```
121            }
122        }
123        // if we don't find the bike, tell the user and return nothing
124        System.out.println("Bike with number '" + bikeNum + "' not found" + "\n");
125        return null;
126    }
127
128    public void showDetails( ){
129        // print out all the details
130        System.out.println("Details for bike number '" + bikeNumber + "'");
131        System.out.println("DEPOSIT: " + deposit);
132        System.out.println("RATE: " + rate + "\n");
133    }
134
135    public void calculateCost(int numberOfDays){
136        // work out the cost
137        int cost = deposit + (rate*numberOfDays);
138        System.out.println("COST would be £" + cost + "\n");
139
140 }
```

Figure 11.8 Code for the Bike class (continued)

a. *For purposes of computational efficiency, bikeList is implemented as a static array of :Bikes and findBike() as a static method in the Bike class. For more information on the Java keyword static, see Deitel and Deitel (2003).*

:Bikes until it finds one with a bike number that matches the value of bikeNum, which is passed in as a parameter. When a match is found the reference to the :Bike is returned to the calling method. Notice that the signature for this method, +findBikeByNumber(bikeNum:int):Bike, specifies that a Bike reference must be returned. All of the other methods we have met so far have specified a void return, meaning nothing is returned by the method.

The method showDetails() is declared on line 128. This method displays the bike number, deposit and rate of the :Bike to which it is sent. The method calculateCost() is declared on line 135. This method calculates the total cost of hiring the :Bike to which it was sent.

Sequence diagram

For students new to the topic, trying to follow the sequence of execution in an object-oriented program can be very confusing. The architecture of the code is dictated by the classes, but the sequence of execution is dictated by the use cases. The effect of this is that the sequence of execution jumps about all over a code listing. In this section we compare the sequence diagram for one of the 'Issuebike' use case scenarios, with the code that implements it. We map each message in the sequence diagram to the line of code