Proxy (pełnomocnik)

Cel:

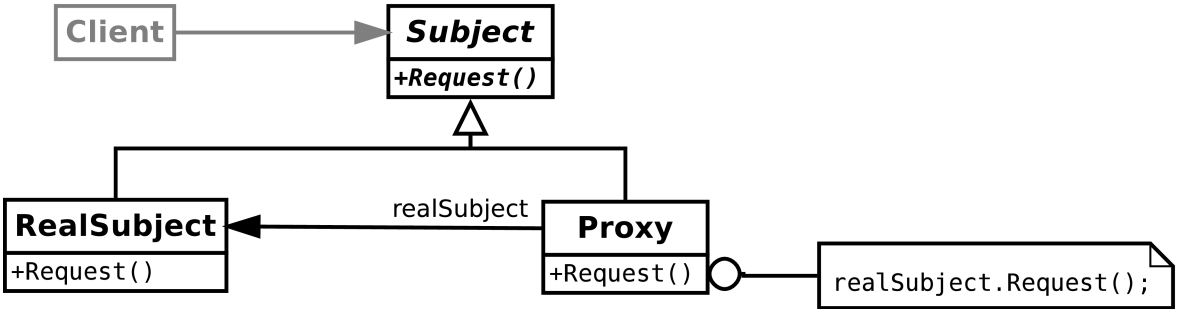
- Dostarczyć zamiennik pewnego obiektu, pozwalający kontrolować dostęp do niego.

Zastosowanie:

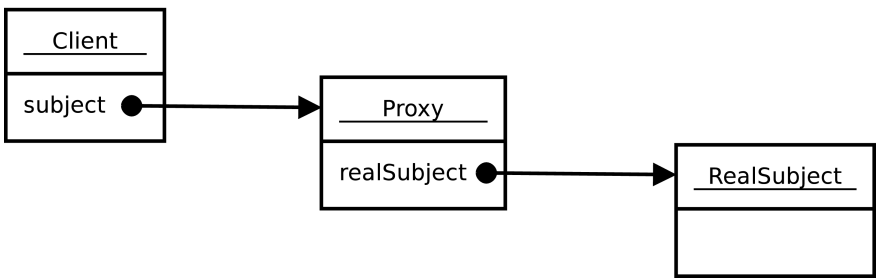
Wszędzie tam, gdzie oczekujemy bardziej zaawansowanego odwołania do obiektu, niż zwykły wskaźnik czy referencja. Przykłady:

- **zdalny proxy** – lokalna reprezentacja obiektu znajdującego się w innej przestrzeni adresowej (np. RMI), odpowiada za wysłanie żądań klienta i odebranie odpowiedzi („ambasador”)
- **wirtualny proxy** – tworzenie kosztownych obiektów na żądanie (np. ImageProxy), część danych może być przechowywana przez proxy
- **ochronny proxy** – kontroluje dostęp do obiektu, dzięki temu może wyłączać pewne operacje lub przeprowadzać kontrolę ich wykonywania
- **sprytne referencje** – zastępują tradycyjny wskaźnik, aby podejmować dodatkowe akcje, gdy obiekt jest udostępniany:
  - licznik referencji (aby obiekt mógł być usunięty, gdy osiągnie on 0)
  - lockowanie obiektów (przy dostępie współbieżnym)
  - copy-on-write (zapobiega tworzeniu zbędnych kopii)

Struktura:



Struktura dynamiczna:



Składniki:

- **Proxy**
  - Przechowuje referencję do rzeczywistego obiektu (RealSubject).
  - Ma ten sam interfejs, co RealSubject, dzięki czemu może go zastąpić.
  - Kontroluje dostęp do rzeczywistego obiektu; może też odpowiadać za jego tworzenie i usuwanie.
  - Inne możliwe odpowiedzialności:
    - *zdalny proxy* będzie odpowiadał za wysyłanie żądań do innej przestrzeni adresowej – może wymagać to np. nawiązania połączenia i wysyłania/ odbierania zakodowanych danych korzystając z określonego protokołu.
    - *wirtualny proxy* może przechowywać część danych rzeczywistego obiektu lub samodzielnie spełniać część żądań (np. ImageProxy może przechowywać i udostępniać rozmiary rzeczywistego obrazka).
    - *ochronny proxy* może kontrolować możliwość wykonania danego żądania (np. możliwość kasowania rekordów przez danego użytkownika).
  - Proxy oddelegowuje żądania do rzeczywistego obiektu (RealSubject).
- **Subject**
  - Interfejs wspólny dla RealSubject i Proxy – dzięki temu Proxy może być używany tam, gdzie klient oczekuje RealSubject.
- **RealSubject**
  - Definiuje rzeczywisty obiekt, kryjący się za Proxy.

Implementacja:

```
interface Subject {
     void request();
     void request(double value1, double value2);
     double request(int data);
}
class RealSubject implements Subject { ... }
class Proxy implements Subject {
     private RealSubject realSubject;
     public Proxy() {
          realSubject = new RealSubject();
     }
     public void request() { realSubject.request(); }
     public void request(double value1, double value2) {
          realSubject.request(value1, value2); }
     public double request(int data) {
          return realSubject.request(data); }
}
//... Client ...
     Subject subject = new Proxy();
     subject.request();
     subject.request(subject.request(10), 2.5);
```

Zdalny proxy

```
interface MailBox {
    void send(String message);
    String receive(String message);
}
class MailBoxProxy implements MailBox {
    private String host;
    public MailBox(String host) {
        this.host = host;
    }
    void send(String message) {
        try {
            Socket connection = new Socket(host);
            OutputStream os = connection.getOutputStream();
            ObjectOutputStream oos;
            oos = new ObjectOutputStream(os);
            oos.writeObject(message);
            connection.close();
        catch (Exception ex) { /*...*/ }
    }
    String receive(String message) { /*...*/ }
}
```

Wirtualny proxy

```
interface Image {
    int width();
    int height();
    void setPixel(int x, int y, Color color);
    Color getPixel(int x, int y);
}
class ImageProxy implements Image {
    private RealImage realImage = null;
    private int width, height;
    public ImageProxy(int width, int height) {
        this.width = width;    this.height = height;
    }
    public int width() { return width; }
    public int height() { return height; }
    public void setPixel(int x, int y, Color color) {
        if(realImage == null)
            realImage = new RealImage(width, height);
        realImage.setPixel(x, y, color);
    }
    public Color getPixel(int x, int y) { /* ... */ }
}
```

Ochronny proxy

```
interface Database {
    Result addRecord(Record rec);
    Result removeRecord(Record rec);
    Result changeRecord(Record rec);
}
class DatabaseProxy implements Database {
    private RealDatabase realDatabase;
    private DatabaseProxy() {
        realDatabase = new RealDatabase();
    }
    public Result addRecord(Record rec) {
        if(User.currentUser().has(UserPrivileges.ADD))
            return realDatabase.addRecord(rec);
        else
            return Result.ERROR;
    }
    public Result removeRecord(Record rec) {
        return Result.ERROR;
    }
    public Result changeRecord(Record rec) { /* ... */ }
}
```

Sprytne referencje (licznik referencji w C++)

```
template<class E>
class SmartPointer {
     E* realSubject;
     int* licznik;
public:
     SmartPointer(E* s) { s = s; licznik = new int(1); }
     SmartPointer(const SmartPointer& sp) {
          realSubject = sp.realSubject;
          licznik = sp.licznik; ++(*licznik); }
     ~SmartPointer() {
          if(--(*licznik) == 0) delete realSubject; }
     E* operator->() { return realSubject; }
};

int main() {
     SmartPointer<Stos> p1 = new Stos(10);
     p1->push(3); p1->push(7);
     if ( /* ... */ ) { SmartPointer<Stos> p2 = p1;
                        p2->pop(); }
     /* ... */
     return 0; }
```

Sprytne referencje (copy-on-write)

```
interface Subject {
     int readOperation();
     void writeOperation(int value);
}
class Proxy implements Subject {
     private RealSubject realSubject;
     private Licznik licznik;
     public Proxy() {
          realSubject = new RealSubject();
          licznik = new Licznik(1);
     }
     public Proxy(Proxy p) {
          // kopia płytka
          realSubject = p.realSubject;
          licznik = p.licznik;
          licznik.increment();
     }
     public int readOperation() {
          return realSubject.readOperation();
     }
```

Sprytne referencje (copy-on-write)

```
    public void writeOperation(int value) {
        if(licznik.value() > 1) {
            licznik.decrement();
            // kopia głęboka
            realSubject = realSubject.clone();
            licznik = new Licznik(1);
        }
        realSubject.writeOperation(value);
    }
}
```

Powiązania:

- Adapter – używa podobnej struktury, by zmieniać interfejs obiektów
- Dekorator – zbliżony w implementacji, choć stawia nacisk na inne cele: dodawanie funkcjonalności do obiektu. Niektóre rodzaje proxy (np. *ochronny proxy*) mogą być realizowane jako dekoratory.