to modify the withdrawal() method so that if the amount to be withdrawn causes the balance to drop below zero, a NegativeBalanceException is thrown and the balance is not changed.

(continued on the next page)

The following try/catch block shows a hypothetical call to the withdrawal() method:

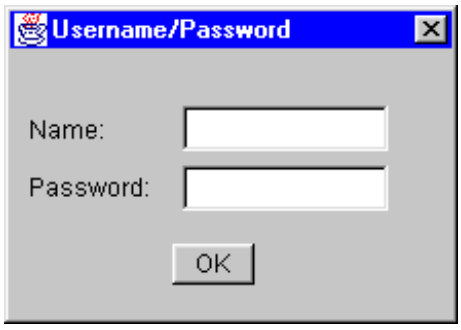
```
// variable 'amount' indicates the withdrawal amount
try
{
    accnt.withdrawal(amount);
}
catch (NegativeBalanceException e)
{
    double negBal = e.getNegativeBalance();
    System.out.println("Transaction cancelled – would cause negative balance of "
                                    + negBal);
}
```

Given this information, define the NegativeBalanceException class (using Java). Also implement the changes that would need to be made to the Checking class's withdrawal() method.

5. (10) The following is the API for Symantec's PasswordDialog class:

```
public class PasswordDialog extends ModalDialog
    public PasswordDialog(Frame parent, String title)
    public PasswordDialog(Frame parent)
    public PasswordDialog(Frame parent, boolean modal)
    public PasswordDialog(Frame parent, String title, boolean modal)
    public String getUserName()
    public String getPassword()
    public void setUserName(String name)
    public void setPassword(String pass)
```

When displayed, the dialog box looks like:



Given this API, write a code ***segment*** that first instantiates and displays a password dialog box. After the user clicks on OK, the code segment then sets the String variable userName equal to the value the user entered into the dialog's Name text field and sets the String variable userPassword equal to the value the user entered into the dialog's Password text field.

6. (20) Write the **complete** Java definition for a class named “Finance” which is stored in a package named “financial”. This class should include a **class method** named presVal that computes the present value using the following formula:

$$PV = \frac{FV}{(1 + \frac{r}{m})^{(n * m)}}$$

where: FV is the future value.
n is the number of years.
m is the number of times interest is compounded per year.
r is the annual interest rate.

After creating the class definition, write the **code segment** that would use this class (from a project’s Frame class for example). Just show the statement(s) that call the method (assume that any arguments used by the call have already been declared and defined).

Sample Answers

1. Pseudocode:

   The approach shown here first adds the new element at the end of the current list. It then compares the key of this new element to the key of the element right above it (assuming you are thinking of the vector arranged vertically with cell 0 at the top). If the key of the new element is less than the one right above it, the two elements are swapped. This comparison and swapping continues until the key of the new element is not less than the key of the element right above it.

   Code:

```
public void addElementByKey(Object x) throws
                        ObjectNotManageableException
{
  // put the new element at the end
  this.addElement(x);

  // starting at the end, check adjacent cells and swap if necessary
  for (int i = (this.size()-1); i > 0; i--)
  {
     if (this.elementAt(i) instanceof Manageable)
     {
        String newKey = ((Manageable)this.elementAt(i)).getKey();
        String keyAbove = ((Manageable)this.elementAt(i-1)).getKey();
        if (newKey.compareTo(keyAbove) < 0)
        {
           // swap elements [i] and [i-1]
           Object temp = this.elementAt(i);
           this.setElementAt(this.elementAt(i-1),i);
           this.setElementAt(temp,i-1);
        }
        else
        {
           return;  // terminate loop if no swap is needed
        }
     }
     else
     {
        ObjectNotManageableException e =
           new ObjectNotManageableException(i);
        throw e;
     }
  }
}
```

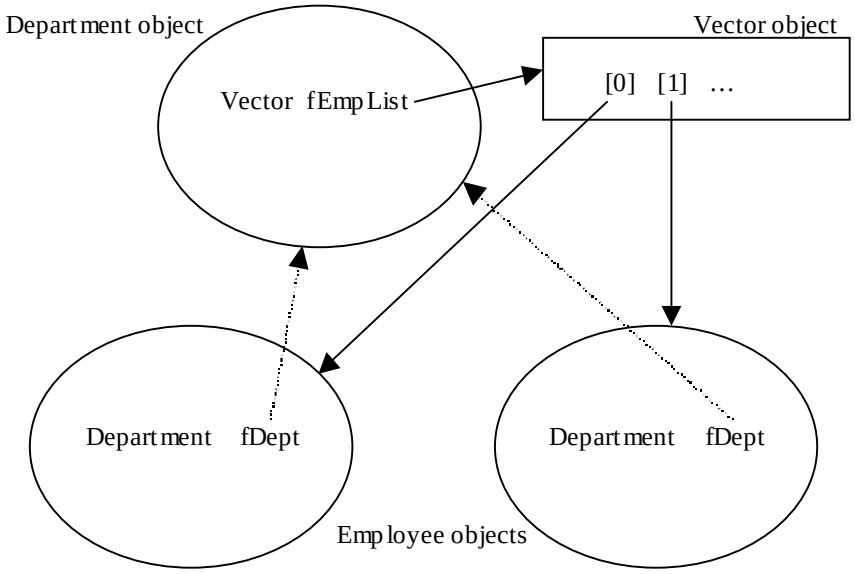
2. Yes, the concepts are strongly related. For polymorphism to work, an object reference must be able to point to different objects during execution. That is, an object reference must be able to point to either an object of its class or objects of any of its subclass.

   Abstract classes cannot be instantiated and therefore must be subclassed. This creates the ideal circumstances to implement polymorphic behavior. The abstract class must have subclasses, so an object reference of the abstract-class type can, be definition, point to difference classes (its subclasses) during execution.

3. For the Department class, you need to define an instance variable that can store references to many objects (a Vector, Hashtable, or array). This instance variable is called a "collection". Each element of this collection would point to an Employee object. This will support the relationship between the department and employees.

   The Employee class would need an instance variable that can point to a Department object. This will support the relationship between an employee and department.

   Graphically you have:



4. The NegativeBalanceException class definition is:

```
public class NegativeBalanceException extends Throwable
{
   private double fNegBalance;

   public NegativeBalanceException() {}
   public NegativeBalanceException(String s)
   {
      super(s);
   }
   public NegativeBalanceException(double d)
   {
      fNegBalance = d;
   }
   public double getNegativeBalance()
   {
      return fNegBalance;
   }
}
```

Changes to Checking class's withdrawal method are:

```
public void withdrawal(double amt) throws NegativeBalanceException
{
      if (fBalance - (amt + fTransCharge) < 0)
      {
         NegativeBalanceException e = new NegativeBalanceException(fBalance -
                                                   (amt + fTransCharge));
         throw e;
      }
      else
      {
         fBalance = fBalance - (amt + fTransCharge);
      }
}
```

5. Required code segment is (note that different constructors could be used):

```
PasswordDialog pd = new PasswordDialog(this,true);
pd.setVisible(true);
String userName=pd.getUserName();
String userPassword = pd.getPassword();
```

6. The class definition for the Finance class is:

```
package financial;

public class Finance
{
   public static double presVal(double fv, int n, int m, double rate)
   {
      double denom = Math.pow((1+(rate/m)),(n*m));
      return fv/denom;
   }
}
```

The code segment to use this class is:

```
double pv = Finance.presVal(fv,n,m,rate);
```