UML Class Diagram

A class diagram describes the structure of an object-oriented system by showing the classes (and interfaces) in that system and the relationships between the classes (and interfaces).

The most common relationships are: dependency, association, generalization, and realization.

Dependency Relationship

Dependency indicates a “uses” relationship between two classes. In a class diagram, a dependency relationship is rendered as a dashed directed line.

![Dependency Diagram](image)

If a class A “uses” class B, then one or more of the following statements generally hold true:

1. Class B is used as the type of a local variable in one or more methods of class A.
2. Class B is used as the type of parameter for one or more methods of class A.
3. Class B is used as the return type for one or more methods of class A.
4. One or more methods of class A invoke one or more methods of class B.

Association Relationship

Some objects are made up of other objects. Association specifies a “has-a” or “whole/part” relationship between two classes. In an association relationship, an object of the whole class has objects of part class as instance data.

In a class diagram, an association relationship is rendered as a directed solid line.

![Association Diagram](image)

If there is an association relationship between class A and class B, i.e., class A “has-a” class B, then the following statements hold true:

1. Class B is used as the type of one or more fields (instance or class variables) in class A.
2. The methods of class A generally invoke the methods of class B.

Aggregation and composition are two variants (or special forms) of the association relationship. In a class diagram, aggregation relationship is rendered as a solid line with an open diamond near the whole class.
Composition is rendered as a solid line with a filled diamond near the whole class.

For the purposes of this course, we will treat association, aggregation, and composition relationships as equal and interchangeable.

**Generalization Relationship**

A generalization is a relationship between a general thing (called the superclass or parent class) and a more specific kind of that thing (called the subclass or child class). Generalization is sometimes called an “is-a” relationship and is established through the process of inheritance.

In a class diagram, generalization relationship is rendered as a solid directed line with a large open arrowhead pointing to the parent class.

Use generalizations when you want to show parent/child relationships. The above class diagram reflects the relationships present in the following Java code fragment:

```java
public class B extends A {
    ...
}
```

**Realization Relationship**

A realization is a relationship between two things where one thing (an interface) specifies a contract that another thing (a class) guarantees to carry out by implementing the operations specified in that contract.
In a class diagram, realization relationship is rendered as a dashed directed line with an open arrowhead pointing to the interface.

Use realizations to show the relationship between a class and an interface. The above class diagram reflects the relationships present in the following Java code fragment. In this case, class A implements the contract (operations) specified in the interface B.

```java
public class A implements B {
    ...
}
```
Example: Consider the following program with five classes and one interface. The corresponding class diagram is shown on the last page.

```java
public interface IBook {
    void setPages(int pages);
    int getPages();
}

public class Book implements IBook {
    protected int pages = 1500;
    private Publisher publisher;

    public void setPages(int numPages) {
        pages = numPages;
    }

    public int getPages() {
        return pages;
    }

    public Publisher getPublisher() {
        return publisher;
    }

    public void setPublisher(Publisher publisher) {
        this.publisher = publisher;
    }
}

public class Dictionary extends Book {
    private int definitions = 52500;

    public double computeRatio() {
        return definitions/pages;
    }

    public void setDefinitions(int numDefinitions) {
        definitions = numDefinitions;
    }

    public int getDefinitions() {
        return definitions;
    }
}
```
public class Publisher {
    private String name;
    private Address address;

    public Publisher(String name, Address address) {
        this.name = name;
        this.address = address;
    }

    public String getName() {
        return name;
    }

    public Address getAddress() {
        return address;
    }
}

public class Address {
    private String streetAddress, city, state;
    private long zipCode;

    public Address(String street, String town, String st, long zip) {
        streetAddress = street;
        city = town;
        state = st;
        zipCode = zip;
    }

    public String toString() {
        String result;
        result = streetAddress + "\n";
        result += city + ", " + state + " " + zipCode;
        return result;
    }
}
public class Words {

    public static void main(String[] args) {
        Dictionary webster = new Dictionary();

        System.out.println("Number of pages: "+
                            webster.getPages());

        System.out.println("Number of definitions: "+
                            webster.getDefinitions());

        System.out.println("Definitions per page: "+
                            webster.computeRatio());
    }
}

Class diagram for the program above is shown below. Note that the String class is not included in the class diagram. Also, this class diagram doesn’t include attributes (fields) and operations (methods) of classes. Take time to identify and understand the dependency, aggregation, generalization, and realization relationships shown in the class diagram.