Replace Inheritance with Delegation

Replace an **inheritance relationship** with **delegation** to reduce coupling and increase flexibility.

- We learned that there are two ways to reuse other code: inheritance and delegation.

Favor composition over inheritance principle suggests using **composition (delegation)** as the primary way to reuse code, reserving **inheritance** only for true **"is-a"** relationships where polymorphism is needed.

Inheritance

- Inheritance is a powerful tool, but it also couples one class to another strongly.
- After using inheritance, it becomes harder to change the inheritance relationship.

- In design patterns, Template method, Composite, Strategy patterns use inheritance.
- It is often said that Inheritance is the last resort, which means consider using delegation first.
- In this example, we discuss how to refactor to use delegation.

```
class Another {
    void method() {
        ...
    }
    ...
}

class Something extends Another {
    ...
}
```



```
class Something {
    Another _delegate = new Another();
    ...
    void method() {
        _delegate.method();
    }
}
```

Example: Dice

Before:

- The Dice class inherits from the Random class.

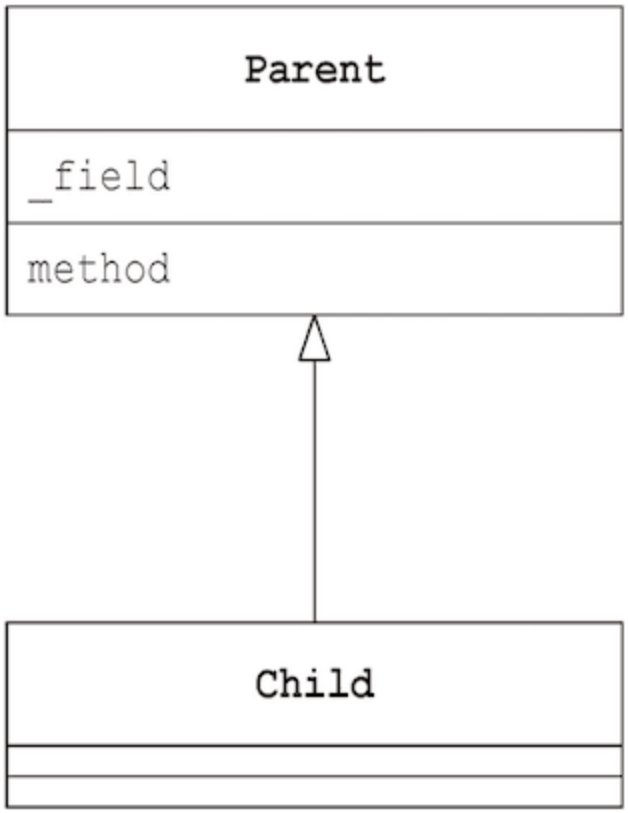
```
class Dice(random.Random):
    def __init__(self, seed=314159):
        super().__init__(seed)
    def randint(self, a, b):
        if a != 1 or b != 6:
            raise UnsupportedError("Dice only supports 1-6 range")
        return super().randint(1, 6)

    def next_int(self):
        """Get next dice roll (1-6)"""
        return super().randint(1, 6)
```

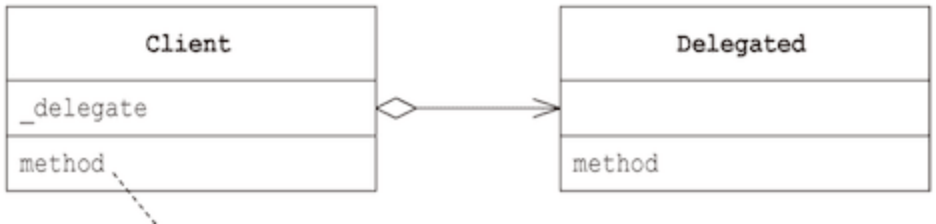
Refactoring: use delegation (aggregation) instead

After:

```
import random
class Dice:
    def __init__(self, seed=314159):
        self._random = random.Random(seed)
    def next_int(self):
        return self._random.randint(1, 6)
    def set_seed(self, seed):
        """Set the random seed"""
        self._random.seed(seed)
```

- These UML diagrams show the change from inheritance to delegation using aggregation.





Discussion

The problems with inappropriate inheritance

1. **Tight coupling** - subclass depends on superclass implementation details
2. **Breaks encapsulation** - subclass can access protected members
3. **Fragile base class** - changes to superclass can break subclasses
4. **Interface pollution** - subclass inherits methods it doesn't need
5. **Multiple inheritance conflicts** - diamond problem in some languages

Benefits for Replace Inheritance with Delegation

1. **Loose coupling** - delegating class only depends on interface
2. **Flexible composition** - can change delegate at runtime
3. **Selective interface** - expose only needed methods
4. **Better encapsulation** - delegate's internals are hidden
5. **Easier testing** - can mock the delegate