27. Object-Oriented Design

Java

Fall 2009

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

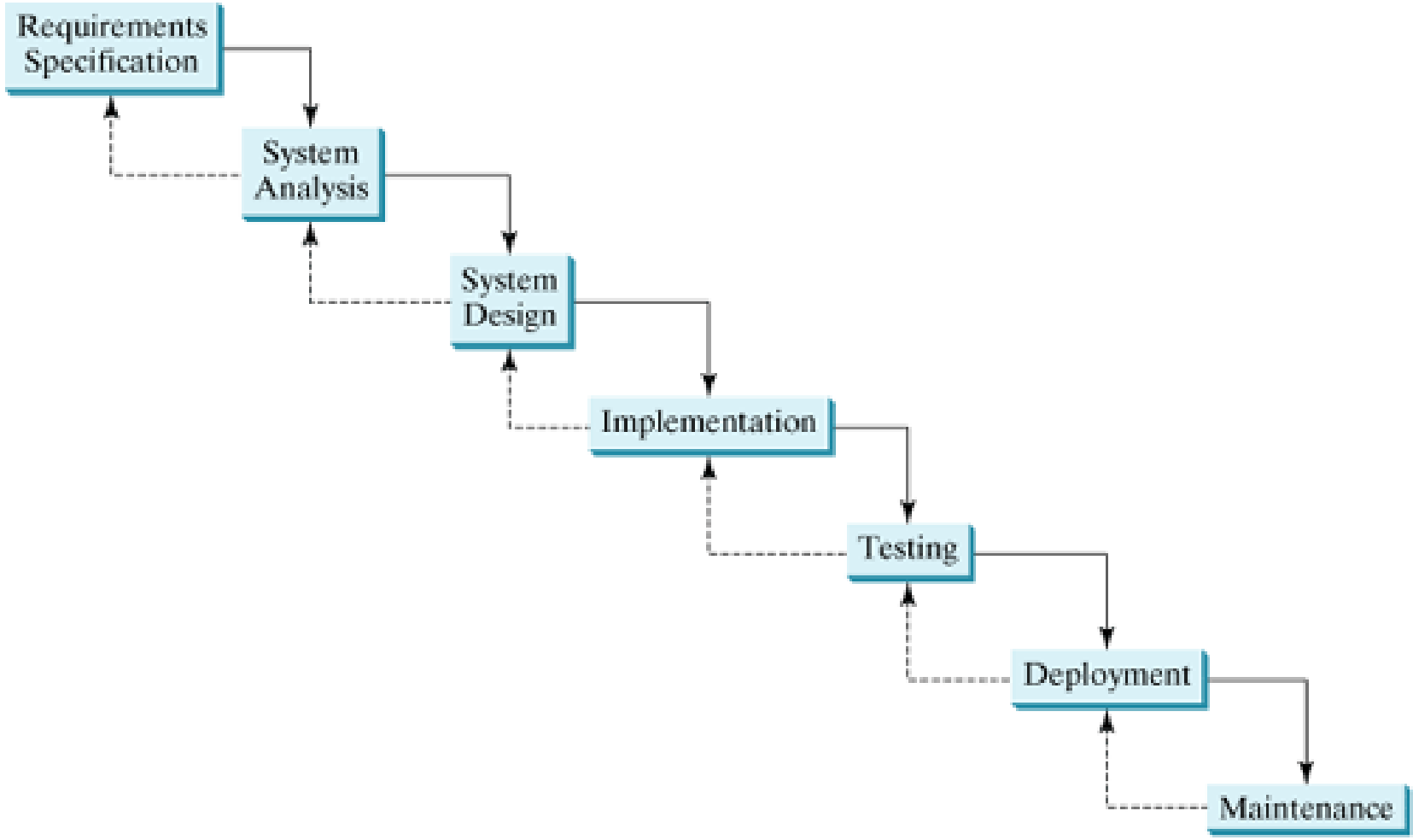
- The Software Development Process
- Discovering Class Relationships
- Case Study: Borrowing Loans
- References

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
  - 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

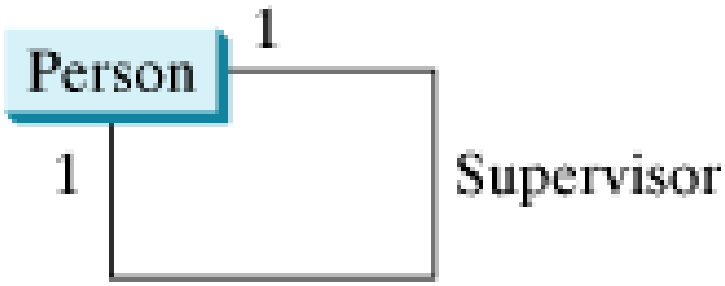
- Each class involved in an association may specify a multiplicity.
- A multiplicity could be a number or an interval that specifies how many objects of the class are involved in the relationship.
- The character * means unlimited number of objects, and the interval **m..n** means that the number of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

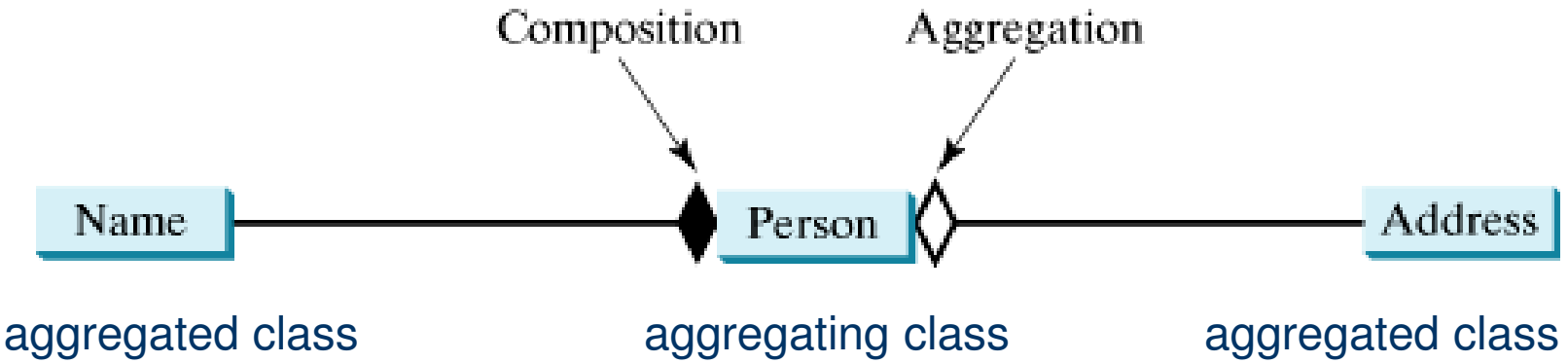
- **Aggregation** is a special form of association that represents an ownership relationship between two objects.
- Aggregation models has-a relationships.
- The owner object is called an **aggregating object**, and its class, an **aggregating class**.
- The subject object is called an **aggregated object**,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

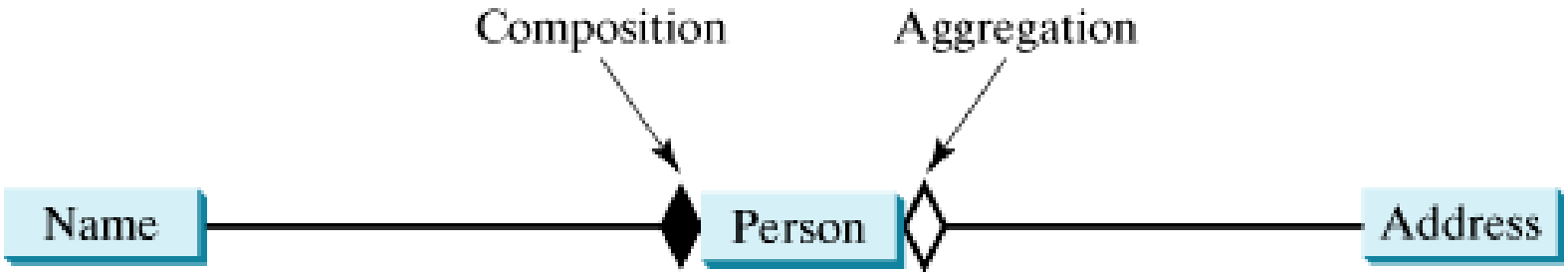
Aggregation & Composition

- 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

- 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

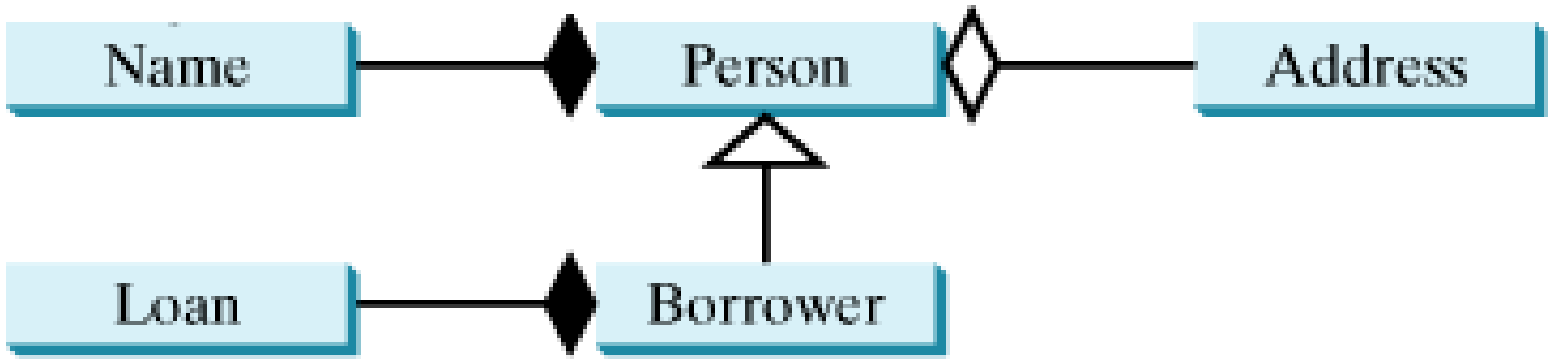
- There is no unique solution to find classes even for simple problems.
- Software development is more an art than a science.
- The quality of a program ultimately depends on the programmer's experience, and knowledge.

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



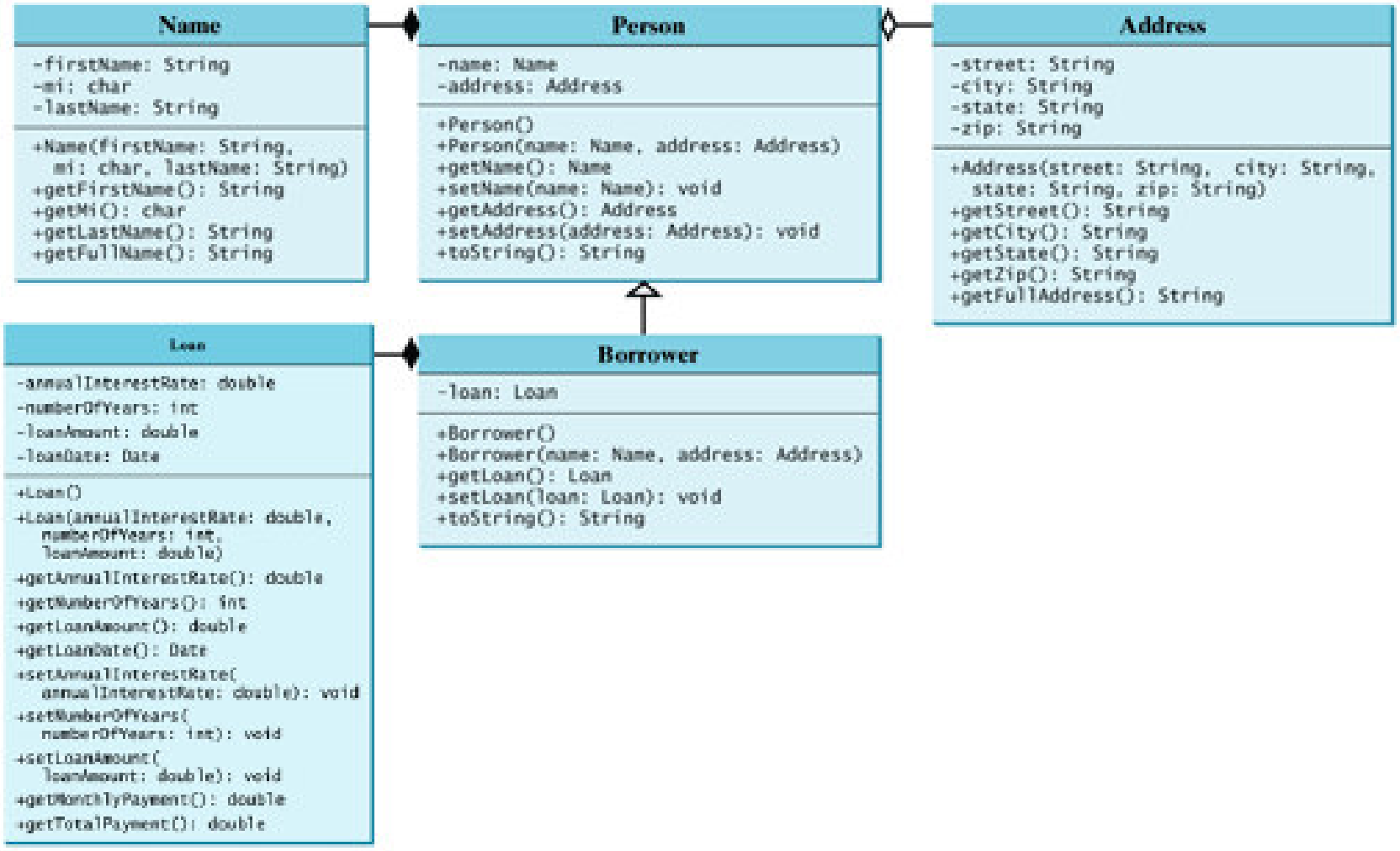
Describe the attributes and methods

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

Describe the attributes and methods

- The Loan class has:
  - Fields: annualInterestRate, numberOfYears, loanAmount,
  - Methods: get, set, getMonthlyPayment , getTotalPayment
- The Person class has:
  - Fields: name, address
  - Methods: get, set, toString method for displaying complete information about the person.
- Borrower is a subclass of Person.
  - Fields: loan
  - Methods: get, set, toString method for displaying the person and the loan payments.

Describe the attributes and methods

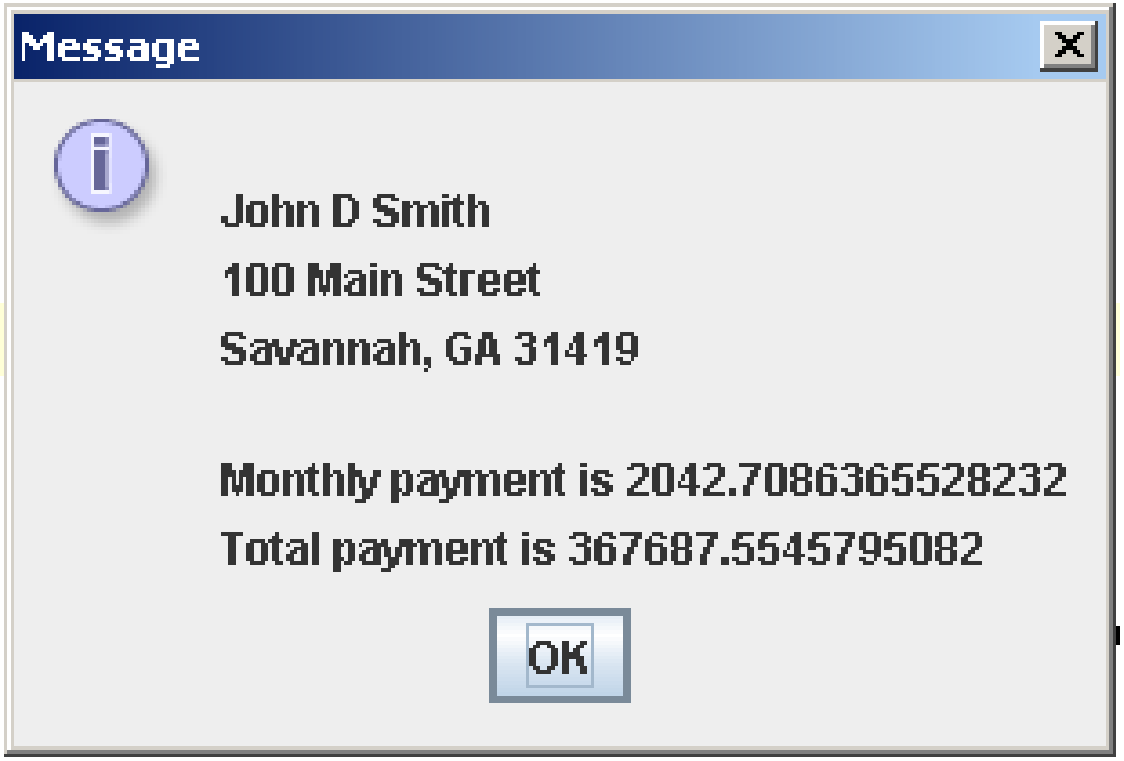


Write the code for the classes

- The fourth step is to write the code for the classes.
- The program:
  - Name.java
  - Address.java
  - Person.java
  - Borrower.java
  - Loan.java
  - BorrowLoan.java

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