3.1 An object-oriented program

What is an object? The classes that we have constructed so far are definitions of objects – the object does not exist until it is instantiated by running the code. So far we have used programs with only one class, so only one object has been created. However, the power of objects lies in the fact that they contain or encapsulate their own data and methods. The implication of this is that if we define one class and instantiate it several times, we get several different objects, each with its own set of data and methods. If we designed a class to describe a bank account, we could use it many times to set up individual bank accounts. If we design code to describe an aeroplane, we can create an airforce. The following code is intended only as a demonstration of the power of object-oriented programming; it is incomplete for simplicity's sake and exaggerated for clarity's sake!

```java
import java.awt.*;
import java.awt.image.*;

abstract class Aeroplane extends Object
// create an aeroplane template - an abstract class would never be instantiated, only used to derive specialized // classes from:
{
    // give it some necessary data
    private int engines;
    private int seats;
    private int fuel;

    // tell it where it is on the screen
    int x, y;

    // construct an object
    public Aeroplane(int engines, int seats, int fuel)
    {
        this.engines = engines;
        this.seats = seats;
        this.fuel = fuel;
    }

    public abstract void Draw(Graphics g, ImageObserver o);

    class Airliner extends Aeroplane
    {
        private int maxPassengers;

        public Airliner(int engines, int seats, int fuel, int maxPassengers)
        {
            super(engines, seats, fuel);
            this.maxPassengers = maxPassengers;
        }

        public void Draw(Graphics g, ImageObserver o)
        {
            // Graphics code goes here
        }
    }

    class Fighter extends Aeroplane
    {
        private int guns;

        public Fighter(int engines, int seats, int fuel, int guns)
        {
            super(engines, seats, fuel);
            this.guns = guns;
        }

        public void Draw(Graphics g, ImageObserver o)
        {
            // Graphics code goes here
        }
    }
}
```

This example demonstrates a way we might choose to implement a number of different aeroplanes. We have chosen aeroplanes because all aeroplanes have some things in common, which gives us an opportunity to abstract those qualities that all aeroplanes share and to use them as a template for building any type of aeroplane, thus saving us