```
            SalariedEmployee otherSalariedEmployee =
                                  (SalariedEmployee)otherObject;
            return (super.equals(otherSalariedEmployee)
                   && (salary == otherSalariedEmployee.salary));
        }
    }
```

extra code on website

21. A version of the Date class with this definition of equals is in the subdirectory improvedEquals of the ch07 directory on the accompanying website.

```
public boolean equals(Object otherObject)
{
    if (otherObject == null)
      return false;
    else if (getClass() != otherObject.getClass())
       return false;
    else
    {
          Date otherDate = (Date)otherObject;
          return ( month.equals(otherDate.month)
                    && (day == otherDate.day)
                    && (year == otherDate.year) );
    }
}
```

22. Not the same class.

extra code on website

23. The following is included in the definition of EnhancedStringTokenizer on the accompanying website.

```
public Object nextElement()
{
    String token = super.nextToken();
    a[count] = token;
    count++;
    return (Object)token;
}
```

Programming Projects

1. Define a class named Person that contains two instance variables of type String that stores the first name and last name of a person and appropriate accessor and mutator methods. Also create a method named displayDetails that outputs the details of a person. Next, define a class named Student that is derived from Person, the constructor for which should receive first name and last name from the class Student and also assigns values to student id, course, and teacher name. This class should redefine the displayDetails method to person details as well as details of a student. Include appropriate constructor(s). Define a class named Teacher that

is derived from `Person`. This class should contain instance variables for the subject name and salary. Include appropriate constructor(s). Finally, redefine the `displayDetails` method to include all teacher information in the printout. Create a main method that creates at least two student objects and two teacher objects with different values and calls `displayDetails` for each.

2. Define a class named `Message` that contains an instance variable of type String named text that stores any textual content for the `Message`. Create a method named `toString` that returns the text field and also include a method to set this value.

   Next, define a class for `SMS` that is derived from `Message` and includes instance variables for the `recipientContactNo`. Implement appropriate accessor and mutator methods. The body of the `SMS` message should be stored in the inherited variable text. Redefine the `toString` method to concatenate all text fields.

   Similarly, define a class for `Email` that is derived from `Message` and includes an instance variable for the sender, receiver, and subject. The textual contents of the file should be stored in the inherited variable text. Redefine the `toString` method to concatenate all text fields.

   Create sample objects of type `Email` and `SMS` in your main method. Test your objects bypassing them to the following subroutine that returns true if the object contains the specified keyword in the text property.

```
public static boolean ContainsKeyword(Message messageObject,
                                       String keyword)
{
        if (messageObject.toString().indexOf(keyword,0) >= 0)
           return true;
        return false;
}
```

   Finally, include a method to encode the final message "This is Java" using an encoding scheme, according to which, each character should be replaced by the character that comes after it. For example, if the message contains character B or b, it should be replaced by C or c accordingly, while Z or z should be replaced with an A or a. If the final message is "This is Java", then the encoded message should be "UijtjtKbwb".

VideoNote
Solution to Programming Project 7.3

3. The following is some code designed by J. Hacker for a video game. There is an `Alien` class to represent a monster and an `AlienPack` class that represents a band of aliens and how much damage they can inflict:

```
class Alien
{
     public static final int SNAKE_ALIEN = 0;
     public static final int OGRE_ALIEN = 1;
     public static final int MARSHMALLOW_MAN_ALIEN = 2;

     public int type; // Stores one of the three above types
     public int health; // 0=dead, 100=full strength
     public String name;
```

```
    public Alien (int type, int health, String name)
    {
        this.type = type;
        this.health = health;
        this.name = name;
    }
}
public class AlienPack
{
      private Alien[] aliens;

      public AlienPack (int numAliens)
      {
          aliens = new Alien[numAliens];
      }
      public void addAlien(Alien newAlien, int index)
      {
          aliens[index] = newAlien;
      }
      public Alien[] getAliens()
      {
          return aliens;
      }
}
public int calculateDamage()
{
       int damage = 0;
       for (int i=0; i < aliens.length; i++)
       {
           if (aliens[i].type==Alien.SNAKE_ALIEN)
           {
               damage +=10;// Snake does 10 damage
           }
           else if (aliens[i].type==Alien.OGRE_ALIEN)
           {
               damage +=6;// Ogre does 6 damage
           }
            else if (aliens[i].type==
            Alien.MARSHMALLOW_MAN_ALIEN)
           {
               damage +=1;
            // Marshmallow Man does 1 damage
           }
       }
       return damage;
     }
}
```

The code is not very object oriented and does not support information hiding in the Alien class. Rewrite the code so that inheritance is used to represent the different types of aliens instead of the "type" parameter. This should result in deletion of the "type" parameter. Also rewrite the Alien class to hide the instance variables and create a getDamage method for each derived class that returns the amount of damage the alien inflicts. Finally, rewrite the calculateDamage method to use getDamage and write a main method that tests the code.

4. Define a class called Administrator, which is a derived class of the class SalariedEmployee in Display 7.5. You are to supply the following additional instance variables and methods:

   - An instance variable of type String that contains the administrator's title (such as "Director" or "Vice President").
   - An instance variable of type String that contains the administrator's area of responsibility (such as "Production", "Accounting", or "Personnel").
   - An instance variable of type String that contains the name of this administrator's immediate supervisor.
   - Suitable constructors, and suitable accessor and mutator methods.
   - A method for reading in an administrator's data from the keyboard.

   Override the definitions for the methods equals and toString so they are appropriate to the class Administrator.

   Also, write a suitable test program.

VideoNote
Solution to Programming Project 7.5

5. Give the definition of a class named Doctor whose objects are records for a clinic's doctors. This class will be a derived class of the class SalariedEmployee given in Display 7.5. A Doctor record has the doctor's specialty (such as "Pediatrician", "Obstetrician", "General Practitioner", and so forth; so use the type String) and office visit fee (use type double). Be sure your class has a reasonable complement of constructors, accessor, and mutator methods, and suitably defined equals and toString methods. Write a program to test all your methods.

6. Create a class called Vehicle that has the manufacturer's name (type String), number of cylinders in the engine (type int), and owner (type Person given next). Then, create a class called Truck that is derived from Vehicle and has the following additional properties: the load capacity in tons (type double since it may contain a fractional part) and towing capacity in pounds (type int). Be sure your class has a reasonable complement of constructors, accessor and mutator methods, and suitably defined equals and toString methods. Write a program to test all your methods.

   The definition of the class Person follows. Completing the definitions of the methods is part of this programming project.

```
public class Person
{
    private String name;
    public Person()
    {...}
    public Person(String theName)
    {...}
    public Person(Person theObject)
    {...}
    public String getName()
    {...}
    public void setName(String theName)
    {...}
    public String toString()
    {...}
    public boolean equals(Object other)
    {...}
}
```

7. Give the definition of two classes, `Patient` and `Billing`, whose objects are records for a clinic. `Patient` will be derived from the class `Person` given in Programming Project 7.6. A `Patient` record has the patient's name (inherited from the class `Person`) and primary physician of type `Doctor` defined in Programming Project 7.5 A `Billing` object will contain a `Patient` object, a `Doctor` object, and an amount due of type `double`. Be sure your classes have a reasonable complement of constructors, accessor and mutator methods, and suitably defined `equals` and `toString` methods. First write a driver program to test all your methods, then write a test program that creates at least two patients, at least two doctors, and at least two `Billing` records, and then prints out the total income from the `Billing` records.
8. Programming Project 4.10 required adding an instance variable to the `Pet` class defined in Display 4.15 to indicate if the pet is a dog or cat. A better organization is to define `Pet` as a superclass of the `Dog` and `Cat` classes. This organization eliminates the need for an instance variable to indicate the type of the pet. Do or redo Programming Project 4.10 with inheritance. The `acepromazine()` and `carprofen()` methods should be defined in the `Pet` class to simply return 0. Override both methods in the `Dog` and `Cat` classes to calculate the correct dosage. Write a main method with appropriate tests to exercise the changes.

9. Programming Project 6.18 asked you to use an array of Strings to store the fruits and vegetables shipped in a `BoxOfProduce` object for a CSA farm.

   Modify your solution further by creating a `Produce` class. This class should have an instance variable of type `String` for the name, appropriate constructors, and a public `toString()` method. Then create a `Fruit` and a `Vegetable` class that are derived from `Produce`. These classes should have constructors that take the name as a `String` and invoke the appropriate constructor from the base class to set the name.

   Next, modify the text file of produce so it indicates whether each item is a fruit or a vegetable. Here is one possible organization, although you can use others:

   ```
   Broccoli,Vegetable
   Tomato,Fruit
   Kiwi,Fruit
   Kale,Vegetable
   Tomatillo,Fruit
   ```

   Finally, modify the `BoxOfProduce` class so it creates an array of type `Produce` instead of type `String`. The class should read the produce from the text file and create instances of either `Fruit` or `Vegetable`, with the appropriate name, in the array. After a box is finished, loop through the contents of the array and output how many fruit and how many vegetables are in the box. The rest of the program should behave the same as the solution to Programming Project 6.18.