**面向对象设计（Object-Oriented Design）**

# 软件工程学科与实践

## 概述

* 为什么要进行面向对象设计？
* 面向对象设计原则
* UML
  + 结构：类图（Class Diagrams）
  + 行为：序列图（Sequence Diagrams）
  + 内容：用例（Use Cases）

# 什么是面向对象设计？

面向对象软件：围绕对象及其交互结构化软件。

对象作为现实世界的实体，包含**状态（state）和行为（behavior）**。

主要构造：

* **类（Class）**：对象类型的蓝图（例如，汽车 Car）。
* **对象（Object）**：该类型的具体实例（例如，带有车牌号 AB25CDF 的 Ford Fusion ）。

# 为什么要进行设计？

* 组织想法
* 规划工作
* 建立对系统结构和行为的理解
* 与开发团队沟通
* 帮助未来的维护团队理解系统

# 面向对象设计原则

## 封装（Encapsulation）

* 将数据和行为封装在对象内部
* 防止未经授权的访问
  + **受控访问**：对象只能通过允许的方法修改
  + **保护数据完整性**

示例代码（部分）：

public class Animal {

private String name;

public String getName() {

return name;

} //name是private，只有用这个方法可以访问name

}

## 抽象（Abstraction）

* 关注核心问题
* 忽略不必要的细节
* 仅暴露必要的信息
* 隐藏类/对象的复杂性
  + **松耦合（Loose Coupling）**：对象通过抽象接口交互

示例代码（部分）：

public abstract class Animal {

private String name;

public String getName() {

return name;

}

public abstract void makeSound();

}

class Cat extends Animal {

public void makeSound() {

System.out.println("Meow!");

}

}

class Dog extends Animal {

public void makeSound() {

System.out.println("Woof!");

}

}

## 继承（Inheritance）

* 继承属性和行为
* 代码复用
  + **减少重复编写**
  + **减少错误和代码不一致**

示例代码（部分）：

public abstract class Animal {

private String name;

public String getName() {

return name;

}

public abstract void makeSound();

}

class Cat extends Animal {

public void makeSound() {

System.out.println("Meow!");

}

}

## 多态（Polymorphism）

* 允许子类对象替代父类对象
* 行为取决于子类对象
  + **动态方法分派（Dynamic Method Dispatch）**
  + **可扩展性（Extensibility）**
  + **灵活性和模块化系统**

示例代码（部分）：

public class Main {

public static void main(String[] args) {

Animal animal1 = new Dog();

Animal animal2 = new Cat();

animal1.makeSound(); // 输出: Woof!

animal2.makeSound(); // 输出: Meow!

}

}

## 组合（Composition）

* 通过其他对象构建对象
  + **代码复用**
  + **增量构建**

示例代码（部分）：

class Legs {

private int numOfLegs;

public Legs(int numOfLegs) {

this.numOfLegs = numOfLegs;

}

public void walk() {

System.out.println("Walking on " + numOfLegs + " legs");

}

}

public class Animal {

private Legs legs;

public void move() {

System.out.print("Animal is ");

legs.walk();

}

}

# UML 类图（Class Diagrams）

## 什么是类图？

* 静态视图，展示系统的类及其关系。

## 类的可见性

* + 公开（public）
* - 私有（private）
* # 保护（protected）
* ~ 包级（package-private）

## UML 关系

* **关联（Association）**：表示两个类之间的关系。
* **聚合（Aggregation）**：表示整体-部分关系，部分可以独立存在。
* **组合（Composition）**：更紧密的整体-部分关系，部分不能独立存在。
* **泛化（Generalization）**：即继承关系。

# UML 序列图（Sequence Diagrams）

## 对象协作

* 对象通过\*\*消息（message）\*\*进行交互。
* **时间轴**：垂直方向表示时间流动。
* **对象存在的生命周期**：对象的生存期用虚线表示。
* **消息传递**：可以是同步或异步的。

## 序列图分析方法

1. **分解场景**，确定涉及的对象。
2. **确定对象执行的操作**。
3. **确定控制流如何传递**。

## 交互框架（Interaction Frames）

* alt：用于 if-else 逻辑。
* loop：表示循环结构。

# 总结

* **面向对象设计原则**：封装、抽象、继承、多态、组合。
* **UML 类图**：定义类、属性、方法及其关系。
* **UML 序列图**：表示对象之间的交互和时间顺序。
* **关系类型**：关联AssociationMultiplicity？、聚合Aggregation、组合Composition、泛化Navigability。
* Multiplicity、Aggregation、Composition、Navigability。
* **分析方法**：如何找出类、对象及其交互方式。

以上是完整的翻译内容，如果需要调整或补充，请告诉我！