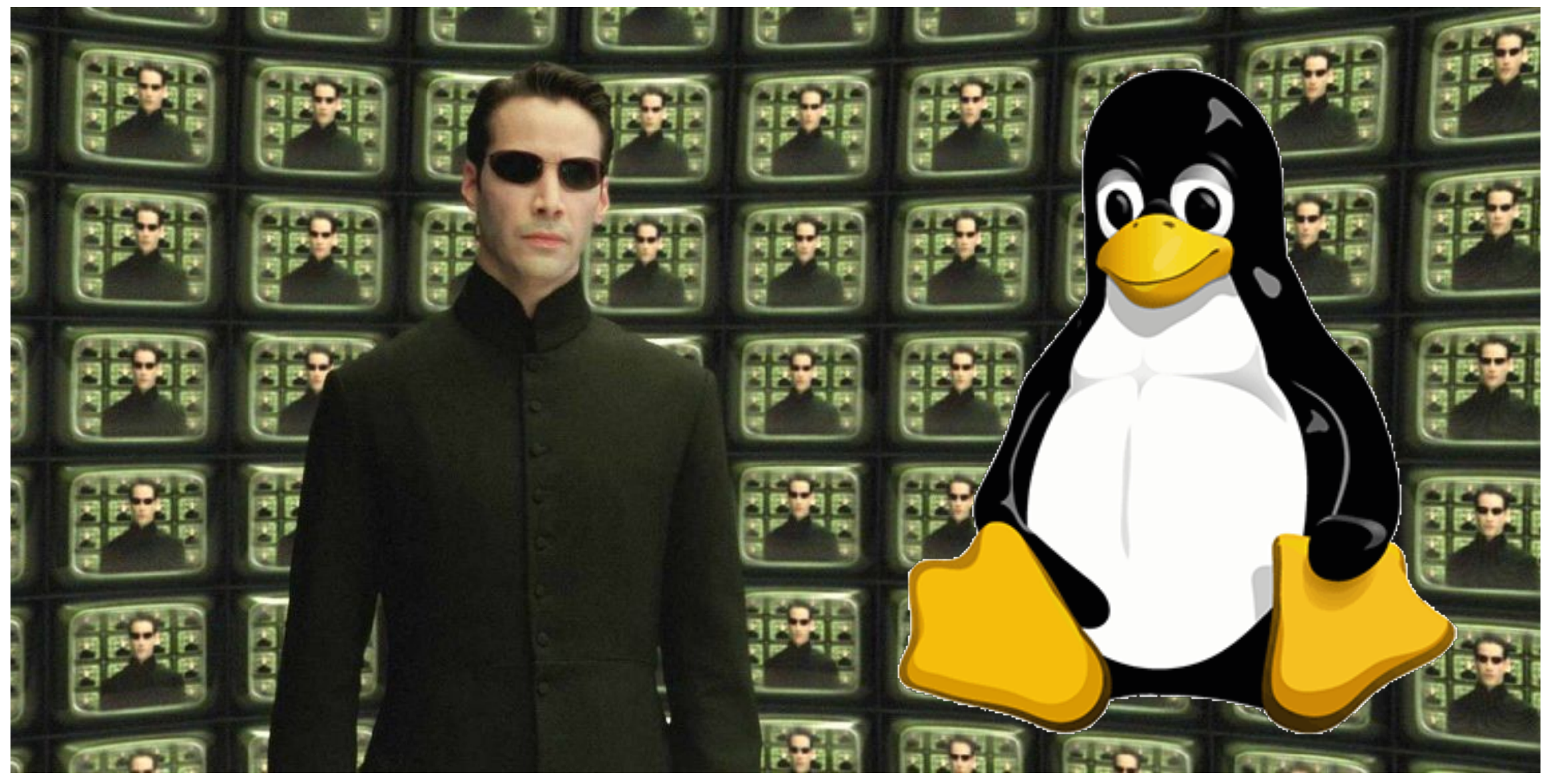
CSCI 125
Introduction to Computer Science and Programming II
Lecture 6: User Class III



Jetic Gū

2020 Summer Semester (S2)

Overview

- Focus: Basic C/C++ Syntax
- Architecture: Linux/Unix OS
- Core Ideas:
  1. Inheritance

P0
Review

C++ Class

- User defined data types
- Members: variables and function
  - access specifiers
- Constructors; Destructors
- Pointer Operations

Review

Inheritance

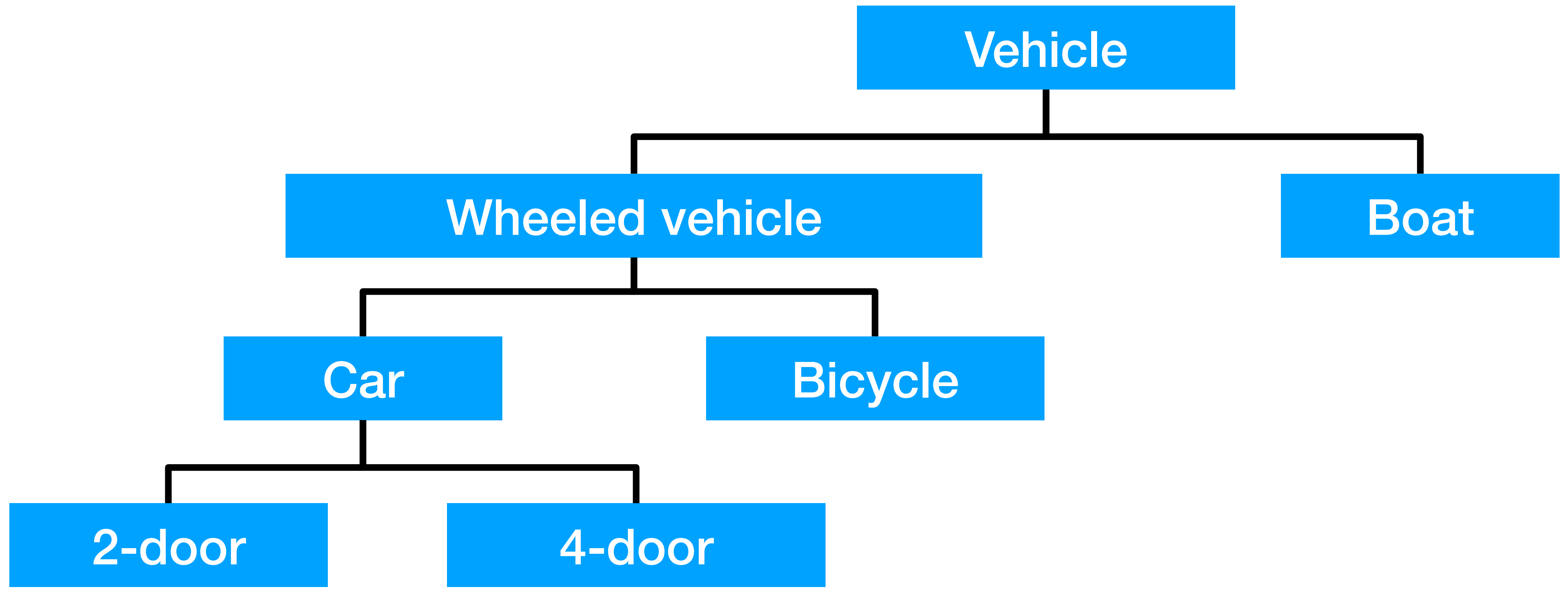
Not of any fortune though

What is inheritance?

- One of the most important features of OOP
- Not only are objects grouped together as classes, classes themselves may have commonalities
- **Inheritance**
  the mechanism by which **one class acquires the properties of another class**
  - **Base** class: provide more fundamental functions/properties
  - **Derived** class: provide more specified functions/properties

P1
Inheritance

Hierarchy



- Concepts/Classes at higher levels are more general
- Concepts/Classes at lower levels are more specific (inherit properties of concepts at higher levels)

Concept

Why Inheritance?

1. Reuse existing universal structures and methods
2. Extend existing class to incorporate new features, without sacrificing backward compatibility
3. Modify existing class by overloading member functions

C++ Inheritance

- The language mechanism by which one class acquires the properties (data and operations) of another class
- **Base Class (or superclass)**: the class being inherited from
- **Derived Class (or subclass)**: the class that inherits

C++ Inheritance

- Syntax

```
class ClassName: accessIdentifier BaseClass {
    // regular class declaration
};
```

- Addition to normal class declaration:
    - Colon
    - Access identifier : [private, protected, public]
    - BaseClass: name of the Class you want to inherit

What do you inherit? (class access specifier Public)

- `Class1`: superclass of `Class2`
- `Class2`: derived subclass of `Class1`
- What does `Class2` get?
    - `Class1`'s `public` members -> `Class2`'s `public`
    - `Class1`'s `private` members -> Accessible through `Class1`'s `public` functions

```
class Class1 {
    public:
        ...public members...
    private:
        ...private members...
};

class Class2: public Class1 {
    ....
};
```

What do you inherit? (class access specifier Public)

- `Class1`’s `public` members become `Class2`’s `public` members
- In this case, `beingEaten` can be used by all instances of `MyFood` class, anywhere

```
class Food {
    public:
        void beingEaten();
};

class MyFood: public Food {
} cheese;
cheese.beingEaten();
```

What do you inherit? (class access specifier Public)

- `Class1`'s `private` members can only be accessed by `Class1`'s `public` functions
- In this case, `getCal` can be used by all instances of `MyFood` class, anywhere
- `cal` can be accessed by `cheese` only through `getCal`

```
class Food {
   int cal; // private
   public:
      int getCal()
         {return cal;}
};
class MyFood: public Food {
      ...cal...   // no good
      ...getCal()... // works
};
```

What do you inherit? (class access specifier **Public**)

- `public` members of superclass same as `public` members of subclass
- `private` members of superclass accessible only through inherited `public` functions

Access	public	protected	private
Same class	yes	yes	yes
Derived classes	yes	yes	no
Outside classes	yes	no	no

Concept

What do you inherit? (class access specifier **Public**)

- `protected`
  - For superclass, is the same as `private` members
  - For subclass, is the same as it's own `protected` members
  - Why?
    You may have properties wanting to be available for subclasses, but not outside of the class

Access	public	protected	private
Same class	yes	yes	yes
Derived classes	yes	yes	no
Outside classes	yes	no	no

Concept

Example (class access specifier Public)

- `Class1`'s `private` members can only be accessed by `Class1`'s `public` functions
- In this case, `getCal` can be used by all instances of `MyFood` class, anywhere
- `cal` can be accessed by `cheese` only through `getCal`

```
class Food {
  int cal; // private
  public:
    int getCal()
      {return calorie;}
};
class MyFood: public Food {
} cheese;
cout << cheese.cal; // no good
cout << cheese.getCal;
```

Other Inheritance

- `protected` inheritance
  - SuperClass's `public` and `protected` members become subclass's `protected` members
  - SuperClass's `private` members remain not directly accessible
- `private` inheritance
  - SuperClass's `public` and `protected` members become subclass's `private` members
  - SuperClass's `private` members remain not directly accessible

Static Member Variables

- Member variables can be defined as `static` so that its value is shared by all instances/objects of that class
- line 6: initialising outside the class

```
1. class Dummy {
2.   public:
3.      static int n;
4.      Dummy () { n++; };
5. };
6. int Dummy::n=0;
7. Dummy a;   // n==1
8. Dummy b;    // n==2
9. cout << a.n << '\n';
10.Dummy * c = new Dummy;   // n==3
11.cout << Dummy::n << '\n';
```

P2
Future

More About Class

- Virtual members
- Static member functions
- Const member functions
- Class templates
- Class polymorphism

Future

1. For further readings, please refer to C++ reference or our textbook