26. Object-Oriented Design

Java

Summer 2008

Instructor: Dr. Masoud Yaghini

Object-Oriented Design

- In the preceding chapters you learned the concepts of object-oriented programming, such as objects, classes, class inheritance, and polymorphism.
- This chapter focuses on the development of software systems using the object-oriented approach, and introduces class modeling using the Unified Modeling Language (UML).
- You will learn class-design guidelines.

Outline

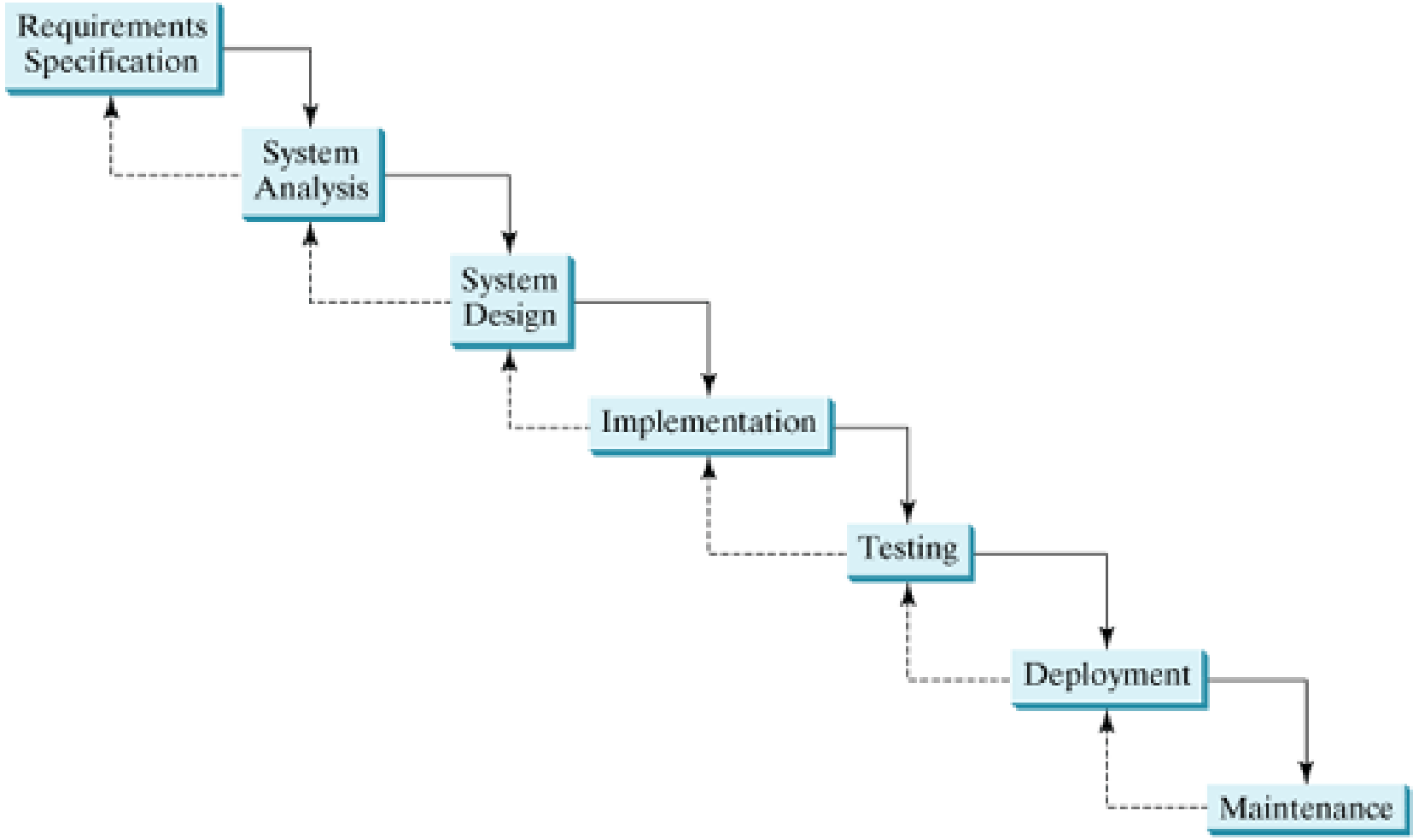
- The Software Development Process
- Discovering Class Relationships
- Case Study: Borrowing Loans
- References

The Software Development Process

The Software Development Process

- Developing a software project is an engineering process.
- Software products, no matter how large or how small, have the same developmental phases:
  - requirements specification,
  - analysis,
  - design,
  - implementation,
  - testing,
  - deployment, and
  - maintenance

The Software Development Process



Requirements specification

- **Requirements specification** is a formal process that seeks to understand the problem and document in detail what the software system needs to do.
- This phase involves close interaction between users and developers.
- In the real world problems are not well defined.
- You need to work closely with your customer and study a problem carefully to identify its requirements.

System analysis

- **System analysis** seeks to analyze the business process in terms of data flow, and to identify the system's input and output.
- Part of the analysis entails modeling the system's behavior.
- The model is intended to capture the essential elements of the system and to define services to the system.

System design

- **System design** is the process of designing the system's components.
- This phase involves the use of many levels of abstraction to decompose the problem into manageable components, identify classes and interfaces, and establish relationships among the classes and interfaces.

Implementation

- **Implementation** is translating the system design into programs.
- Separate programs are written for each component and put to work together.
- This phase requires the use of a programming language like Java.
- The implementation involves coding, testing, and debugging.

Testing

- **Testing** ensures that the code meets the requirements specification and weeds out bugs.
- An independent team of software engineers not involved in the design and implementation of the project usually conducts such testing.

Deployment

- Deployment makes the project available for use.
- For a Java applet, this means installing it on a Web server; for a Java application, installing it on the client's computer.
- A project usually consists of many classes.
- An effective approach for deployment is to package all the classes into a Java archive file.

Maintenance

- **Maintenance** is concerned with changing and improving the product.
- A software product must continue to perform and improve in a changing environment.
- This requires periodic upgrades of the product to fix newly discovered bugs and incorporate changes.

Object-Oriented Design

- This chapter is mainly concerned with object-oriented design.
- While there are many object-oriented methodologies, UML has become the industry-standard notation for object-oriented modeling.
- The process of designing classes calls for identifying the classes and discovering the relationships among them.

Discovering Class Relationships

Discovering Class Relationships

- The relationships among classes :
  - Association
  - Aggregation
  - Composition
  - Inheritance

Association

- **Association** is a general binary relationship that describes an activity between two classes.
- For example,
  - a student taking a course is an association between the Student class and the Course class
  - a faculty member teaching a course is an association between the Faculty class and the Course class

Association

- An association is illustrated by a solid line between two classes with an optional label that describes the relationship.
- The labels are Take and Teach.
- Each relationship may have an optional small black triangle that indicates the direction of the relationship.
- The direction indicates that a student takes a course.

Association

- Each class involved in the relationship may have a role name that describes the role it plays in the relationship.
- Teacher is the role name for Faculty.

Association

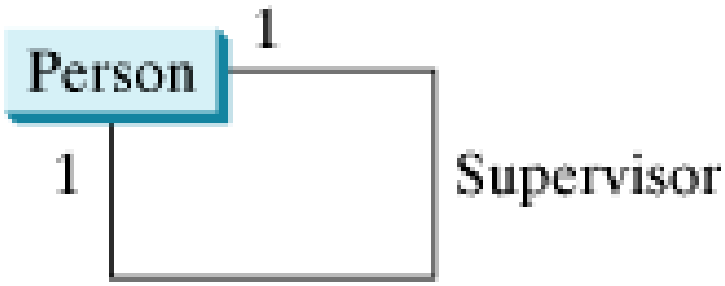
- Each class involved in an association may specify a multiplicity.
- A multiplicity could be a number or an interval that specifies how many objects of the class are involved in the relationship.
- The character * means unlimited number of objects, and the interval m..n means that the number of objects should be between m and n, inclusive.

Association

- Each student may take any number of courses
- Each course must have at least five students and at most sixty students
- Each course is taught by only one faculty member
- A faculty member may teach from zero to three courses per semester

Association Between Same Class

- Association may exist between objects of the same class.
- For example, a person may have a supervisor.



Association

- An association can be implemented using data fields.
- The method in one class contains a parameter of the other class.


```
public class Student {
  private Course[]
    courseList;

  public void addCourse(
    Course s)
}
```

```
public class Course {
  private Student[]
    classList;
  private Faculty faculty;

  public void addStudent(
    Student s)

  public void setFaculty(
    Faculty faculty)
}
```

```
public class Faculty {
  private Course[]
    courseList;

  public void addCourse(
    Course c)
}
```

Aggregation & Composition

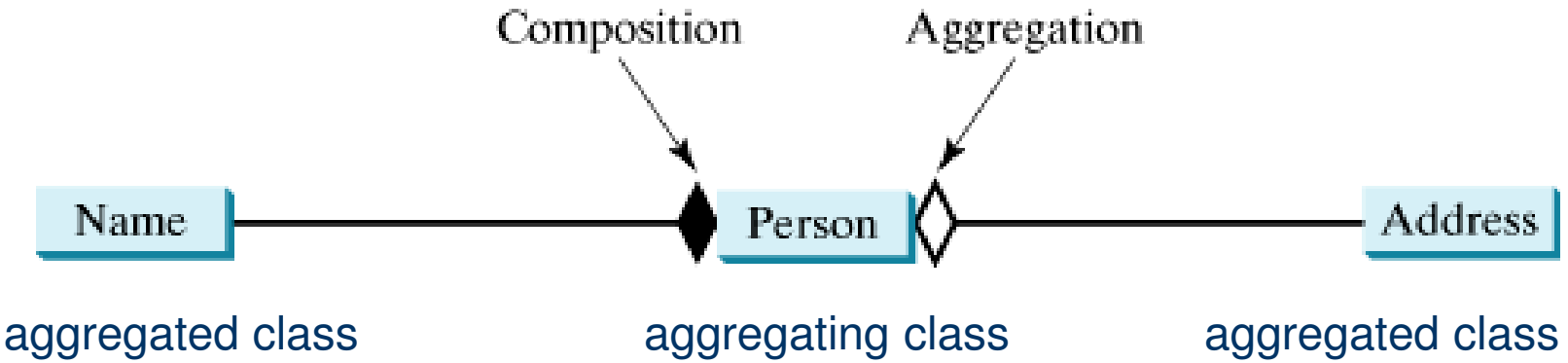
- ***Aggregation*** is a special form of association that represents an ownership relationship between two objects.
- Aggregation models has-a relationships.
- The owner object is called an ***aggregating object***, and its class, an ***aggregating class***.
- The subject object is called an ***aggregated object***, and its class, an ***aggregated class***.

Aggregation & Composition

- If an object is exclusively owned by an aggregating object, the relationship between the object and its aggregating object is referred to as ***composition***.
- For example, "a student has a name" is a composition relationship between the Student class and the Name class
- Whereas "a student has an address" is an aggregation relationship between the Student class and the Address class, since an address may be shared by several students.

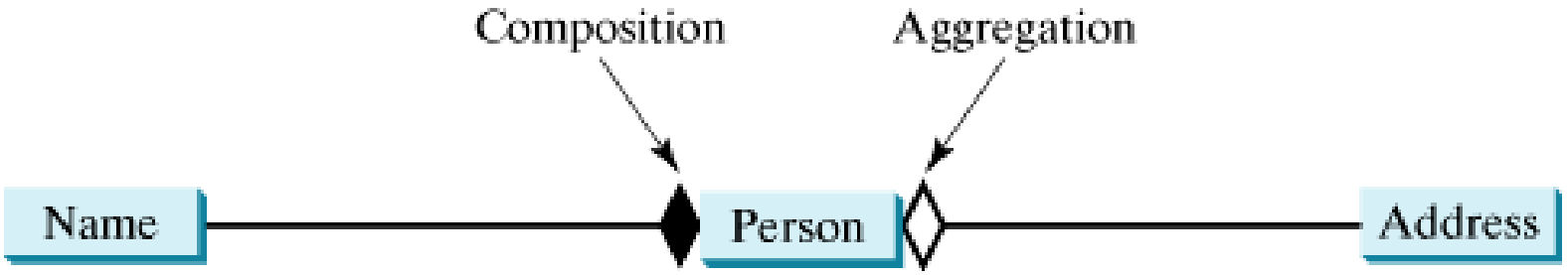
Aggregation & Composition

- In UML, a filled diamond is attached to an aggregating class (e.g., Student) to denote the composition relationship with an aggregated class (e.g., Name)
- An empty diamond is attached to an aggregating class (e.g., Student) to denote the aggregation relationship with an aggregated class (e.g., Address)



Aggregation & Composition

- An aggregation relationship is usually represented as a data field in the aggregating class.




```
public class Name {
  ...
}
```

Aggregated class

```
public class Person {
  private Name name;
  private Address address;

  ...
}
```

Aggregating class

```
public class Address {
  ...
}
```

Aggregated class

Inheritance

- ***Inheritance*** models the *is-an-extension-of* relationship between two classes.

(a)

```
public class Faculty extends Person {
  ...
}
```

(b)

Case Study: Borrowing Loans

Case Study: Borrowing Loans

- This case study models borrowing loans to demonstrate:
  - how to identify classes,
  - discover the relationships between classes, and
  - apply class abstraction in object-oriented program development.

- For simplicity, it focuses on modeling borrowers and the loans for the borrowers.

Case Study: Borrowing Loans

- The following steps are usually involved in building an object-oriented system:

1. Identify classes for the system.
2. Establish relationships among classes.
3. Describe the attributes and methods in each class.
4. Implement the classes.

Identify classes for the system

- There are many strategies for identifying classes in a system, one of which is to study how the system works and select a number of use cases, or scenarios.
- Since a borrower is a person who obtains a loan, and a person has a name and an address, you can identify the following classes:
  - Person
  - Name
  - Address
  - Borrower
  - Loan

Identify classes for the system

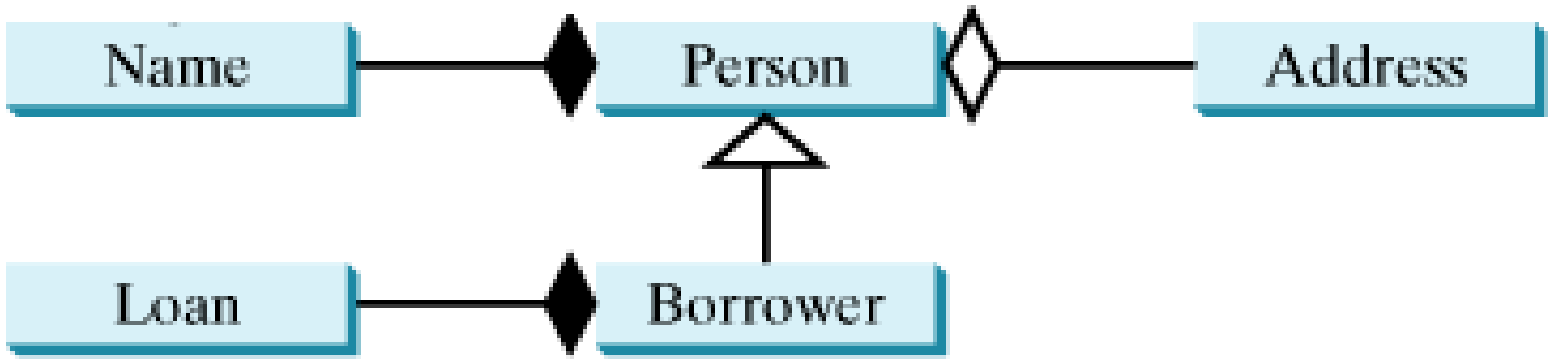
- There is no unique solution to find classes even for simple problems.
- Software development is more an art than a science.
- The quality of a program ultimately depends on the programmer's experience, and knowledge.

Establish relationships among classes

- The second step is to establish relationships among the classes.
- The relationship is derived from the system analysis.
- When you identify classes, you also think about the relationships among them.
- Establishing relationships among objects helps you understand the interactions among objects.
- An object-oriented system consists of a collection of interrelated cooperative objects.

Establish relationships among classes

- Relationships for the classes in this example



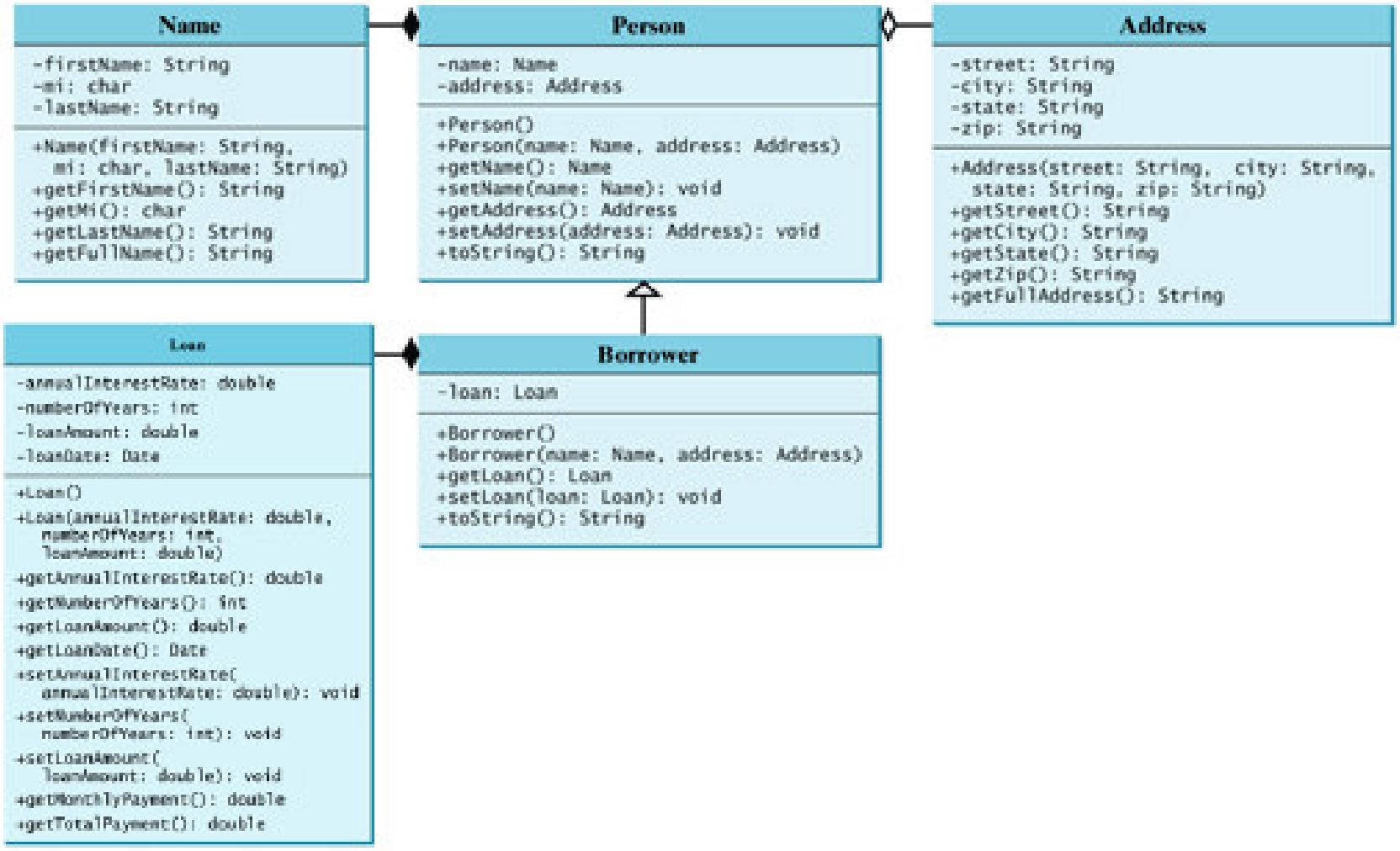
Describe the attributes and methods

- The third step is to describe the attributes and methods in each of the classes you have identified.
- The Name class has:
  - the properties firstName, mi, and lastName,
  - their associated get and set methods, and the getFullName method for returning the full name.
- The Address class has:
  - the properties street, city, state, and zip,
  - their associated get and set methods, and the getAddress method for returning the full address.

Describe the attributes and methods

- The Loan class has:
  - the properties annualInterestRate, numberOfYears, and loanAmount,
  - their associated property get and set methods, and getMonthlyPayment and getTotalPayment methods.
- The Person class has:
  - the properties name and address,
  - their associated get and set methods, and the toString method for displaying complete information about the person.
- Borrower is a subclass of Person. Additionally, Borrower has:
  - the loan property and its associated get and set methods, and the toString method for displaying the person and the loan payments.

Describe the attributes and methods



Write the code for the classes

- The fourth step is to write the code for the classes.

```
 1  package chapter11;
 2
 3  public final class Name {
 4    private String firstName;
 5    private char mi;
 6    private String lastName;
 7
 8    /** Construct a name with firstName, mi, and lastName */
 9    public Name(String firstName, char mi, String lastName) {
10      this.firstName = firstName;
11      this.mi = mi;
12      this.lastName = lastName;
13    }
14
15    /** Return firstName */
16    public String getFirstName() {
17      return firstName;
18    }
19
20    /** Return middle name initial */
21    public char getMi() {
22      return mi;
23    }
```

```
24
25    /** Return lastName */
26    public String getLastname() {
27      return lastName;
28    }
29
30    /** Obtain full name */
31    public String getFullName() {
32      return firstName + ' ' + mi + ' ' + lastName;
33    }
34
35    /** Implement compareTo in the Comparable interface */
36    public int compareTo(Object o) {
37      if (!lastName.equals(((Name)o).lastName)) {
38        return lastName.compareTo(((Name)o).lastName);
39      }
40      else if (!firstName.equals(((Name)o).firstName)) {
41        return firstName.compareTo(((Name)o).firstName);
42      }
43      else {
44        return mi - ((Name)o).mi;
45      }
46    }
47  }
```

```
 1  package chapter11;
 2
 3  public final class Address {
 4    private String street;
 5    private String city;
 6    private String state;
 7    private String zip;
 8
 9    /** Create an address with street, city, state, and zip */
10    public Address(String street, String city,
11                   String state, String zip) {
12      this.street = street;
13      this.city = city;
14      this.state = state;
15      this.zip = zip;
16    }
17
18    /** Return street */
19    public String getStreet() {
20      return street;
21    }
```

```
22
23      /** Return city */
24      public String getCity() {
25          return city;
26      }
27
28      /** Return state */
29      public String getState() {
30          return state;
31      }
32
33      /** Return zip */
34      public String getZip() {
35          return zip;
36      }
37
38      /** Get full address */
39      public String getFullAddress() {
40          return street + '\n' + city + ", " + state + ' ' + zip + '\n';
41      }
42  }
```

```
1  package chapter11;
2
3  public class Person {
4    private Name name;
5    private Address address;
6
7    /** Construct a person with default properties */
8    public Person() {
9      this(new Name("Jill", 'S', "Barr"),
10           new Address("100 Main", "Savannah", "GA", "31411"));
11   }
12
13   /** Construct a person with specified name and address */
14   public Person(Name name, Address address) {
15     this.name = name;
16     this.address = address;
17   }
18
19   /** Return name */
20   public Name getName() {
21     return name;
22   }
23
24   /** Set a new name */
25   public void setName(Name name) {
26     this.name = name;
27   }
```

```
28
29    /** Return address */
30    public Address getAddress() {
31      return address;
32    }
33
34    /** Set a new address */
35    public void setAddress(Address address) {
36      this.address = address;
37    }
38
39    /** Override the toString method */
40    public String toString() {
41      return '\n' + name.getFullName() + '\n' +
42            address.getFullAddress() + '\n';
43    }
44
45    /** Implement compareTo in the Comparable interface */
46    public int compareTo(Object o) {
47      return name.compareTo(((Person)o).name);
48    }
49  }
```

```
 1  package chapter11;
 2
 3  public class Borrower extends Person {
 4    private Loan loan;
 5
 6    /** Construct a borrower with default properties */
 7    public Borrower() {
 8      super();
 9    }
10
11    /** Create a borrower with specified name and address */
12    public Borrower(Name name, Address address) {
13      super(name, address);
14    }
15
```

```
16      /** Return loan */
17      public Loan getLoan() {
18        return loan;
19      }
20
21      /** Set a new loan */
22      public void setLoan(Loan loan) {
23        this.loan = loan;
24      }
25
26      /** String representation for borrower */
27      public String toString() {
28        return super.toString() +
29              "Monthly payment is " + loan.getMonthlyPayment() + '\n' +
30              "Total payment is " + loan.getTotalPayment();
31      }
32  }
```

```
1  package chapter11;
2
3  public class Loan {
4    private double annualInterestRate;
5    private int numberOfYears;
6    private double loanAmount;
7    private java.util.Date loanDate;
8
9    /** Default constructor */
10   public Loan() {
11     this(7.5, 30, 100000);
12   }
13
14   public Loan(double annualInterestRate, int numberOfYears,
15               double loanAmount) {
16     this.annualInterestRate = annualInterestRate;
17     this.numberOfYears = numberOfYears;
18     this.loanAmount = loanAmount;
19     loanDate = new java.util.Date();
20   }
21
```

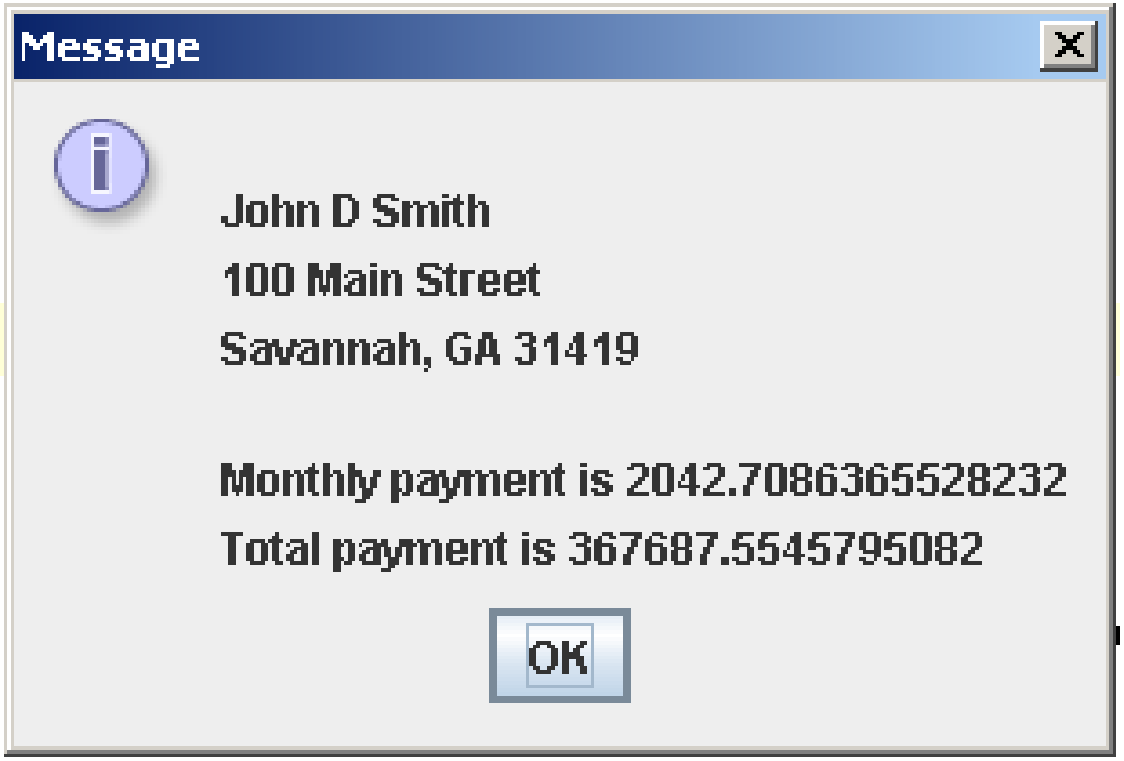
```
22    /** Return annualInterestRate */
23    public double getAnnualInterestRate() {
24      return annualInterestRate;
25    }
26
27    /** Set a new annualInterestRate */
28    public void setAnnualInterestRate(double annualInterestRate) {
29      this.annualInterestRate = annualInterestRate;
30    }
31
32    /** Return numberOfYears */
33    public int getNumberOfYears() {
34      return numberOfYears;
35    }
36
37    /** Set a new numberOfYears */
38    public void setNumberOfYears(int numberOfYears) {
39      this.numberOfYears = numberOfYears;
40    }
41
42    /** Return loanAmount */
43    public double getLoanAmount() {
44      return loanAmount;
45    }
```

```
46
47    /** Set a newloanAmount */
48    public void setLoanAmount(double loanAmount) {
49      this.loanAmount = loanAmount;
50    }
51
52    /** Find monthly payment */
53    public double getMonthlyPayment() {
54      double monthlyInterestRate = annualInterestRate / 1200;
55      return loanAmount * monthlyInterestRate / (1 -
56          (Math.pow(1 / (1 + monthlyInterestRate), numberOfYears * 12)));
57    }
58
59    /** Find total payment */
60    public double getTotalPayment() {
61      return getMonthlyPayment() * numberOfYears * 12;
62    }
63
64    /** Return loan date */
65    public java.util.Date getLoanDate() {
66      return loanDate;
67    }
68  }
```

```
1  package chapter11;
2
3  import javax.swing.JOptionPane;
4
5  public class BorrowLoan {
6    /** Main method */
7    public static void main(String[] args) {
8      // Create a name
9      Name name = new Name("John", 'D', "Smith");
10
11     // Create an address
12     Address address = new Address("100 Main Street", "Savannah",
13         "GA", "31419");
14
15     // Create a loan
16     Loan loan = new Loan(5.5, 15, 250000);
17
18     // Create a borrower
19     Borrower borrower = new Borrower(name, address);
20
21     borrower.setLoan(loan);
22
23     // Display loan information
24     JOptionPane.showMessageDialog(null, borrower.toString());
25   }
26 }
```

Write the code for the classes

- The program creates name, address, and loan, stores the information in a Borrower object, and displays the information with the loan payment.



References

References

- Y. Daniel Liang, **Introduction to Java Programming**, Sixth Edition, Pearson Education, 2007. (Chapter 11)

The End