Classes

Abstract Class

A class from which no object will ever be created.
An artificial class meant solely so that subclasses of it can take advantage of.
Starting point / Framework

(method without body)
All abstract classes contain 1 or more abstract method that classes must have

Final Class

A class you can't inherit from (you can't extend)
Opposite of abstract class.
Prevents you from extending, no subclasses are ever used
It has less compiler method.
Individual methods can be made final.
Overwriting - the child overwrites parent class

Class X implements Y, Z		Valid
Class X implements Y extends Z		Valid
Class X extends Y implements Z		Valid
Class X extends Y, Z		Invalid

and doesn't have variables
Interface - a Java class where all methods are abstract by default. When a class inherits from an interface, use implements rather than extends
e.g. a checkbox

Inner Class

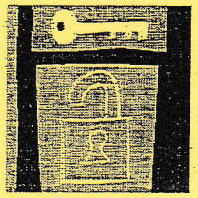
A class completely nested within another class.
Allows/provides help to the enclosing class
"Helper class" that no-one else needs or ever uses

Differences between Access Modifiers

- A summary of the differences between `private`, `protected`, and `public` when applied to a variable or method in a class:

Access Modifier	*Meaning*
`private`	Can be accessed only in the same class
`protected`	Can be accessed in the same class or in a subclass
`public`	Can be accessed in any class
None (plain)	Can be accessed in any class in the same package

- A class also has access to the protected variables and methods of all classes in the same package.



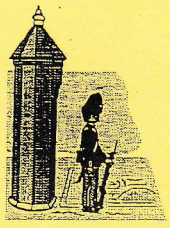
Practice with Java Access Modifiers

Suppose you have the class `Mystery` partially defined below:

```
public class Mystery {
    private int priv;
    protected int prot;
    int plain;
}
```

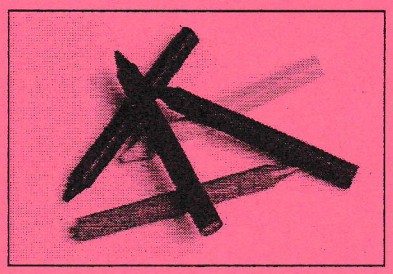
Tell whether or not each of these variables can be used in the following situations:

Situation	priv		prot		plain	
Within the methods in class `Mystery`	YES	NO	YES	NO	YES	NO
Within classes that extend `Mystery`	YES	NO	YES	NO	YES	NO
Within classes in the same package as `Mystery`	YES	NO	YES	NO	YES	NO
Within classes in the same file as `Mystery`	YES	NO	YES	NO	YES	NO
Within any other class	YES	NO	YES	NO	YES	NO



Practice with Abstract Classes

The Shape, Circle, and Rectangle Classes



The Shape Class

```
public abstract class Shape {
  private int x;
  private int y;
  private Color color;

  protected Shape(int x, int y, Color color) {
    this.x = x;
    this.y = y;
    this.color = color;
  }

  public abstract void draw(Graphics g);
  public abstract int getHeight();
  public abstract int getWidth();

  public Color getColor() {
    return color;
  }

  public int getX() {
    return x;
  }

  public int getY() {
    return y;
  }

  public void move(int dx, int dy) {
    x += dx;
    y += dy;
  }

  public void setColor(Color color) {
    this.color = color;
  }
}
```

Can be used by subclasses (allowed on methods and variables) Blocks client, not children

Practice with Abstract Classes

The Shape, Circle, and Rectangle Classes

The Circle Class

```
import java.awt.*;
public class Circle extends Shape {
  // Instance variables
  private int diameter;

  public Circle(int x, int y, Color color, int diameter) {
    super(x, y, color);
    this.diameter = diameter;
  }

  public void draw(Graphics g) {
    g.setColor(getColor());
    g.fillOval(getX(), getY(), diameter, diameter);
  }

  public int getHeight() {
    return diameter;
  }

  public int getWidth() {
    return diameter;
  }
}
```

Implemented 3 abstract methods

The Rectangle Class

```
import java.awt.*;
public class Rectangle extends Shape {
  // Instance variables
  private int width;
  private int height;

  public Rectangle(int x, int y, Color color, int width, int height){
    super(x, y, color);
    this.width = width;
    this.height = height;
  }

  public void draw(Graphics g) {
    g.setColor(getColor());
    g.fillRect(getX(), getY(), width, height);
  }

  public int getHeight() {
    return height;
  }

  public int getWidth() {
    return width;
  }
}
```