CS 246 Spring 2018 — Tutorial 8

June 27, 2018

Summary

1 Class Relationships

- There are three types of relationships between classes which we typically discuss:
  - **Composition(owns-a):** class A *owns an* instance of class B. This means that class A is responsible for deleting the instance of class B when an object of class A is destroyed.
    - the instance of B that is owned by A cannot exist outside of A, though it can be referred to or pointed at from outside A
  - **Aggregation(has-a):** class A *has an* instance of class B. This means that class A is not responsible for deleting the instance of class B.
    - if an object of class A is deleted, the instance of class B associated to it lives on.
    - multiple objects of class A can have the same instance of class B.
  - **Inheritance (is-a):** class B *is an* instance of class A. This means that an instance of class B can be used in any situation where an instance of class A can be used.

    Note:

    - the converse is not true. That is, an instance of class A cannot always be used where an instance of class B can be used.
    - for this course, we are mainly concerned with public inheritances.

- **Note:** If a class A has a pointer to an instance of class B, you cannot know if the relationship is composition or aggregation without looking at documentation.

```
class B {
    ...
};

class A {
    B b; // This is composition
    B *b2; // This could be composition or aggregation
};
```

2 Inheritance

- Example:

```
class A {
    int a;

public:
    A(int a) : a{a} {}
};

class B : public A {
    int b;

public:
    B(int a, int b) : A{a}, b{b} {}
};
```

- In this example, B **inherits** from A (this is what the "`: public A`" is for). This means that every instance of B has the fields and methods which an instance of A has.
- Note the constructor for the B class. The first element of the MIL is `A{a}` which is calling the constructor for the A portion of the B object.

3 Encapsulation and Inheritance

- If A has fields which are private, B cannot access these fields (as they are private).
- What are some benefits of an inherited class not having direct access to the fields of the superclass?
  - Other people may inherit from our classes and this means they'd have access to the fields of the superclass in their implementation of their class, and *this breaks encapsulation.*

- However, we often want to give subclasses "special access" to the class.
  - For instance, we might want to have some accessor methods so that subclasses can access fields in a way that we choose, but we don't want to let everyone have access to these members.
- For this purpose, we can use the third type of privacy: `protected`.
- Members which are `protected` can be accessed directly by subclasses but cannot be accessed by the public.
- **Note:** Most of the time you should not make fields `protected` as this also breaks encapsulation.

4 Polymorphism

- As previously stated, if there is an inheritance relationship between two classes, an instance of the subclass can be used anywhere an instance of the superclass can be used.
- This means that each of the following is valid:

```
B b{1, 2};
A a = b;
A *a = new B{3, 4};

void foo(A a);
void foo2(A &a);

foo(b);
foo2(b);
```

- Note however, that a B object is larger than an A object (it has an extra field).

  This means that any time we force a B object into an A object, it doesn't fit and the object will be **sliced**; only the A part of B is copied (if a copy is made).

  This "slicing" is called **object coercion**.
- But if we simply create a pointer/reference to A from a B object (e.g. `A *a = new B{3, 4};`), it is just pointing at the A part inside B, and thus no slicing will occur. Note that you will not be able to access the fields of B through a pointer/reference to A, since it can point to something that is not a B.
- **Note:** You can consider **object slicing** as an alternate term for **object coercion**.

5 Arrays and Inheritance

- Continuing with A and B, consider the following situation:

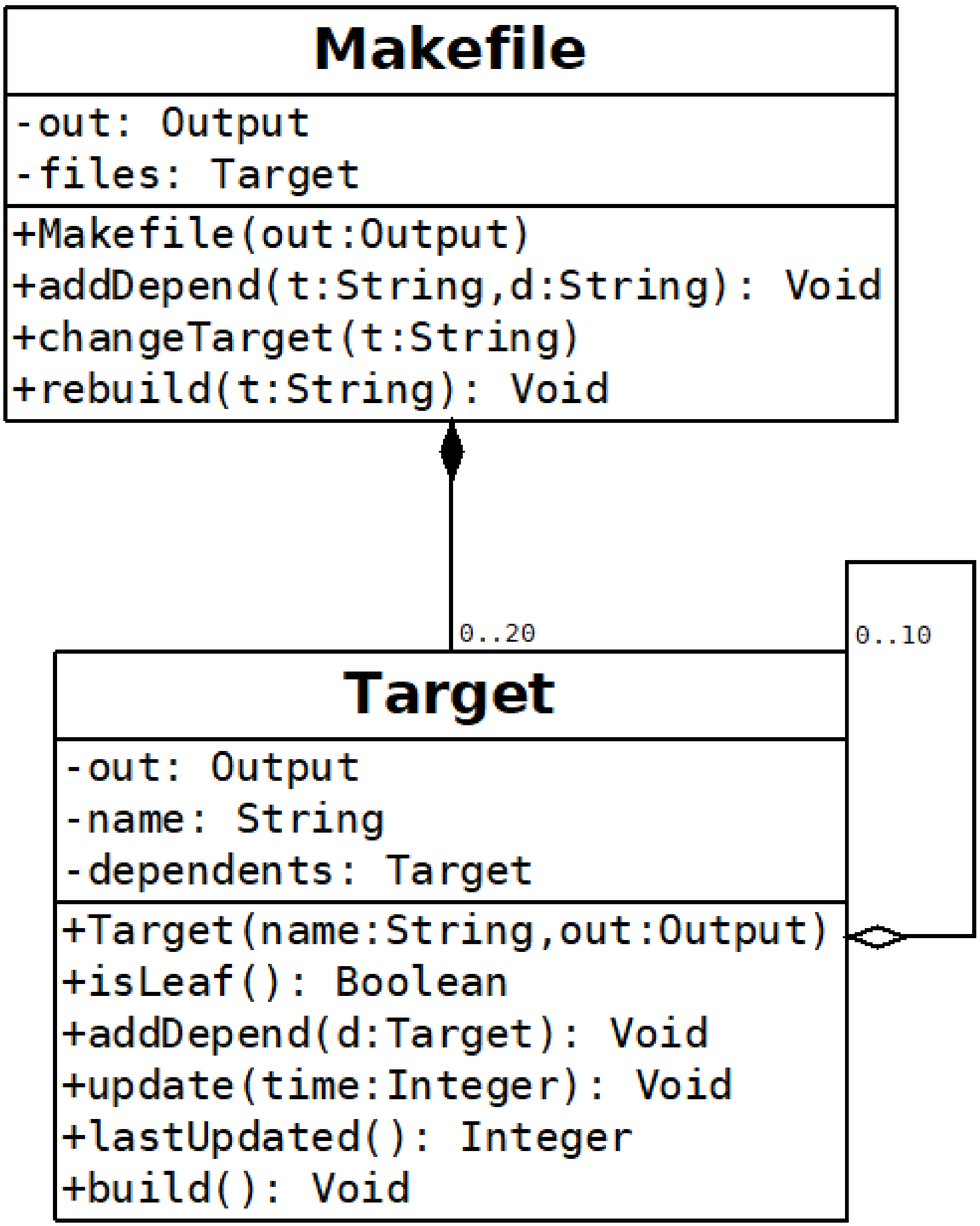
```
void foo(A *arr) {
    arr[0] = A{10};
    arr[1] = A{7};
}

B arr[2] = {{1, 2}, {3, 4}};
foo(arr);
```

- What happens with this code?
  - Well, it compiles perfectly fine as the types match.
  - However, the function `foo` believes the array which it receives is an *array of A's*.
  - This means that when we assign a value to `arr[1]`, *the value 7 will actually be assigned to the location where 2 is stored.*
- This means that our data is **misaligned** and while what we are doing in this case is predictable, this is very dangerous.
- **Take away:** *Never* use array objects polymorphically. If you want polymorphic array, use an array of pointers to objects.

6 UML

- The purpose of a UML is to:
  - Give information for how a class should be implmented.
  - Inform the reader about the multiplicity of each element.
  - Inform the reader of the relationship between classes (composition, aggregation, inheritance, etc.).
- **Note:** A UML diagram should be languaged agnostic.

6.1 Example: UML for makefile



7 Vim Tips of the Week: Windows and Tabs

- In Vim you can view multiple files at the same time. Each view of a file is called a **window**. Here are some commands to operate on windows:

`C-w s`	splits current window into horizontal split (top and bottom windows)
`C-w v`	splits current window into vertical split (left and right windows)
`C-w c`	closes current window
`C-w <dir>`	moves the focus to the window in `<dir>`
`:e <file>`	opens `<file>` in the current window
`:sp <file>`	opens `<file>` in a horizontal split
`:vs <file>`	opens `<file>` in a vertical split

- You can also have multiple groups of windows. Each group is called a **tab**. You can manage tabs using:

`:tabe <file>`	opens `<file>` in a new tab
`:tabc`	closes the current tab
`gt`	goes to the next tab
`gT`	goes to the previous tab

- When you have multiple tabs, vim by default will show the tabs at the top of the editor. This is called the **tabbar**.
- When you close the last window in a tab, the tab will also be closed.
- If you have mouse enabled (`:set mouse=a`), you can click in window to focus on that window, and the tab name to focus on that tab. You can also drag on the separators between windows to resize them.