Lab 4: Object Oriented Programming - JAVA

CLO: 1

The pillars of OOP are:

- **Encapsulation**: Using Classes, combining data and operations on the data a process called encapsulation.
- **Inheritance**: Allows us to extend the definition of a class without making any physical changes to the existing class.
- **Polymorphism**: polymorphism means associating multiple (potential) meanings with the same method name. Polymorphic reference variables can refer to objects of their own class or to objects of the subclasses inherited from their class. **Overloading** & **Overriding**
- **Abstraction**: The process of hiding the implementation details and showing only functionality to the user. **Abstract class** & **Interface**

Inheritance

Inheritance lets you create new classes from existing classes. Any new class that you create from an existing class is called a subclass or derived class; existing classes are called superclasses or base classes. In Java, you can relate two or more classes in more than one way. Two common ways to relate classes:

- Inheritance (‘‘is-a’’ relationship)
- Composition (aggregation) (‘‘has-a’’ relationship)

The general syntax to derive a class from an existing class is:

```
modifier(s) class ClassName extends ExistingClassName modifier(s)
{
    memberList
}
```

```
public class Circle extends Shape
{
    .
    .
    .
}
```

Modifiers

If a member of a class is private, you cannot access it outside the class. The subclass cannot access the private members of the superclass directly.

If a member of a class is public, you can access it outside the class.

If a member of a superclass needs to be accessed directly (only) by a subclass, that member is declared using the modifier protected.

If a data member of a class is declared using the modifier static, it can be accessed by using the name of the class. When you instantiate the objects, only the non-static data members of the class become the data members of each object. For each static data member of the class, Java allocates memory space only once. All objects of the class refer to the same memory space. In fact, static data members of a class exist even when no object of the class type is instantiated.

Methods

Methods are used to divide complicated programs into manageable pieces. There are both **predefined methods**, methods that are already written and provided by Java, and **user-defined methods**, methods that you create.

In Java, predefined methods are organized as a collection of classes, called class **libraries**. For example, the class Math contains mathematical methods. The class Math is contained in the package java.lang. Java also provides methods, contained in the class Character, to manipulate characters in the package java.lang.

User-defined methods in Java are classified into two categories:

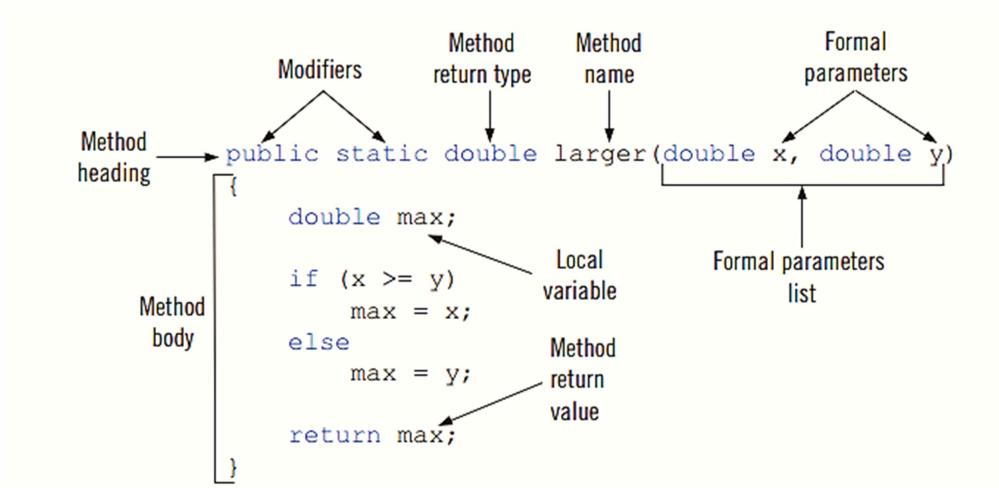
- **Value-returning methods**—methods that have a return data type. These methods return a value of a specific data type using the return statement.

  SYNTAX: VALUE-RETURNING METHOD

  The syntax of a value-returning method is:

```
modifier(s) returnType methodName(formal parameter list)
{
    statements
}
```

  Modifiers are: public, private, protected, static, abstract, and final.



- **Void methods**—methods that do not have a return data type. These methods do not use a return statement to return a value.

  The definition of a void method with parameters has the following syntax:

```
modifier(s) void methodName(formal parameter list)
{
     statements
}
```

Note: Java does not allow the nesting of methods. That is, you cannot include the definition of one method in the body of another method

The (non-static) data members—variables declared without using the modifier static—of a class are called instance variables.

Mutator methods: The methods that change the values of the instance variables.

Accessor methods: The method that only accesses the value of an instance variable

Constructors: Special types of methods. A constructor has the same name as the class, and it executes automatically when an object of that class is created. Constructors are used to guarantee that the instance variables of the class are initialized.

- Types of constructors
  1. Parameterized Constructors
  2. Without parameters – default constructor.

A call to a constructor of the superclass is specified in the definition of a subclass constructor by using the reserved word super.

```
super(parameters);
```

```
public Box()
{
    super();
    height = 0;
}

public Box(double l, double w, double h)
{
    super(l, w);
    height = h;
}
```

Polymorphism

Method overloading: Creating several methods, within a class, with the same name.

```
public void methodXYZ()
public void methodXYZ(int x, double y)
public void methodXYZ(double one, int y)
public void methodXYZ(int x, double y, char ch)
```

Method overriding: Redefine, the public methods of the superclass. In the subclass, you can have a method with the same name, number, and types of parameters as a method in the superclass. However, this redefinition is available only to the objects of the subclass, not to the objects of the superclass.

If the subclass overrides a public method of the superclass, then you must specify a call to that public method of the superclass by using the reserved word super, followed by the dot operator, followed by the method name with an appropriate parameter list. In this case, the general syntax to call a method of the superclass is:

If the subclass does not override a public method of the superclass, you can specify a call to that public method by using the name of the method and an appropriate parameter list.

```
super.methodName(parameters);
```

```
public void setDimension(double l, double w, double h)
{
    super.setDimension(l, w);

    if (h >= 0)
        height = h;
    else
        height = 0;
}
```

Abstract Class

An abstract class is a class that is declared with the reserved word abstract in its heading.

- An abstract class can contain instance variables, constructors, the finalizer, and nonabstract methods.
- An abstract class can contain an abstract method(s).
- If a class contains an abstract method, then the class must be declared abstract.
- You cannot instantiate an object of an abstract class. You can only declare a reference variable of an abstract class type.
- You can instantiate an object of a subclass of an abstract class, but only if the subclass gives the definitions of *all* the abstract methods of the superclass.

```
public abstract class AbstractClassExample
{
    protected int x;

    public abstract void print();

    public void setX(int a)
    {
        x = a;
    }


      public AbstractClassExample()
      {
          x = 0;
      }
  }
```

Interface

An **interface in Java** is a blueprint of a class. It has static constants and abstract methods. It cannot be instantiated just like the abstract class.

```
interface <interface_name>{

   // declare constant fields
   // declare methods that abstract
   // by default.
}
```

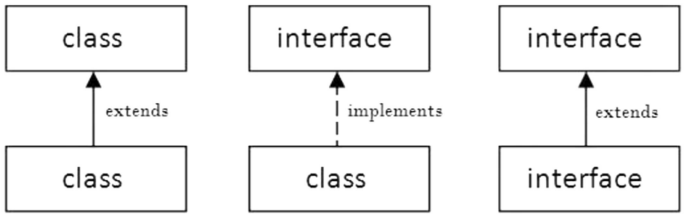
```
public interface WindowListener
{
    public void windowOpened(WindowEvent e);
    public void windowClosing(WindowEvent e);
    public void windowClosed(WindowEvent e);
    public void windowIconified(WindowEvent e);

    public void windowDeiconified(WindowEvent e);
    public void windowActivated(WindowEvent e);
    public void windowDeactivated(WindowEvent e);
}
```

```
public class ExampleInterfaceImp implements ActionListener,
                                            WindowListener
{
    //....
}
```

The relationship between classes and interfaces

As shown in the figure given below, a class extends another class, an interface extends another interface, but a **class implements an interface**.



Task:

CLO: 1, 2

Implement OOP concepts for one of the following systems. Assignment Selection Criteria: (Your roll number % 6)+1

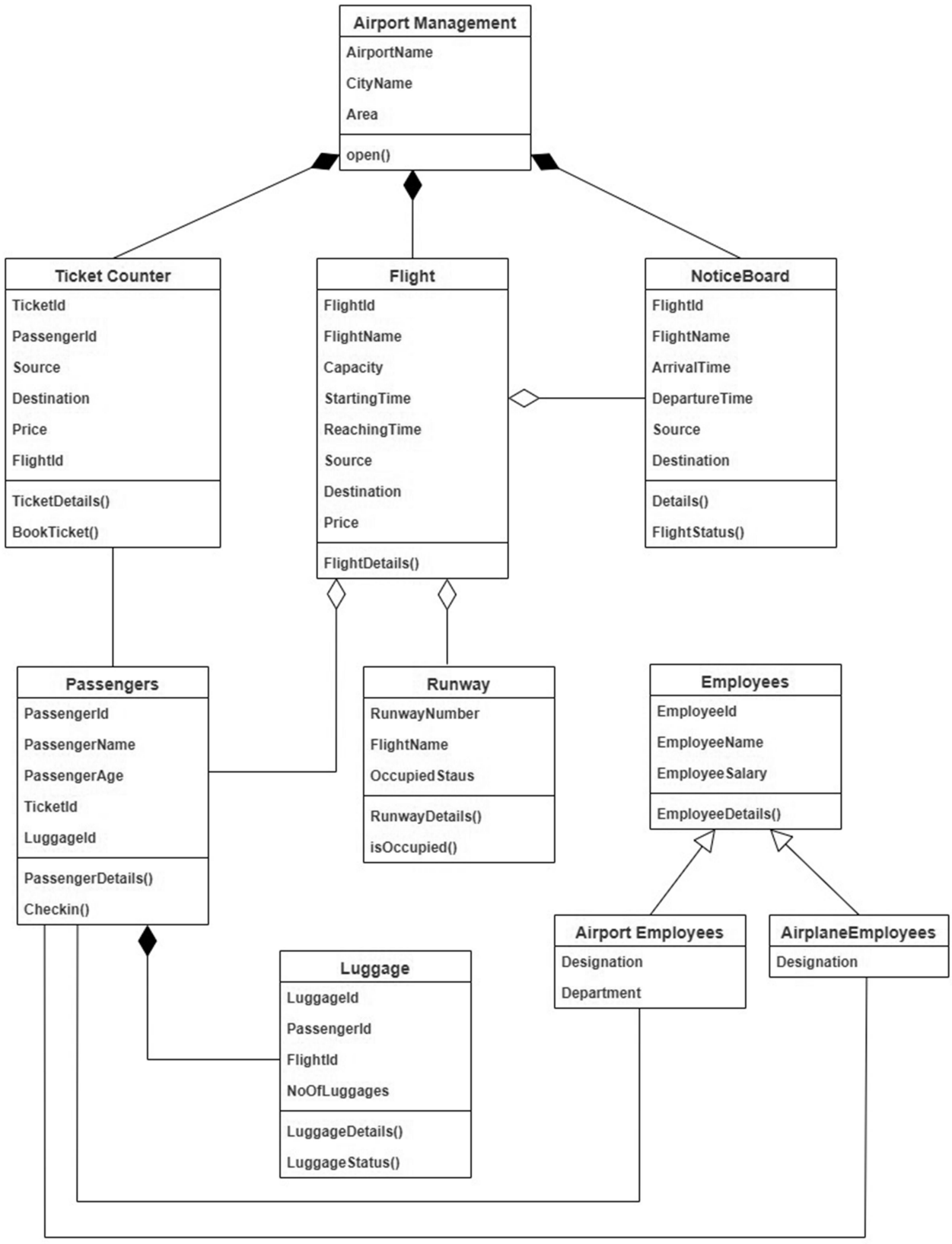


Figure 1: Airport Management System

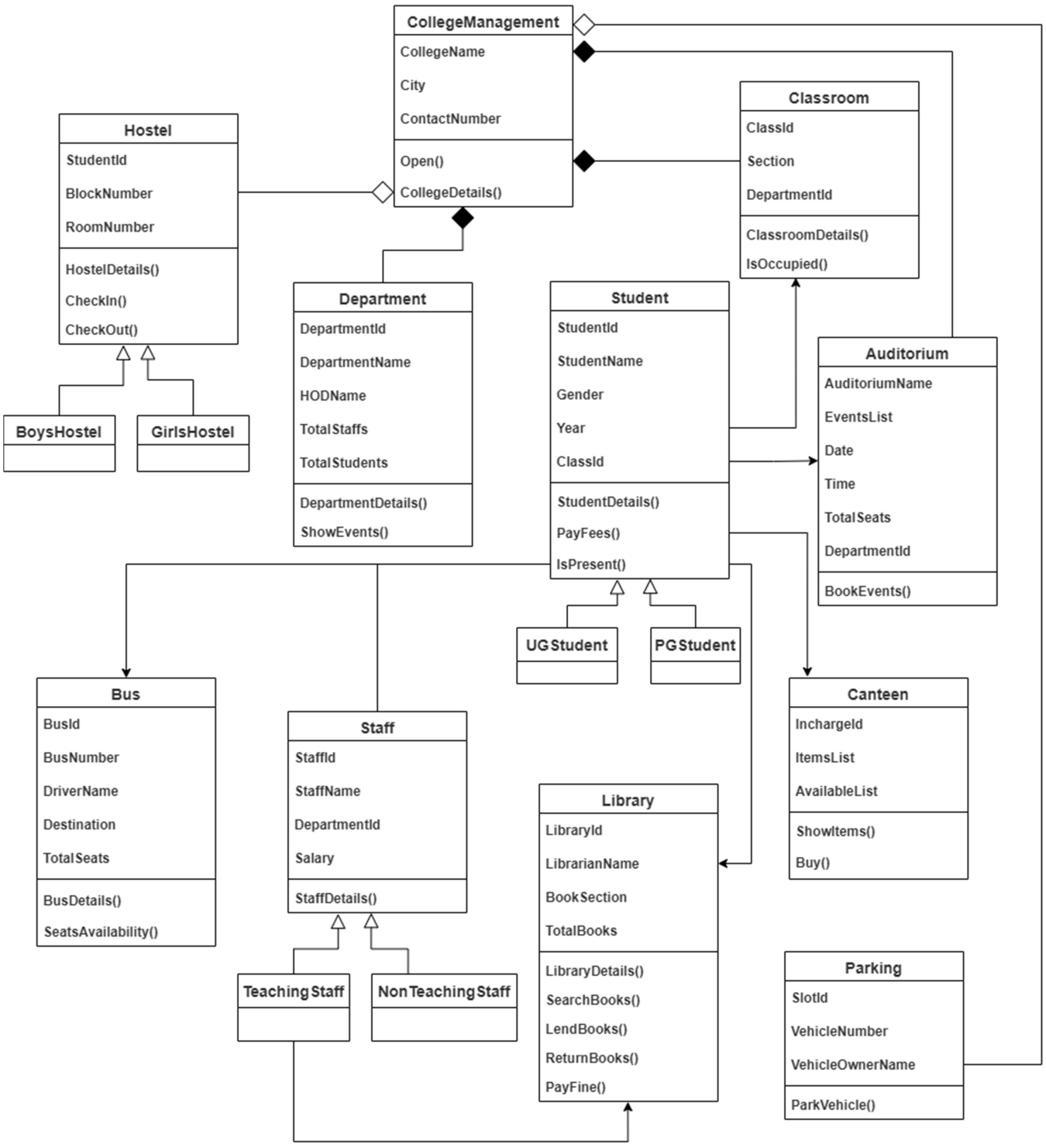


Figure 2: College Management System

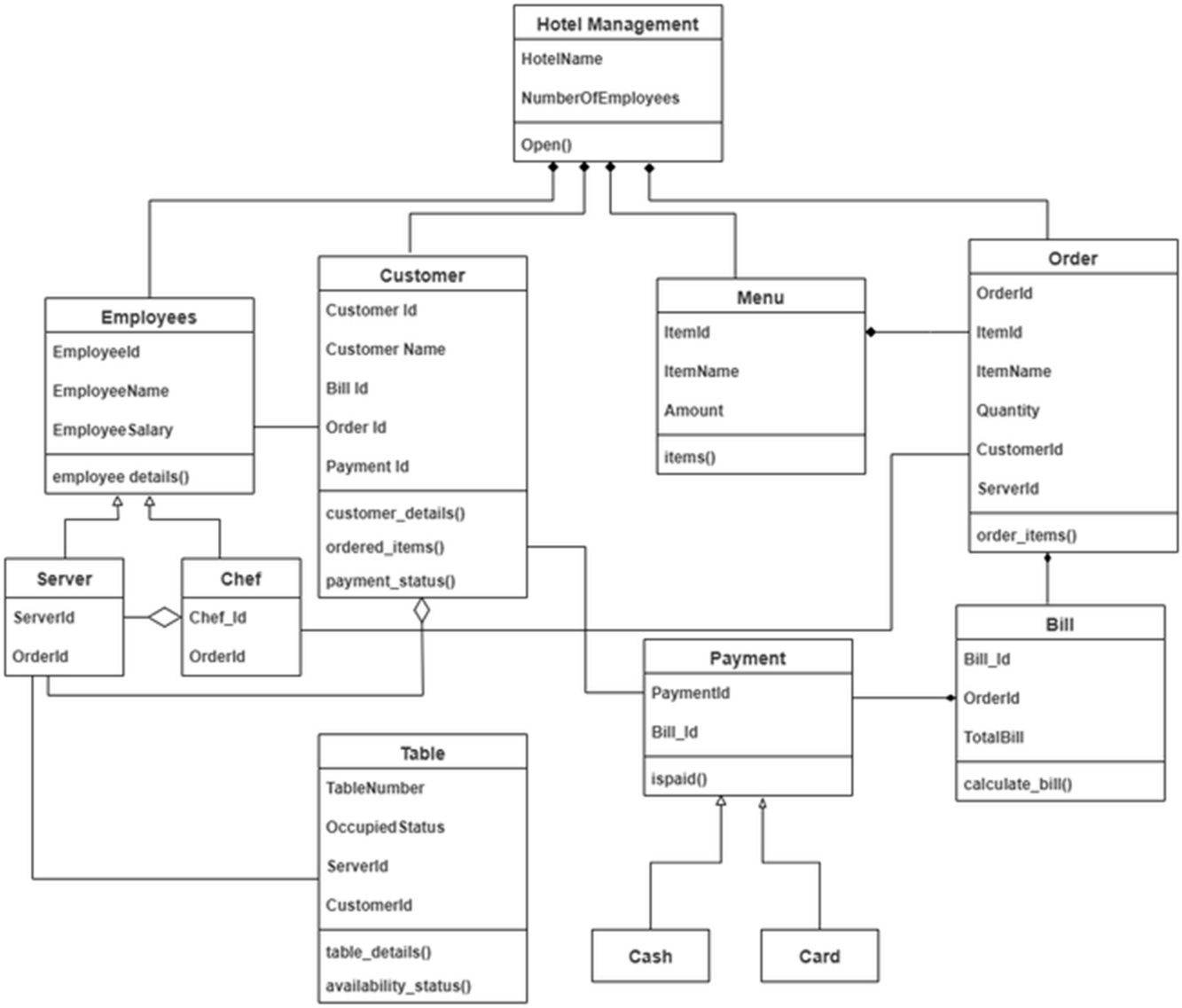


Figure 3: Hotel Management System

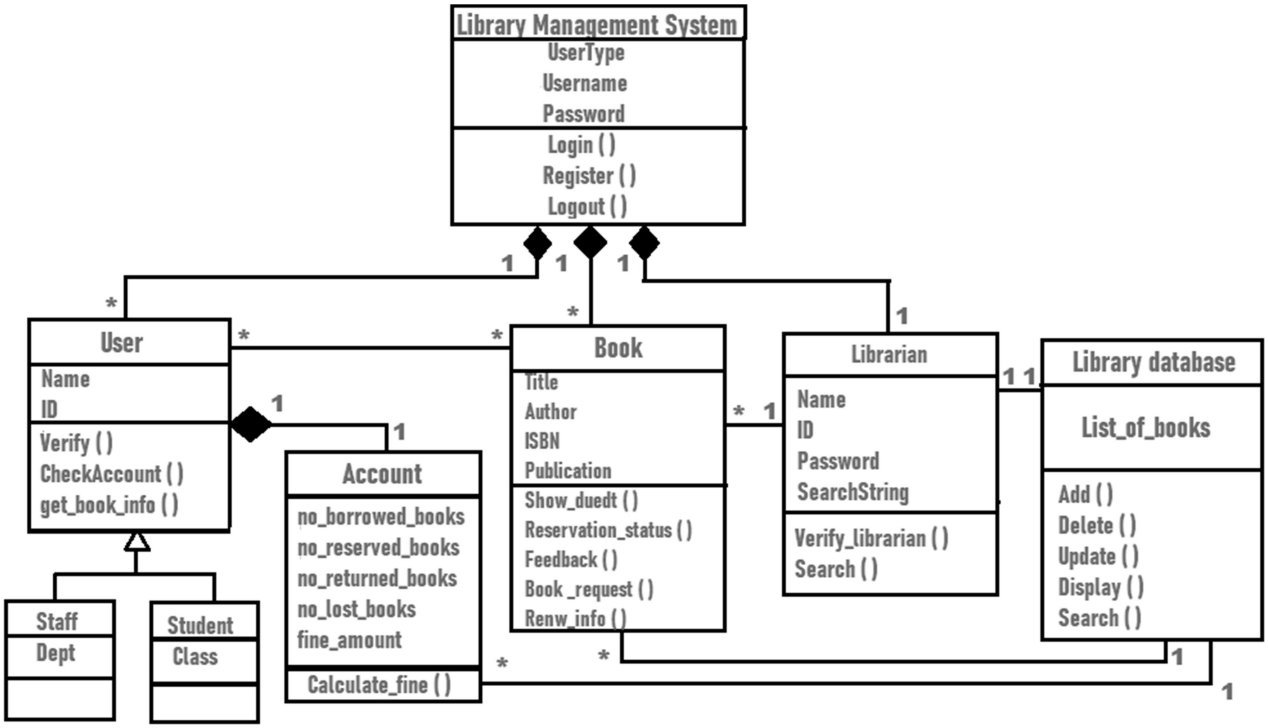


Figure 4: Library Management System

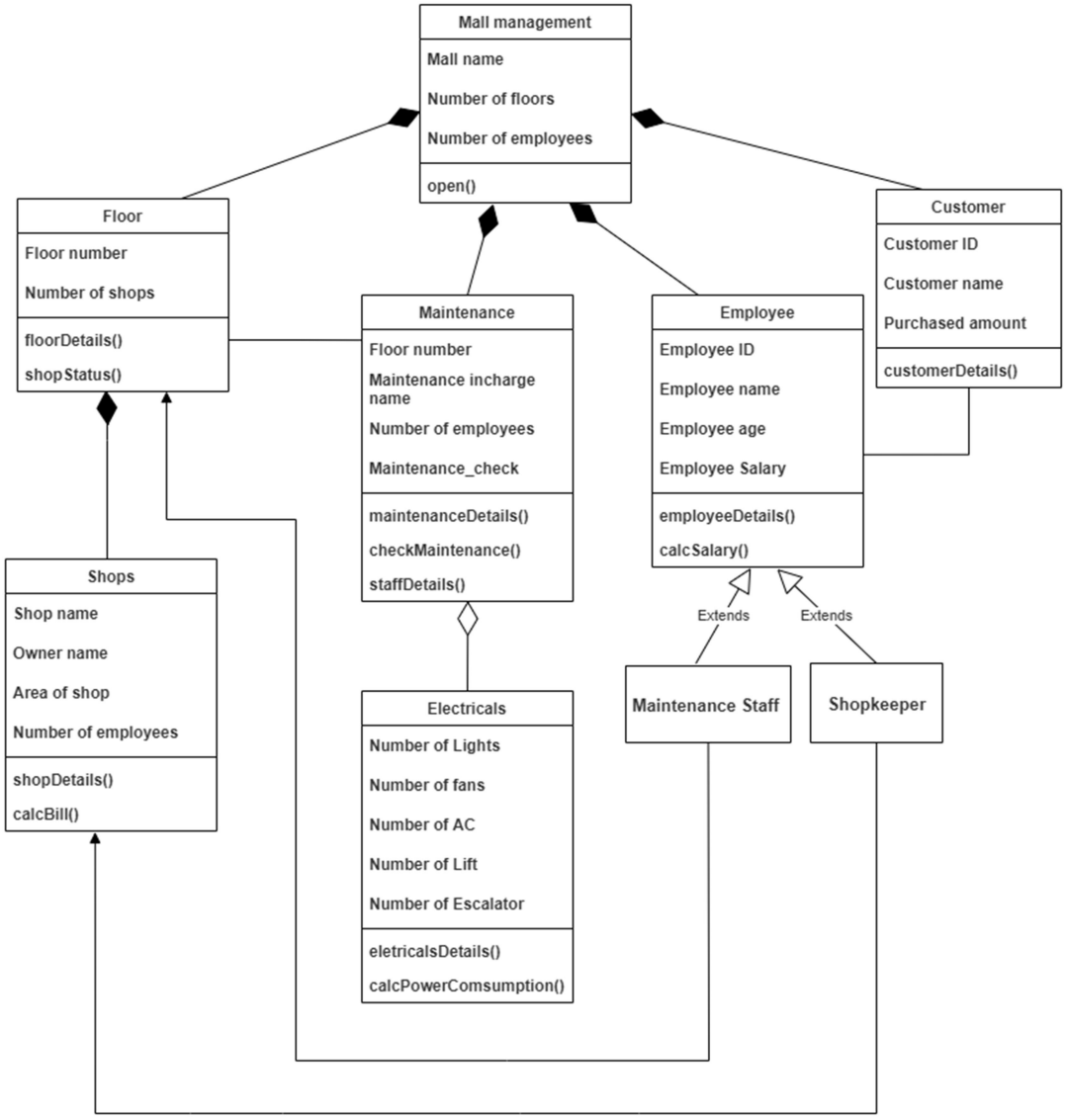


Figure 5: Mall Management System

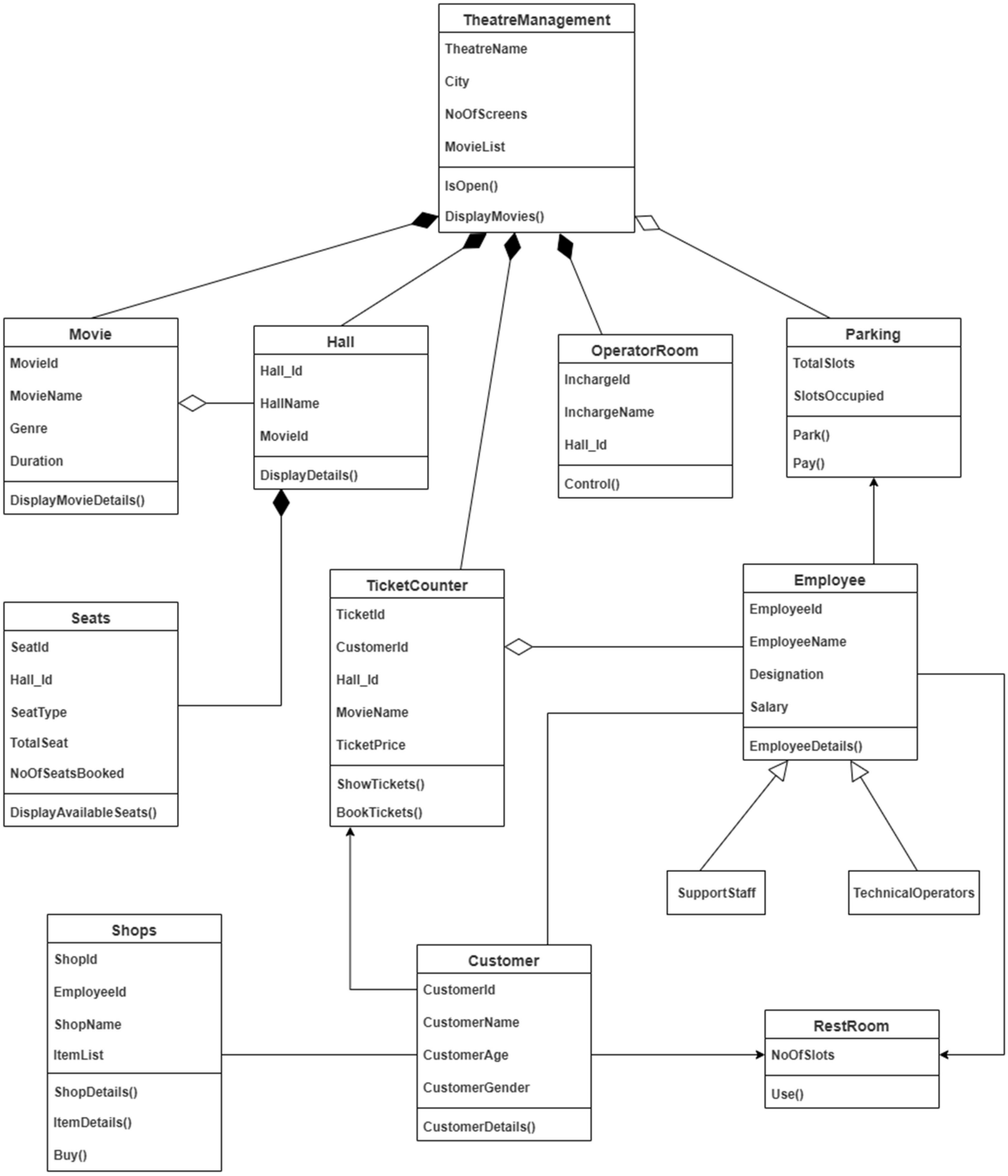


Figure 6: Theater Management System