Proxy

A structural pattern

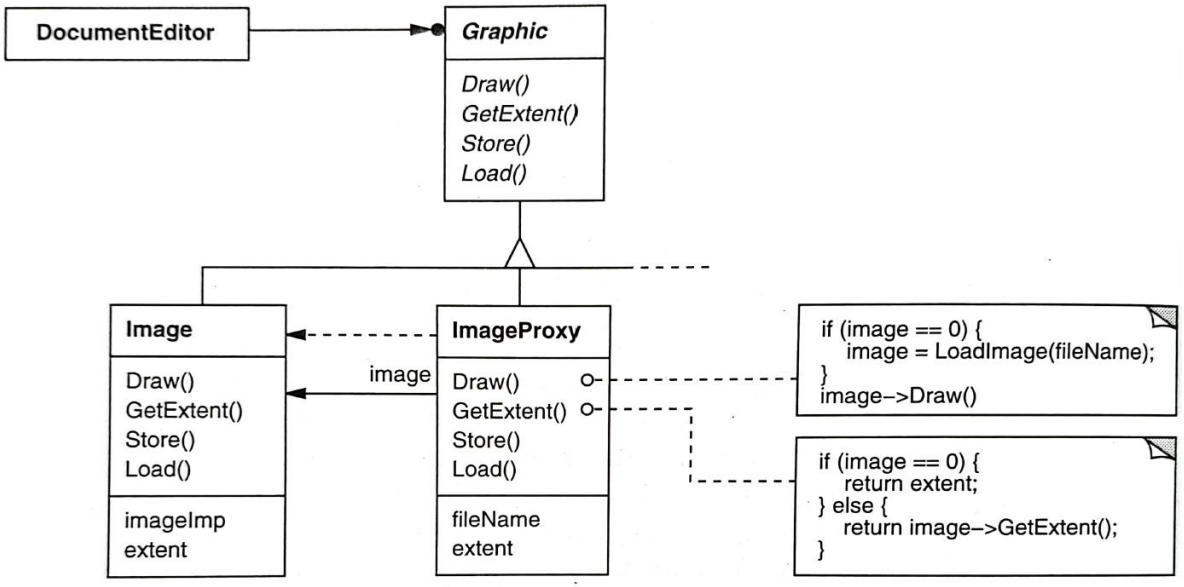
Learning goals

1. Learn the idea, structure, and Java implementation of the Proxy design pattern.
2. Learn to apply the Proxy DP in your own programming.

Idea of Proxy

- In some situations, direct access to an object has to be limited.
  - There may be a need for managing the lifecycle of the object, controlling access, or adding additional functionalities.
- Example: Accessing a large image file might need to be controlled so that the file is loaded into memory only when it is actually necessary (lazy loading).
  - The proxy may still show the image metadata, and load the actual image content only when it is needed.
- The Proxy Design Pattern provides **a surrogate or placeholder** for another object to control access to it.
  - The proxy and the real object both implement the same interface.
  - The client can work with the proxy just like with the real object.
- The proxy is responsible for creating and deleting the real object, controlling the access to it, or handling any additional housekeeping tasks, such as logging or security.
  - The proxy forwards requests to the real object once it decides that the time is right or the preconditions are met.
- The actual object can be changed at runtime by the proxy, and the client code does not need to be altered as long as it works with the object through the proxy interface.

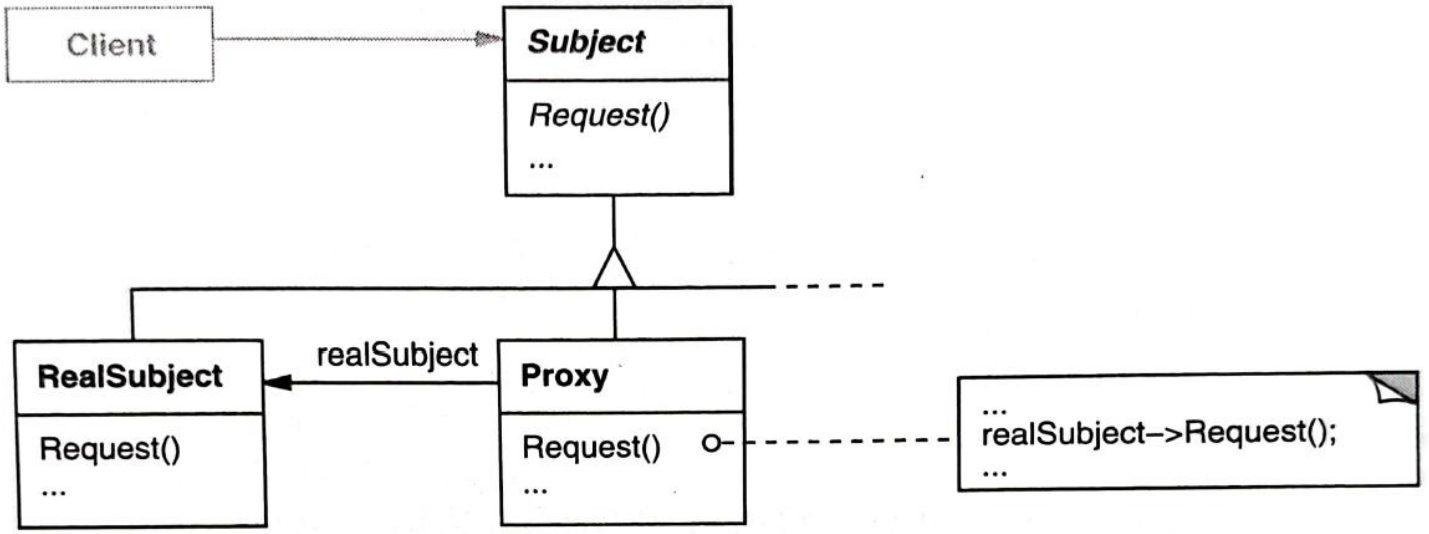
Example 1: Image loading



- The document interacts with a Graphic object that can be either an Image or an ImageProxy.
  - The Proxy is able to provide the image metadata even if the image is not loaded.
  - Whenever the ImageProxy is asked for the image content, it loads the image content from a file and instantiates an Image object.

Image: Gamma et al., Design Patterns. Elements of Reusable Object-Oriented Software. Addison Wesley Longman (1995), p. 208

General structure



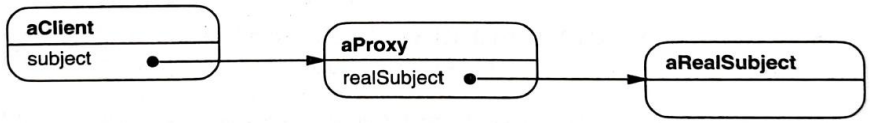
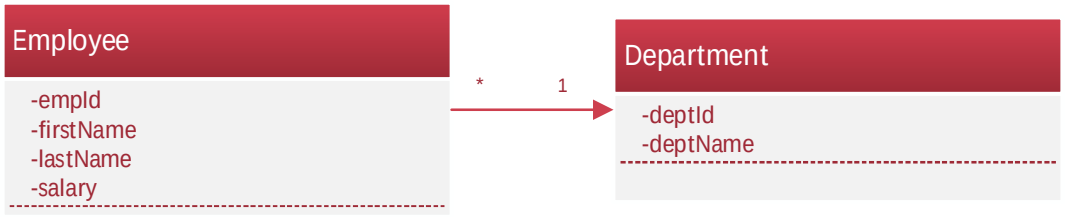


Image: Gamma et al., Design Patterns. Elements of Reusable Object-Oriented Software. Addison Wesley Longman (1995), p. 209

Roles

- **Proxy**: maintains a reference to the RealSubject, and acts as its surrogate. If necessary, creates the real subject.
- **Subject**: A common interface of a Proxy and a RealSubject. As only this is visible to the client, the Proxy and RealSubject are interchangable from client's point of view.
- **RealSubject**: the object that the Proxy represents.

Example 2: JPA lazy loading



company.employee: 3 r Next Show all Sorting

empId	firstName	lastName	salary	department_deptId
1	Mary	Smith	3 600	1
2	Urpo	Koski	2 600	2
3	Olga	Glebova	3 110	2

company.department: 2 Next

deptId	deptName
1	Marketing
2	Technology

- In previous studies, you have been utilizing the Proxy pattern with JPA (Jakarta Persistence API).
- Lazy loading is implemented using the Proxy design pattern.
  - The purpose of JPA lazy loading is to avoid an unnecessary multi-table database query.

Example 2: JPA lazy loading

- When you load an **Employee** object persisted in the database, the corresponding **Department** can be handled in two alternative ways:
  A. Eagerly
     - A **Department** is loaded immediately at the same time as the **Employee**.
  B. Lazily
     - A **Department** is loaded only when it's needed. Until then, it is represented by an unloaded proxy where only the identifying field is set.
       - A JPA proxy is a runtime subclass with modified getters that ask the **EntityManager** to load the real object.
       - The generation of such a **dynamic proxy**[1] requires dedicated libraries. (Hibernate uses ByteBuddy, https://bytebuddy.net/, that does bytecode manipulation).

1) Dynamic proxies are not covered on this course.

Example 2: JPA lazy loading

Dao.java

```
Employee emp = em.find(Employee.class, 2);
// Print data from the loaded object
System.out.println(emp.getFirstName());
// Print data from the related object
System.out.println(emp.getDepartment().getDeptName());
```

- Note that we're within the scope of the **EntityManager**.

With lazy loading, the modified getDeptName() in the auto-generated department proxy triggers a database search.

EAGER

```
@ManyToOne(fetch=FetchType.EAGER)
private Department department;
```

```
Hibernate: select employee0_.empId as empId1_1_0_, employee0_.department
Urpo
Technology
```

LAZY

```
@ManyToOne(fetch=FetchType.LAZY)
private Department department;
```

```
Hibernate: select employee0_.empId as empId1_1_0_, employee0_.department_deptId as d
Urpo
Hibernate: select department0_.deptId as deptId1_0_0_, department0_.deptName as dept
Technology
```

Types of proxies (by use case)

1. **Remote Proxy**
   - Acts as a local stand-in for an object in a different system.
   - Example: allowing a local client to use an object whose proper creation will need access to an external API.
2. **Virtual Proxy**
   - Defers the creation and initialization of an object until it is needed.
   - Example: delaying the loading of a high-resolution image or a large document.
3. **Protection Proxy**
   - Provides access control to sensitive objects.
   - Restricting access to methods based on permissions of the calling user.

Practical issues

- The Proxy pattern can introduce significant complexity and overhead.
  - More classes, indirect access
- Proxies must maintain the state of the service object they represent, which can be challenging
  - This is even harder in concurrent scenarios.
- Lazy loading must be carefully implemented to ensure it does not become a performance bottleneck.
  - Execution can slow down at an unexpected moment (from user's point of view).
- Testing can be more complex because the proxies need to be considered in the testing strategy.
- Use proxies sparingly and only when the benefits outweigh the costs.
- Sometimes, the Proxy pattern is interchangeable with the Decorator pattern, as Decorator adds functionalities to an object without modifying it.
  - However, the Proxy DP is specifically designed for controlling access.
  - The creation of a RealSubject is often deferred in the Proxy DP, and the creation of the RealSubject may be complex (e.g. a database query, or an API call). This is not possible with the Decorator pattern.