WIA2002: Software Modelling
Semester 1, Session 2016/17

Lecture 11: Design Modelling
(Implementation)

Learning Objectives

- Know how to implement a class diagram in a relational database.
- Know how to implement class and sequence diagrams in coding.

IMPLEMENTING A CLASS DIAGRAM IN A RELATIONAL DATABASE

Implementing a class diagram in a relational database

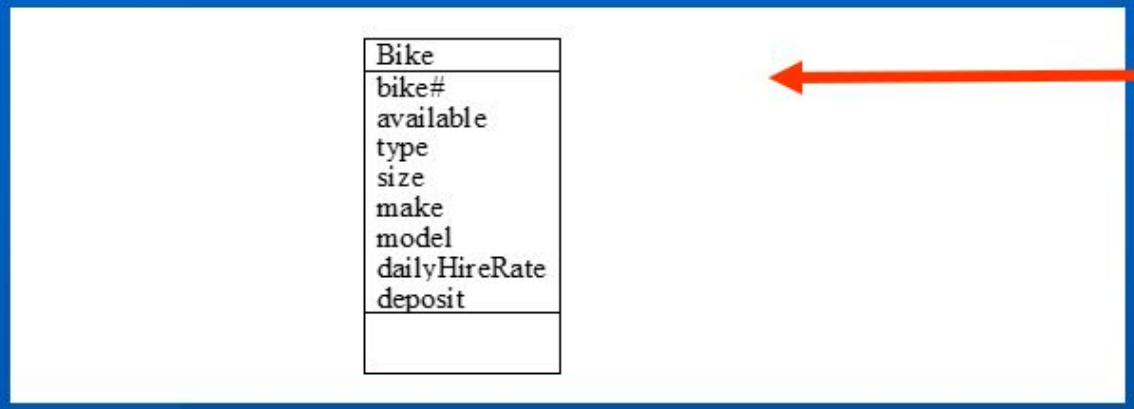
- A **database** stores, organizes and maintains data to support the operations of an organization.

- An **object-oriented database** provides the facilities of a traditional database, and supports the complex data structures of object-oriented systems.

- For an O-O program to access a relational database we need code to establish a connection
  - e.g. JDBC (Java Database Connectivity) - interface that interacts with both the code and the database.

Implementing a class diagram in a relational database

- Basic rule - one class maps to one table



Bike class

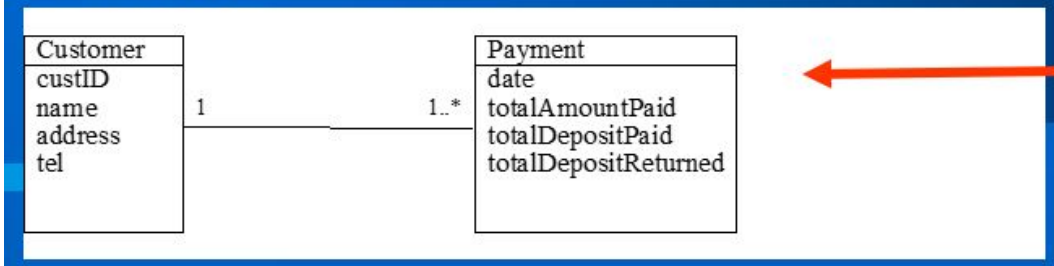
Primary key

Bike table

Bike No.	Available	Type	Size	Make	Model	Daily Hire Rate	Deposit
249	On hire	mountain	woman's	Scott	Atlantic Trail	£8.00	£50.00
250	Available	tourer	man's	Raleigh	Pioneer	£9.00	£60.00
251	On hire	mountain	woman's	Scott	Atlantic Trail	£8.00	£50.00
252	On hire	tourer	man's	Dawes	Galaxy	£8.00	£50.00
253	Available	mountain	child's	Raleigh	Chopper	£5.00	£25.00

Implementing a one to many association in a relational database

- First Method



One to many association

CustID	Name	FirstName	Street	Town	PhoneNo
1	Sykes	Jim	2 High Road	Greenwood	01395 211056
2	Perle	Lee	14 Duke Street	Greenwood	01395 237851
3	Hargreaves	Les	11 Forest Road	Prestwich	01462 501339
4	James	Sheena	4 Duke Street	Greenwood	01395 237663
5	Robins	Charlie	11 Juniper Road	Greenwood	0 1395 267843

Customer table

Payment No.	Date	Total amount paid	Total deposit paid	Total deposit returned
401	19/03/04	£56.00	£50.00	£50.00
402	19/03/04	£20.00	£25.00	£25.00
403	19/03/04	£145.00	£80.00	£80.00
404	20/03/04	£186.00	£100.00	£84.00
405	20/03/04	£44.00	£40.00	£40.00

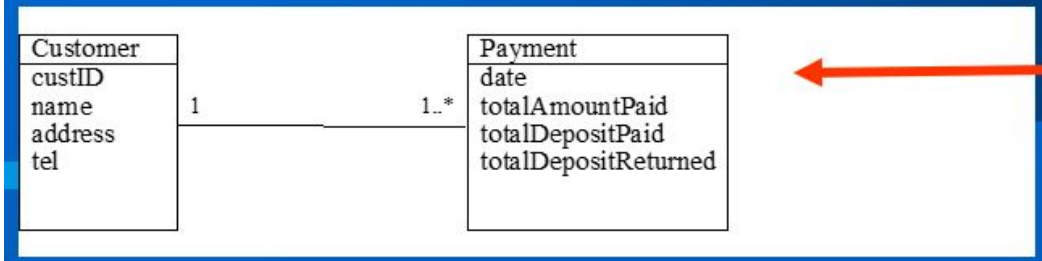
Payment table

CustID	Payment No.
1	409
2	513
2	405
3	404
11	501

Customer-Payment table

Implementing a one to many association in a relational database

- Second Method



One to many association

Customer

CustID	Name	FirstName	Street	Town	PhoneNo
1	Sykes	Jim	2 High Road	Greenwood	01395 211056
2	Perle	Lee	14 Duke Street	Greenwood	01395 237851
3	Hargreaves	Les	11 Forest Road	Prestwich	01462 501339
4	James	Sheena	4 Duke Street	Greenwood	01395 237663
5	Robins	Charlie	11 Juniper Road	Greenwood	0 1395 267843

Customer table

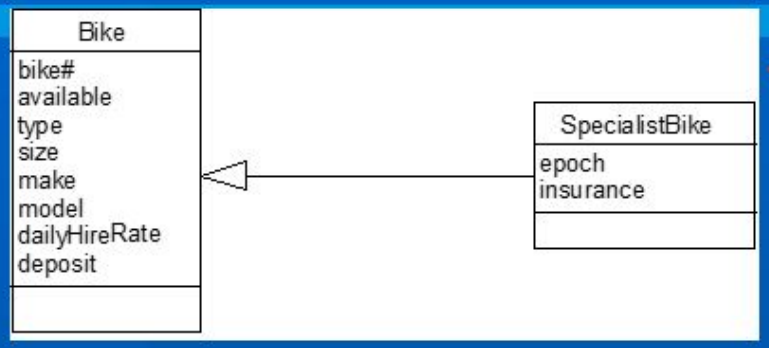
Payment

foreign key

Payment No	CustID	Date	Total amount paid	Total deposit paid	Total deposit returned
411	17	19/03/04	£56.00	£50.00	£50.00
412	20	19/03/04	£20.00	£25.00	£25.00
413	6	19/03/04	£145.00	£80.00	£80.00
414	3	20/03/04	£186.00	£100.00	£84.00
415	17	21/03/04	£44.00	£40.00	£40.00

Payment table with foreign key

Implementing a inheritance in a relational database



Inheritance relationship

normal bikes

Bike No.	Available	Type	Size	Make	Model	Epoch	Insurance	Daily Hire Rate	Deposit
249	On hire	mountain	woman's	Scott	Atlantic Trail			£8.00	£50.00
250	Available	tourer	man's	Raleigh	Pioneer			£9.00	£60.00
251	On hire	mountain	woman's	Scott	Atlantic Trail			£8.00	£50.00
252	On hire	tourer	man's	Dawes	Galaxy			£8.00	£50.00
253	Available	mountain	child's	Raleigh	Chopper			£5.00	£25.00
254	Available	tandem	man's	Sunbeam	Voyager	1930s	£15.00	£20.00	£100.00

specialist bike

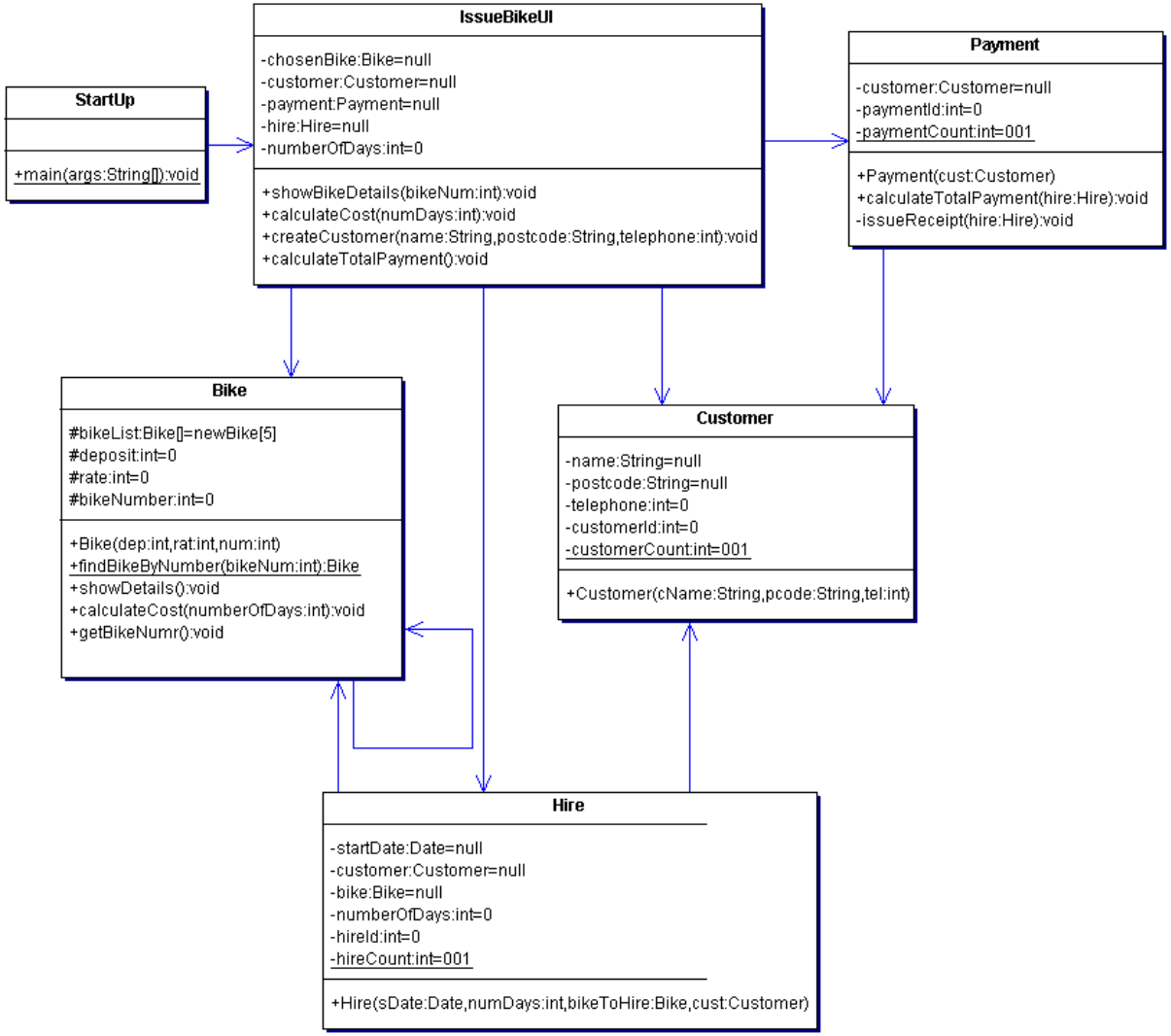
Visual Paradigm

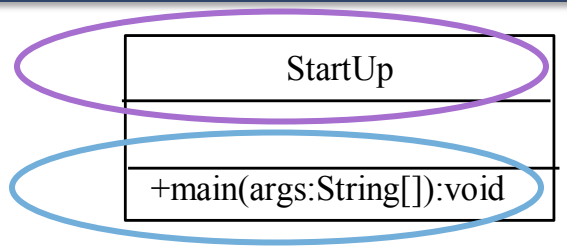
- Visual Paradigm provide feature to generate Class Diagram from Entity Relationship Diagram (ERD).

- Refer to **Lab 9** in Visual Paradigm Guidelines.

IMPLEMENTING CLASS AND SEQUENCE DIAGRAMS

Implementation of a class diagram

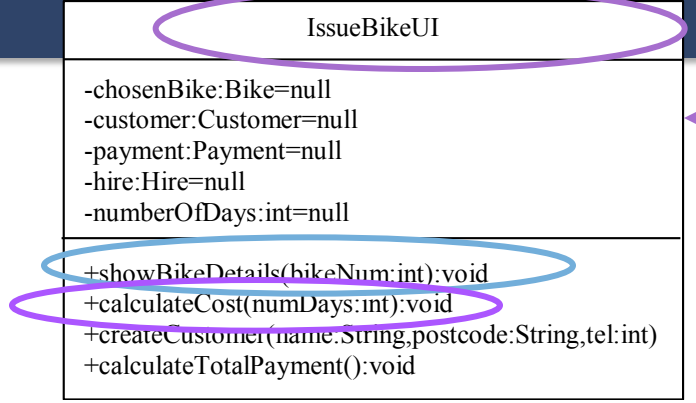




Class diagram

Code

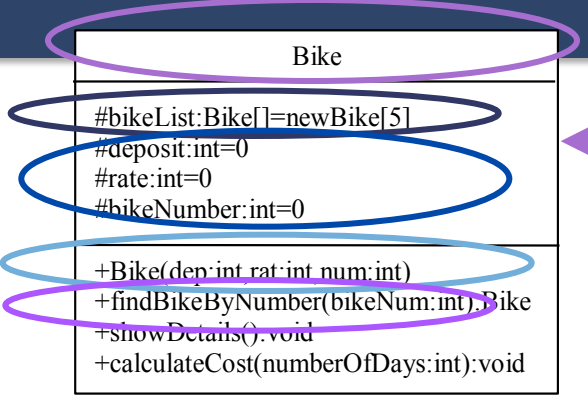
```
01      package bikeshop;
02
03      /* Generated by Together */
04
05      public class StartUp {
06
07          public static void main(String[] args){
08
09              /* This little program will run through the methods on IssueBikeUI
10               * calling each in turn, like a user with a front end would do. */
11
12              // First, create the UI
13              IssueBikeUI ui = new IssueBikeUI();
14
15              // 1. Show details for chosen bike
16              ui.showBikeDetails(100);
17
18              // 2. Calculate cost of hiring this bike for 5 days
19              ui.calculateCost(5);
20
21              // 3. Create new customer, payment and hire
22              ui.createCustomer("Les Hargreaves", "PW2 6TR", 01462501339);
23
24              // 4. Calculate the total cost
25              ui.calculateTotalPayment();
26          }
27      }
```



Class diagram

Code

```
/* Generated by Together */

package bikeshop;

import java.util.Date;

public class IssueBikeUI {

    // Set up the member variables
    private Bike chosenBike = null;
    private Customer customer = null;
    private Payment payment = null;
    private Hire hire = null;
    private int numberOfDays = 0;

    public void showBikeDetails(int bikeNum){
        // Find the bike by its number
        chosenBike = Bike.findBikeByNumber(bikeNum);
        if(chosenBike !=null){
            // then ask it for its details
            chosenBike.showDetails();
        }
    }
    public void calculateCost(int numDays){
        // set the member variable so it can be used later
        numberOfDays = numDays;
        // then ask the bike for the cost
        chosenBike.calculateCost(numDays);
    }

    public void createCustomer(String name, String postcode, int telephone){
        // Create a customer and associated hire and payment
```



Class diagram

Code

```
package bikeshop;

public class Bike {

    // create the BikeList
    protected static Bike[] bikeList = new Bike[5];
    //  set up member variables
    protected int deposit = 0;
    protected int rate = 0;
    protected int bikeNumber = 0;

    /* This block is run when the class is loaded and sets up our bike store.  It arbitrarily
     * populates the attributes: deposit, rate and bikeNumber */
    static{
        int j = 0;
        for(int i=10;i<15;i++){
            Bike b = new Bike(i, i, (j*100));
            bikeList[j] = b;
            j++;
        }
    }

    public Bike(int dep, int rat, int num){
        // set the member variables
        deposit = dep;
        rate = rat;
        bikeNumber = num;
    }

    public int getDeposit(){
        return deposit;
    }

    public int getRate(){
        return rate;
    }

    public int getBikeNumber(){
        return bikeNumber;
    }

    public static Bike findBikeByNumber(int bikeNum){
```

Implementation sequence diagram

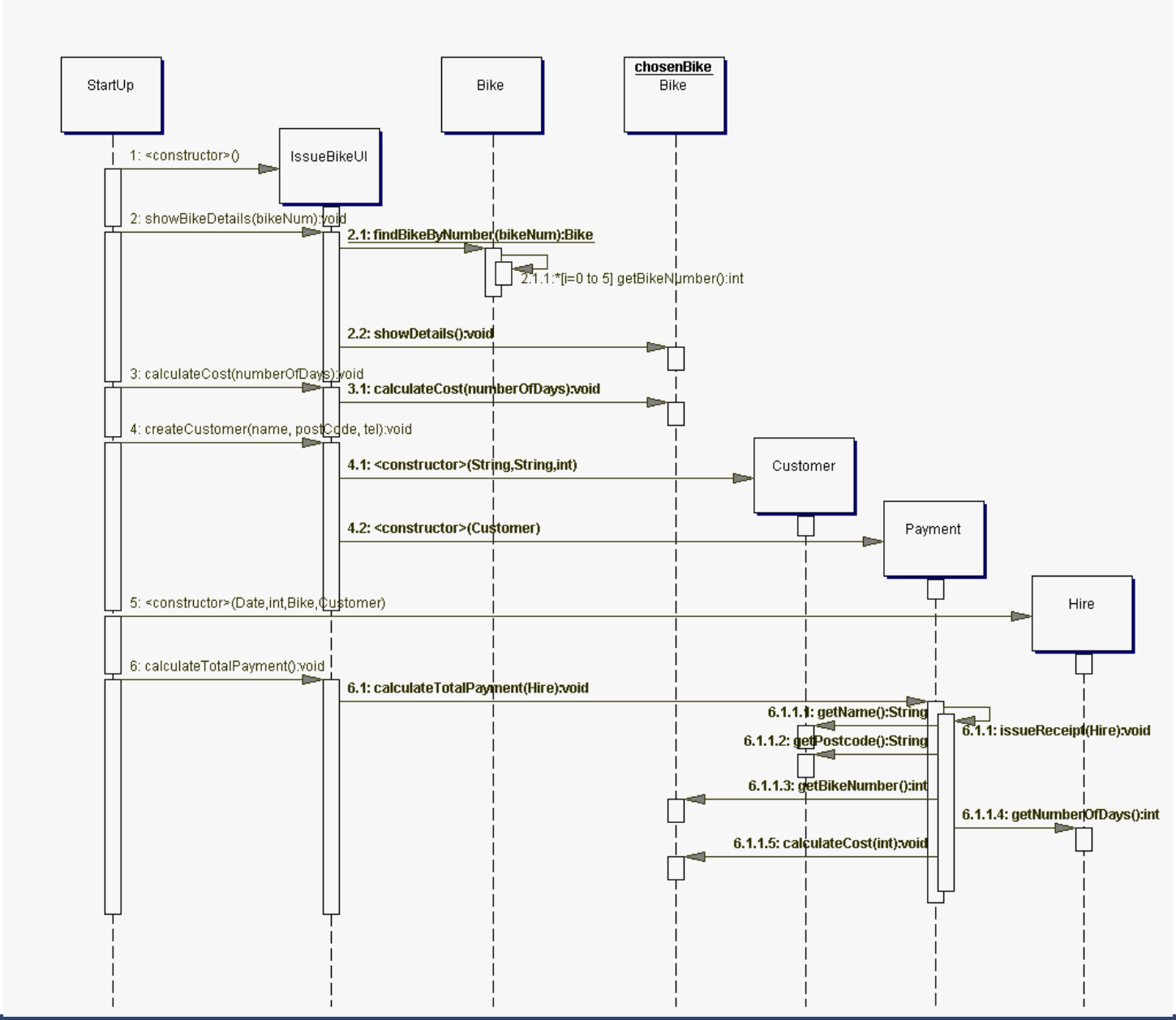
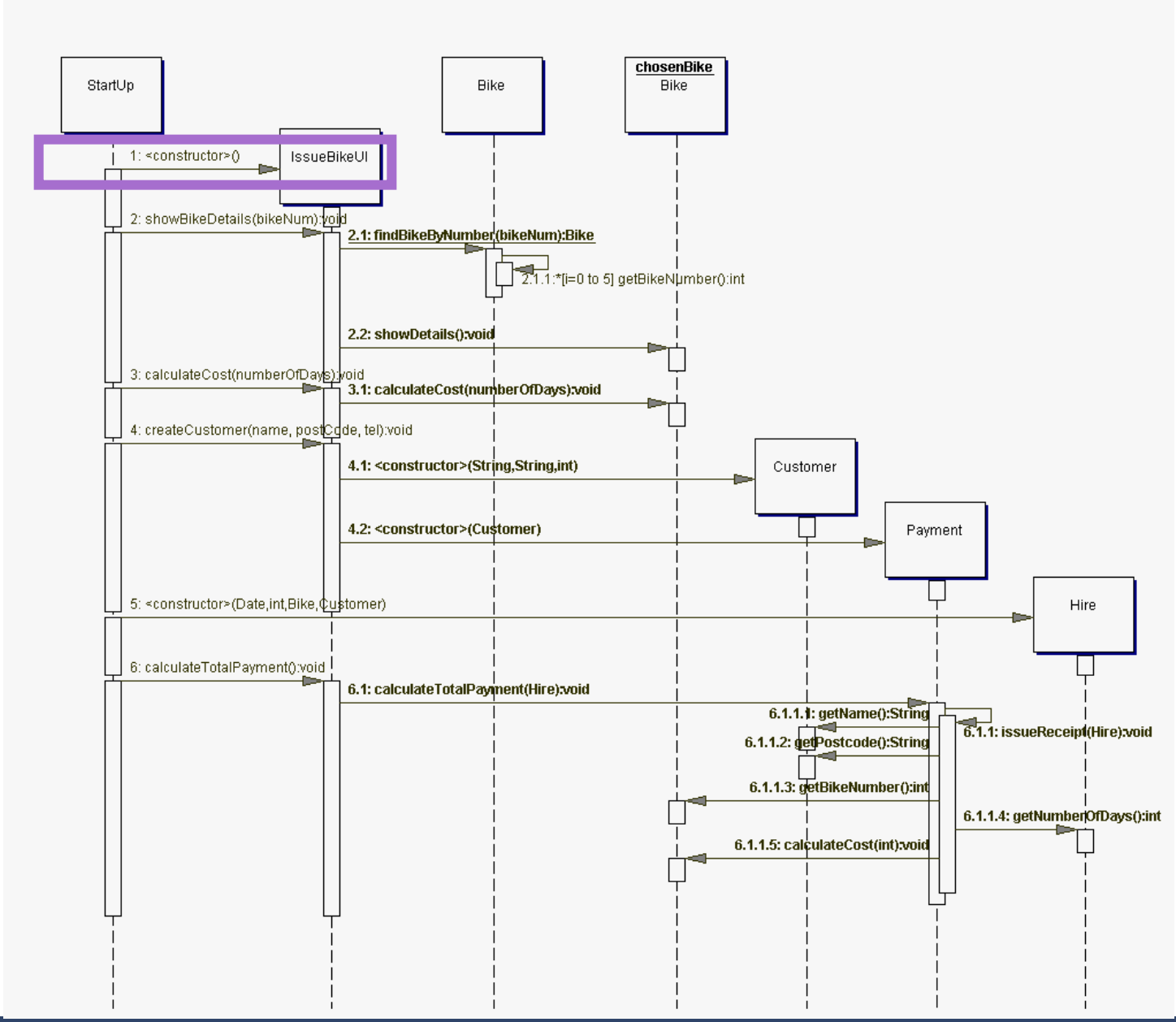


Table mapping sequence diagram messages to lines of code

Sequence diagram message	Code line	Sending object/class	Receiving object/class
1 constructor()	13	StartUp	IssueBikeUI
2 showBikeDetails(bikeNum)	16	StartUp	IssueBikeUI
2.1 findBikeByNumber(bikeNum)	44	IssueBikeUI	Bike
2.1.1 getBikeNumber()	116	Bike	Bike
2.2 showDetails()	47	IssueBikeUI	Bike
3 calculateCost(numberOfDays)	19	StartUp	IssueBikeUI
3.1 calculateCost(numberOfDays)	54	IssueBikeUI	Bike
4 createCustomer(name,postcode,tel)	22	StartUp	IssueBikeUI
4.1 constructor(String,String,int)	59	IssueBikeUI	Customer
4.2 constructor(Customer)	60	IssueBikeUI	Payment
5 constructor(Date,int,Bike,Customer)	61	IssueBikeUI	Hire
6 calculateTotalPayment()	25	StartUp	IssueBikeUI
6.1 calculateTotalPayment(Hire)	66	IssueBikeUI	Payment
6.1.1. issueReceipt(Hire)	237	Payment	Payment
6.1.1.1 getName()	242	Payment	Customer
6.1.1.2 getPostcode()	243	Payment	Customer
6.1.1.3 getBikeNumber()	247	Payment	Bike
6.1.1.4 getNumberOfdays()	248	Payment	Hire
6.1.1.5 calculateCost(int)	250	Payment	Bike

Implementation sequence diagram



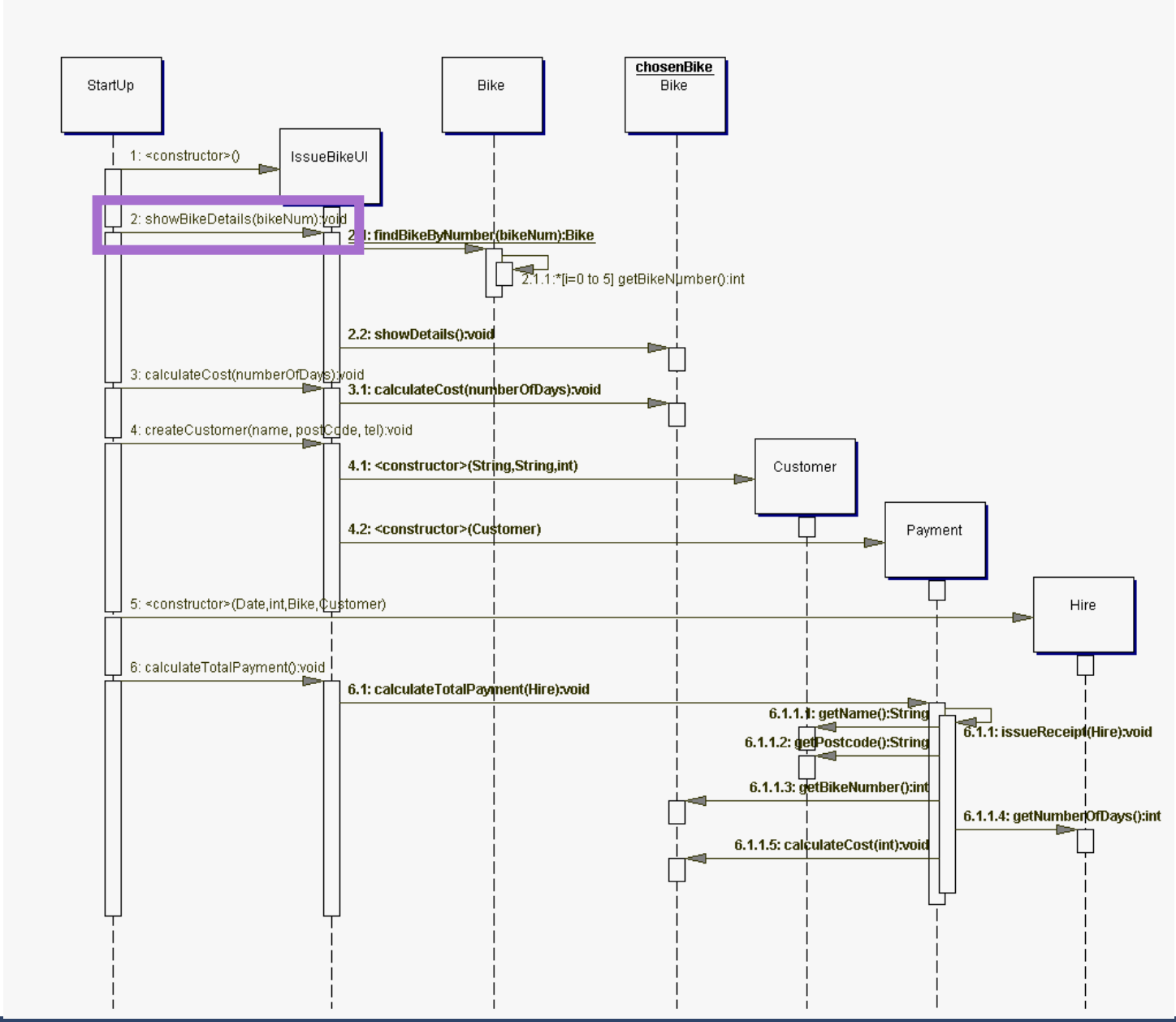
StartUp class

```
01      package bikeshop;
02
03      /* Generated by Together */
04
05      public class StartUp {
06
07          public static void main(String[] args){
08
09              /* This little program will run through the methods on IssueBikeUI
10               * calling each in turn, like a user with a front end would do. */
11
12              // First, create the UI
13              IssueBikeUI ui = new IssueBikeUI()
14
15              // 1. Show details for chosen bike
16              ui.showBikeDetails(100);
17
18              // 2. Calculate cost of hiring this bike for 5 days
19              ui.calculateCost(5);
20
21              // 3. Create new customer, payment and hire
22              ui.createCustomer("Les Hargreaves", "PW2 6TR", 01462501339);
23
24              // 4. Calculate the total cost
25              ui.calculateTotalPayment();
26          }
27      }
```

Table mapping sequence diagram messages to lines of code

Sequence diagram message	Code line	Sending object/class	Receiving object/class
1 constructor()	13	StartUp	IssueBikeUI
2 showBikeDetails(bikeNum)	16	StartUp	IssueBikeUI
2.1 findBikeByNumber(bikeNum)	44	IssueBikeUI	Bike
2.1.1 getBikeNumber()	116	Bike	Bike
2.2 showDetails()	47	IssueBikeUI	Bike
3 calculateCost(numberOfDays)	19	StartUp	IssueBikeUI
3.1 calculateCost(numberOfDays)	54	IssueBikeUI	Bike
4 createCustomer(name,postcode,tel)	22	StartUp	IssueBikeUI
4.1 constructor(String,String,int)	59	IssueBikeUI	Customer
4.2 constructor(Customer)	60	IssueBikeUI	Payment
5 constructor(Date,int,Bike,Customer)	61	IssueBikeUI	Hire
6 calculateTotalPayment()	25	StartUp	IssueBikeUI
6.1 calculateTotalPayment(Hire)	66	IssueBikeUI	Payment
6.1.1. issueReceipt(Hire)	237	Payment	Payment
6.1.1.1 getName()	242	Payment	Customer
6.1.1.2 getPostcode()	243	Payment	Customer
6.1.1.3 getBikeNumber()	247	Payment	Bike
6.1.1.4 getNumberOfdays()	248	Payment	Hire
6.1.1.5 calculateCost(int)	250	Payment	Bike

Implementation of a sequence diagram



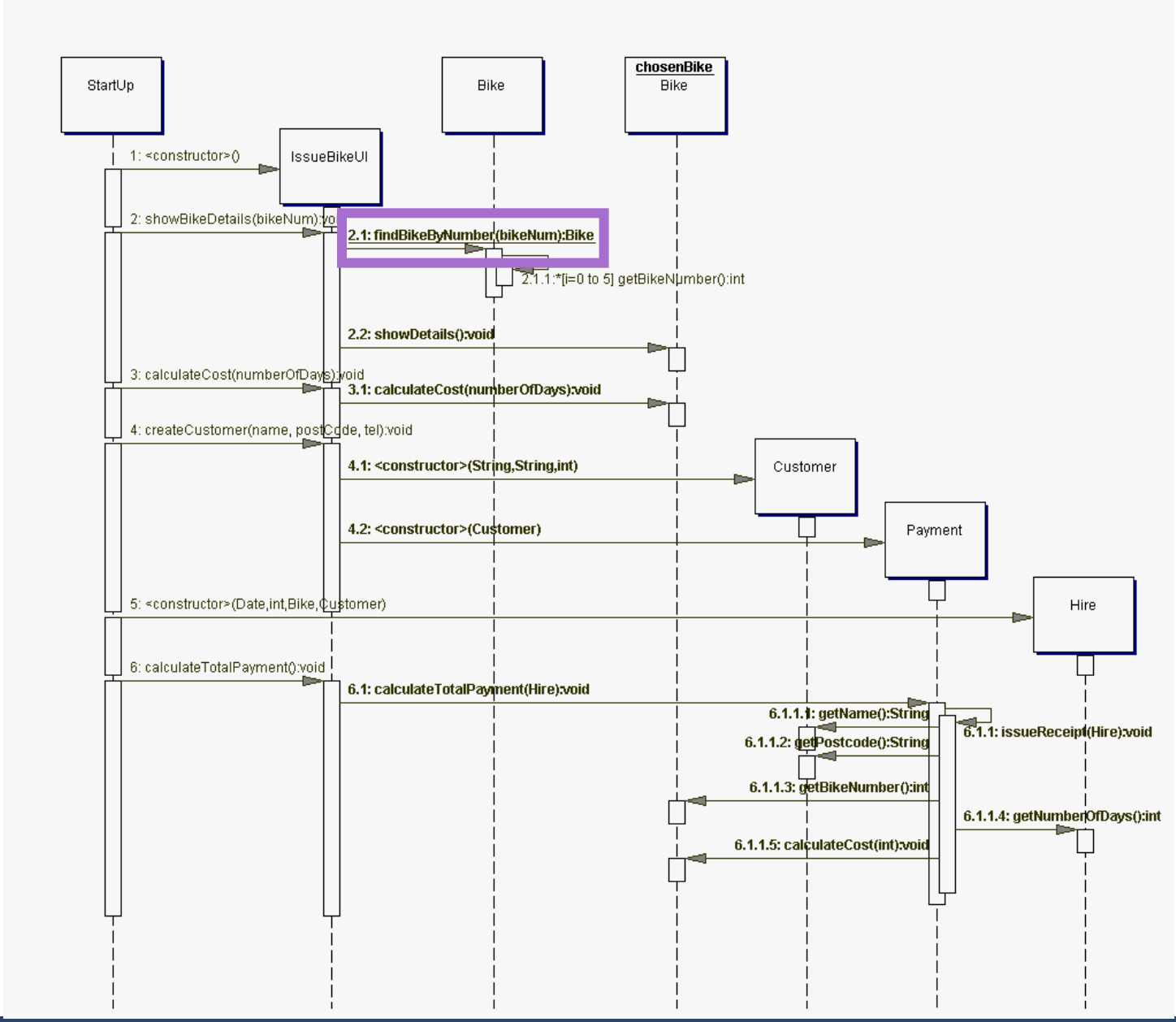
StartUp class

```
01      package bikeshop;
02
03      /* Generated by Together */
04
05      public class StartUp {
06
07          public static void main(String[] args){
08
09              /* This little program will run through the methods on IssueBikeUI
10               * calling each in turn, like a user with a front end would do. */
11
12              // First, create the UI
13              IssueBikeUI ui = new IssueBikeUI();
14
15              // 1. Show details for chosen bike
16              ui.showBikeDetails(100);
17
18              // 2. Calculate cost of hiring this bike for 5 days
19              ui.calculateCost(5);
20
21              // 3. Create new customer, payment and hire
22              ui.createCustomer("Les Hargreaves", "PW2 6TR", 01462501339);
23
24              // 4. Calculate the total cost
25              ui.calculateTotalPayment();
26          }
27      }
```

Table mapping sequence diagram messages to lines of code

Sequence diagram message	Code line	Sending object/class	Receiving object/class
1 constructor()	13	StartUp	IssueBikeUI
2 showBikeDetails(bikeNum)	16	StartUp	IssueBikeUI
2.1 findBikeByNumber(bikeNum)	44	IssueBikeUI	Bike
2.1.1 getBikeNumber()	116	Bike	Bike
2.2 showDetails()	47	IssueBikeUI	Bike
3 calculateCost(numberOfDays)	19	StartUp	IssueBikeUI
3.1 calculateCost(numberOfDays)	54	IssueBikeUI	Bike
4 createCustomer(name,postcode,tel)	22	StartUp	IssueBikeUI
4.1 constructor(String,String,int)	59	IssueBikeUI	Customer
4.2 constructor(Customer)	60	IssueBikeUI	Payment
5 constructor(Date,int,Bike,Customer)	61	IssueBikeUI	Hire
6 calculateTotalPayment()	25	StartUp	IssueBikeUI
6.1 calculateTotalPayment(Hire)	66	IssueBikeUI	Payment
6.1.1. issueReceipt(Hire)	237	Payment	Payment
6.1.1.1 getName()	242	Payment	Customer
6.1.1.2 getPostcode()	243	Payment	Customer
6.1.1.3 getBikeNumber()	247	Payment	Bike
6.1.1.4 getNumberOfdays()	248	Payment	Hire
6.1.1.5 calculateCost(int)	250	Payment	Bike

Implementation sequence diagram



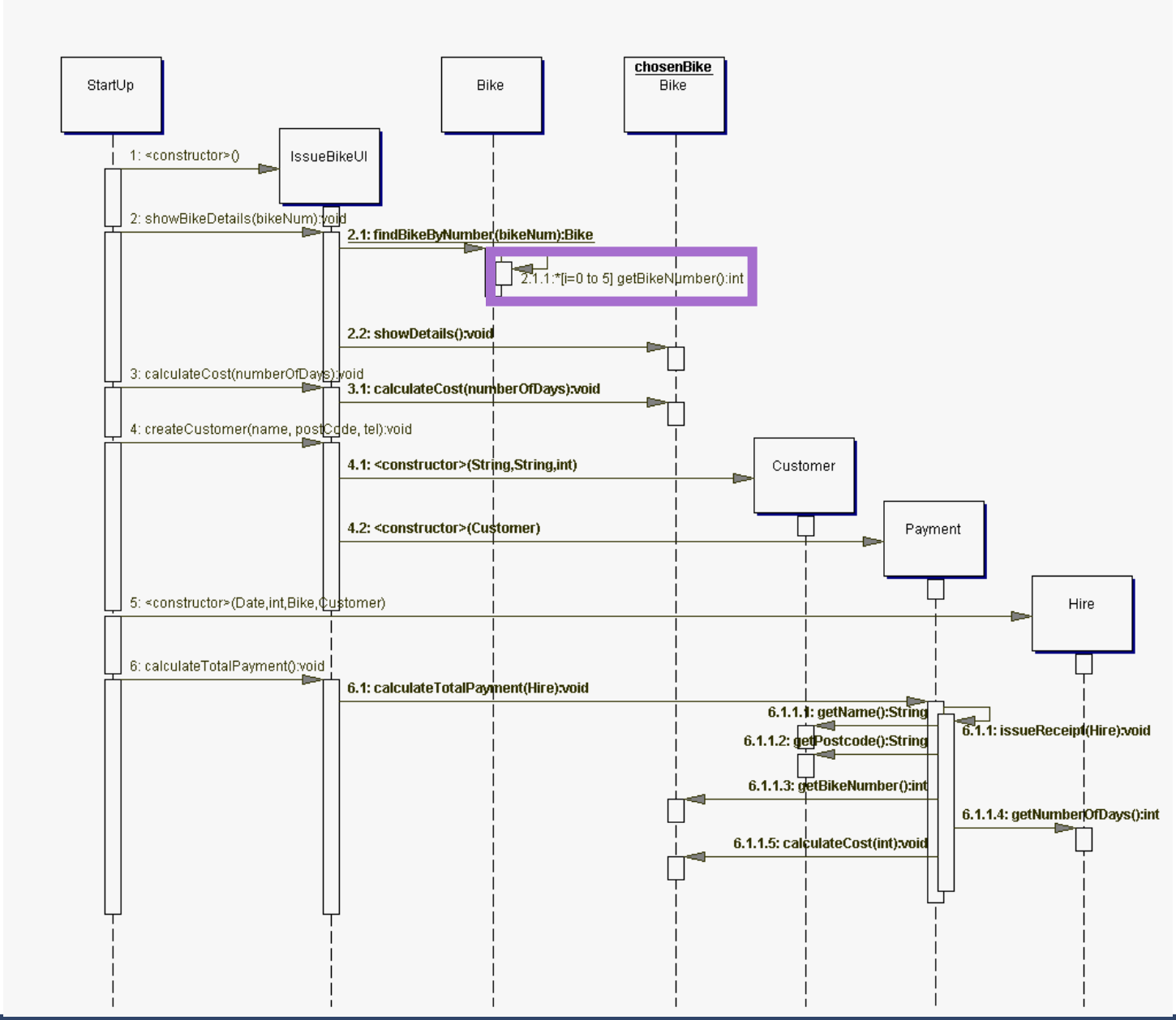
IssueBikeUI class

```
28      /*IssueBikeUI Class*/
29
30      package bikeshop;
31
32      import java.util.Date;
33
34      public class IssueBikeUI {
35
36          // Set up the member (or class-level variables)
37          private Bike chosenBike = null;
38          private Customer customer = null;
39          private Payment payment = null;
40          private Hire hire = null;
41          private int numberOfDays = 0;

42          public void showBikeDetails(int bikeNum){
43              // Find the bike by its number
44              chosenBike = Bike.findBikeByNumber(bikeNum);
45              if(chosenBike !=null){
46                  // then ask it for its details
47                  chosenBike.showDetails();
48              }
49          }
```

Table mapping sequence diagram messages to lines of code

Sequence diagram message	Code line	Sending object/class	Receiving object/class
1 constructor()	13	StartUp	IssueBikeUI
2 showBikeDetails(bikeNum)	16	StartUp	IssueBikeUI
2.1 findBikeByNumber(bikeNum)	44	IssueBikeUI	Bike
2.1.1 getBikeNumber()	116	Bike	Bike
2.2 showDetails()	47	IssueBikeUI	Bike
3 calculateCost(numberOfDays)	19	StartUp	IssueBikeUI
3.1 calculateCost(numberOfDays)	54	IssueBikeUI	Bike
4 createCustomer(name,postcode,tel)	22	StartUp	IssueBikeUI
4.1 constructor(String,String,int)	59	IssueBikeUI	Customer
4.2 constructor(Customer)	60	IssueBikeUI	Payment
5 constructor(Date,int,Bike,Customer)	61	IssueBikeUI	Hire
6 calculateTotalPayment()	25	StartUp	IssueBikeUI
6.1 calculateTotalPayment(Hire)	66	IssueBikeUI	Payment
6.1.1. issueReceipt(Hire)	237	Payment	Payment
6.1.1.1 getName()	242	Payment	Customer
6.1.1.2 getPostcode()	243	Payment	Customer
6.1.1.3 getBikeNumber()	247	Payment	Bike
6.1.1.4 getNumberOfdays()	248	Payment	Hire
6.1.1.5 calculateCost(int)	250	Payment	Bike

Implementation sequence diagram



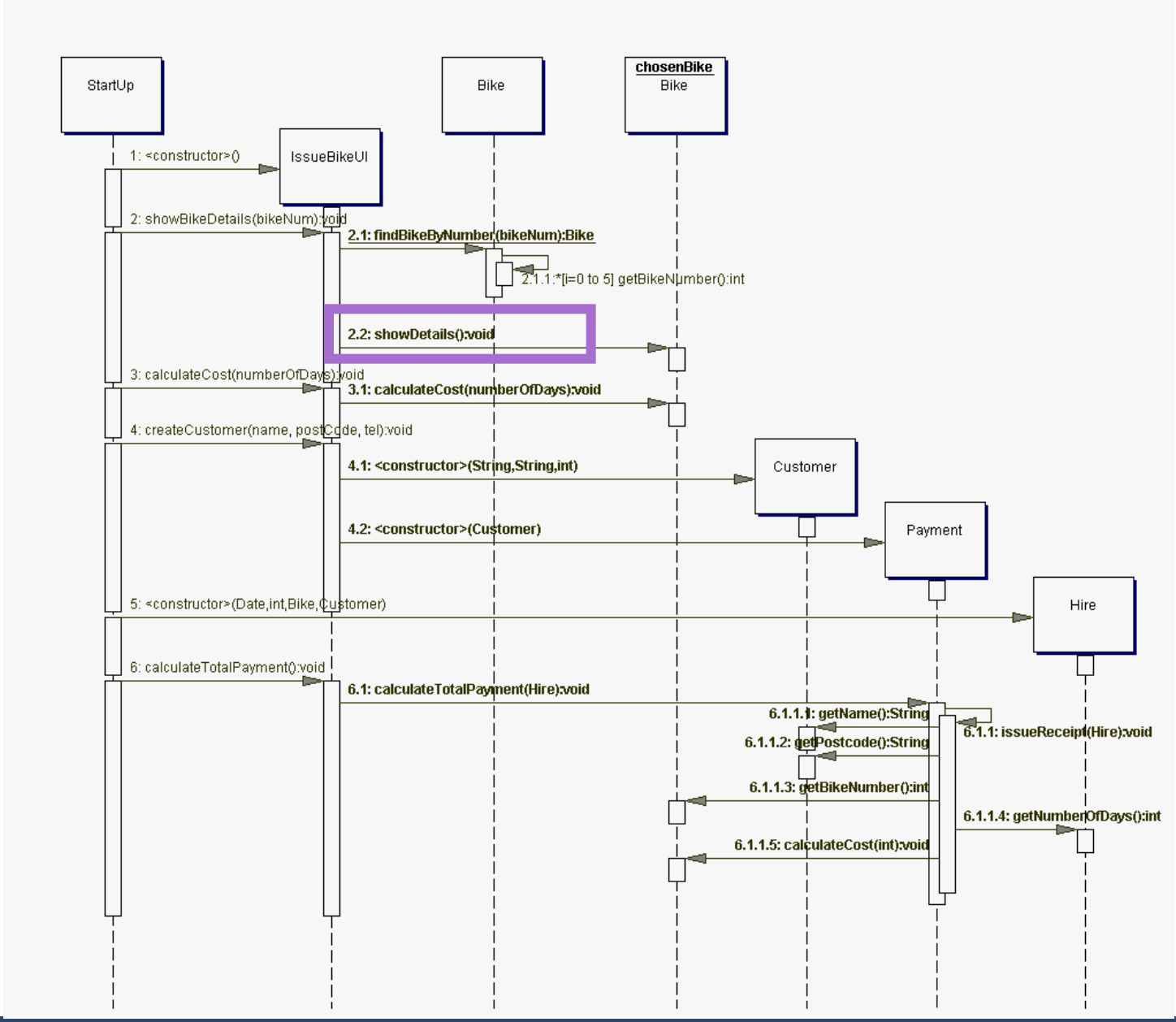
Bike class

```
71      public class Bike {
>
>
>
106         public int getBikeNumber(){
107             return bikeNumber;
108         }
109
110         public static Bike findBikeByNumber(int bikeNum){
111             int numberOfBikes = bikeList.length;
112
113             // iterate over the list of bikes
114             for(int i=0;i<numberOfBikes;i++){
115                 // if we find the bike with the correct number
116                 if(bikeList[i].getBikeNumber() == bikeNum){
117                     // tell user that we've found it
118                     System.out.println("Bike with number '" + bikeNum + "' found" + "\n");
119                     // and return it to the UI
120                     return bikeList[i];
121                 }
122     }
>
>
>
```

Table mapping sequence diagram messages to lines of code

Sequence diagram message	Code line	Sending object/class	Receiving object/class
1 constructor()	13	StartUp	IssueBikeUI
2 showBikeDetails(bikeNum)	16	StartUp	IssueBikeUI
2.1 findBikeByNumber(bikeNum)	44	IssueBikeUI	Bike
2.1.1 getBikeNumber()	116	Bike	Bike
2.2 showDetails()	47	IssueBikeUI	Bike
3 calculateCost(numberOfDays)	19	StartUp	IssueBikeUI
3.1 calculateCost(numberOfDays)	54	IssueBikeUI	Bike
4 createCustomer(name,postcode,tel)	22	StartUp	IssueBikeUI
4.1 constructor(String,String,int)	59	IssueBikeUI	Customer
4.2 constructor(Customer)	60	IssueBikeUI	Payment
5 constructor(Date,int,Bike,Customer)	61	IssueBikeUI	Hire
6 calculateTotalPayment()	25	StartUp	IssueBikeUI
6.1 calculateTotalPayment(Hire)	66	IssueBikeUI	Payment
6.1.1. issueReceipt(Hire)	237	Payment	Payment
6.1.1.1 getName()	242	Payment	Customer
6.1.1.2 getPostcode()	243	Payment	Customer
6.1.1.3 getBikeNumber()	247	Payment	Bike
6.1.1.4 getNumberOfdays()	248	Payment	Hire
6.1.1.5 calculateCost(int)	250	Payment	Bike

Implementation sequence diagram



IssueBikeUI class

```
public class IssueBikeUI {

    // Set up the member (or class-level variables)
    private Bike chosenBike = null;
    private Customer customer = null;
    private Payment payment = null;
    private Hire hire = null;
    private int numberOfDays = 0;

    public void showBikeDetails(int bikeNum){
        // Find the bike by its number
        chosenBike = Bike.findBikeByNumber(bikeNum);
        if(chosenBike !=null){
            // then ask it for its details
            chosenBike.showDetails();
        }
    }
```

return to here

Visual Paradigm

- Visual Paradigm provide feature to reverse engineering Sequence Diagram from Java Source Code.

- Refer to **Lab 10** in Visual Paradigm Guidelines.

Key points

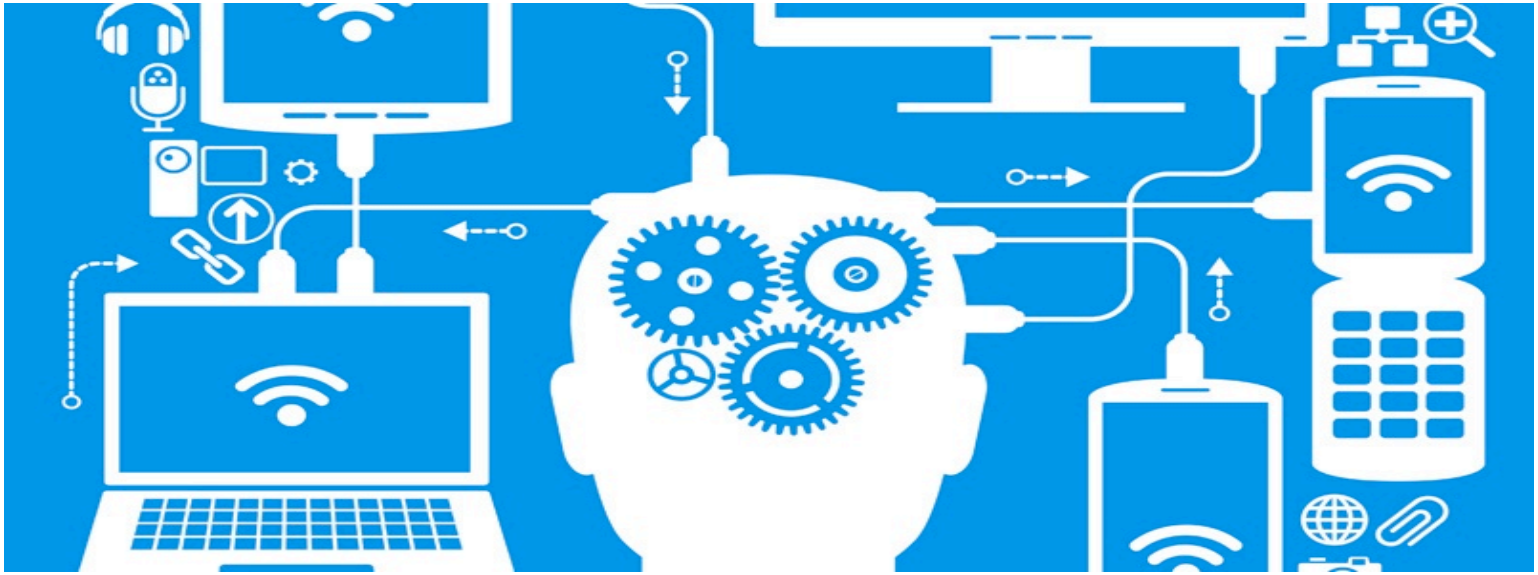
In this lecture you have learned about:

- Implement a class diagram in a relational database.
- Implement class and sequence diagrams in coding.

References

- A Student Guide to Object-Oriented Development (Chapters 9, 10 and 11)

In the next lecture..



Lecture 12: Software Testing