1. The word *extends* is used when defining a subclass of another class, for instance, “public class Circle extends Ellipse”. The word *implements* is used when stating that a class obeys the rules of a particular interface, for instance, “public class GuessingGame implements SolitaireGame”.

2. ```java
public class Student extends Person {
    private String _school;

    // Constructor:
    public Student(String name, int age, String school) {
        super(name, age);
        _school = school;
    }

    // Get method for school (other get methods are inherited):
    public String getSchool() {
        return _school;
    }
}
```

3. For this particular example there is no gain in making these *protected* since they are never directly referred to from within the *Student* class. However, see the next two problems.

4. The *Child* class will not compile. Even though it inherits the instance variable _x, it is not allowed to directly refer to _x since that variable is declare *private* inside the *Person* class. It can, however, access and modify _x if we add “getX” and “setX” methods to the *Person* class.

5. This time *Child* will compile with no error. The variable _x is declared to be *protected*, which means that it is accessible from the *Person* class and all subclasses of *Person*.

6. No. In Java, a class may *extend* only a single class. (Other languages, such as C++, permit what is called “multiple inheritance,” but this causes problems when there are variables or methods with identical names in two or more of the parent classes — which ones get inherited?)

7. ```java
public Circle(int rad) {
    super(); // create a default ellipse
    super.setSize(2*rad, 2*rad); // set the size of the ellipse
    _radius = rad;
}
```
// Override the "setSize" method with a do-nothing method:
public void setSize(int width, int height) {}

8. public void mouseDragged(MouseEvent e) {
    int x = e.getX();
    int y = e.getY();
    setLocation(x, y);
}

9. // Either year is not divisible by 100 and is divisible by 4, or else
   // year is divisible by 400:
   (year % 4 == 0 && year % 100 != 0) || year % 400 == 0

10. if ((year % 4 == 0 && year % 100 != 0) || year % 400 == 0) {
        System.out.println("leap year");
    } else {
        System.out.println("not leap year");
    }

11. It is illegal to assign an expression of type double to an int. Note that this is illegal even if
    the double value has no fractional part to it (as in the case shown).

12. double x = 10;
    int y = (int) x; // This converts the double value to an int

13. public double tax(double price) {
        return .065 * price;
    }

14. public double total(int quantity, double price) {
        return quantity * price + tax(quantity * price);
        // or: return quantity * price + quantity * tax(price);
        // or: return quantity * (price + tax(price));
    }

15. public boolean isSmall(int x) {
        return x > -10 && x < 10;
        // or: return -9 <= x && x <= 9;
    }

16. public double area(double a, double b, double c) {
        double s = (a + b + c)/2.0;
        return Math.sqrt(s * (s - a) * (s - b) * (s - c));
    }

17. 3
    10
    5
    16
    8
    4
    2
18. 

19. public class Simple implements AnInterface {
   public void printSomething() {
      int blah = 10; // doesn't matter what goes here
   }

   public int returnSomething(double x) {
      return 10; // doesn't matter what goes here
   }
}

20. A while loop tests its condition at the beginning of each repetition of the loop. A do loop tests its condition at the end of each repetition of the loop. Therefore, a do loop always performs at least one execution of the loop body.