**항목 38. “has-a (~는 ~를 가짐)” 혹은 “is-implemented-in-terms-of(~는 ~를 써서 구현됨)”를 모형화할 때는 객체 합성을 사용하자.**

합성(composition)이란, 어떤 타입의 객체들이 그와 다른 타입의 객체들을 포함하고 있을 경우에 성립하는 그 타입들 사이의 관계를 일컫습니다. 포함된 객체들을 모아서 이들을 포함한 다른 객체를 합성한다는 뜻인데, 다음과 같은 경우이다.

|  |
| --- |
| class Address {...};  class PhoneNumber {...};  class Person  {  public:  ......  private:  **std::string name;**  **Address address;**  **PhoneNumber voidNumber;**  **PhoneNumber faxNumber;**  } |

Person 객체는 string, Address와 PhoneNumber 객체로 이루어져 있습니다. 이런 경우를 두고 합성이라는 말을 사용하는데 다른 말로는 **레이어링(layering), 포함(containment), 통합(addregetion), 내장(embedding)** 이라고 한다

Public 상속의 의미가 “is-a(~는 ~의 일종이다.)” 라고 배웠다.(항목 32) 객체 합성 역시 의미를 갖고 있다. 실제 뜻이 2개나 되는데 2개인 이유는 소프트웨어 개발에서 대응하는 영역이 두 가지 이기 때문이다. 사물을 본뜬 사람, 이동 수단, 비디오 프레임 등인데, 이런 객체는 소프트웨어의 응용 영역(application domain)에 속하고, 나머지 들은 버퍼, 뮤텍스, 탐색 트리 등 순수하게 시스템 구현 만을 위한 인공물입니다. 이런 종류의 객체가 구현 영역(implementation domain)이라고도 한다. 여기서 객체 합성이 응용 영역의 객체들 사이에서 일어나면 has-a 관계이다. 반면, 구형 영역에서 일어나면 is-implemented-in-terms-of 관계를 나타낸 것이다.

위의 **person 클래스는 has-a관계**이다. 사람이 이름의 일종, 사람이 주소의 일종이라고는 말할 수는 것이다. 그러나 사람이 이름을 가지며 사람이 주소를 가진다고 하는 것은 자연스럽다. 이렇기 때문에 is-a관계와 has-a 관계의 역할을 헷갈리는 경우가 없어야 한다.

헷갈리는 부분은 is-a 관계와 is-implemented-in-terms-of 관계의 차이점일 것이다.

한 예로, 객체로 구성된 작은 집합, 중복 원소가 없는 집합체를 나타내고 저장 공간도 적게 차지하는 클래스 템플릿이 필요하다고 했을 때, 표준 라이브러리의 set템플릿을 활용해 보기로 하자. Set 템플릿은 원소 한 개당 포인터 세 개의 오버헤드가 걸리도록 구현되어 있다.

그러므로 표준 라이브러리의 set템플릿은 적당하지 않다. 이번엔 list를 사용해 보도록 하자. 이번 포인트는 set 템플릿을 만들되 list에서 써서 구현 되는 (is-implemented in terms of) 형태의 설계가 가능하다는 사실을 잡아내는 것이다.

|  |
| --- |
| template<class T>  class set  {  public:  bool member(const T& item) const;  void insert(const T& item);  void remove(const T& item);  std::size\_t size() const;  private:  **std::list<T> rep;**  } |

Set의 멤버 함수는 list에서 이미 제공하는 기능 및 표준 C++ 라이브러리의 다른 구성 요소를 잘 버무려서 만들기만 하면 되기 때문에, 실제 구현은 아주 쉽게 이해할 수 있을 정도로 간단하다.

|  |
| --- |
| template<class T>  class set  {  public:  bool member(const T& item) const;  void insert(const T& item);  void remove(const T& item);  std::size\_t size() const;  private:  std::list<T> rep;  }  template<class T>  bool set<T>::menber(const T& item) const  {  return set<T>::find(rep.begin(), rep.end(), item) != rep.end();  }  template<class T>  void set<T>::insert(const T& item)  {  if(!member(item))  rep.push\_back(item);  }  template<class T>  void set<T>::remove(const T& item)  {  typename std::list<T>::iterator it = std::find(rep.begin(), rep.end(), item);  if (it != rep.end())  rep.erase(it);  }  template<class T>  std::size\_t set<T>::size() const  {  return rep.size();  } |

구현된 set에서 중요한 점은 주렁주렁 인터페이스가 아니라 바로 list와의 관계이다. 이 관계는 is-a(~는 ~를 가짐)이 아니라, is-implemented-in-terms-of 이다.

**요약**

- 객체 합성(composition)의 의미는 public 상속이 가진 의미와 완전히 다릅니다.

- 응용 영역에서 객체 합성의 의미는 has-a(~는 ~를 가짐)입니다. 구현 영역에서는 is-implemented-in-terms-of(~는 ~를 써서 구현됨)의 의미를 갖습니다.