Introduction to Inheritance

Outline

- **How to define classes that inherit variables and functions**
- **How to design classes and class hierarchy**

```
#include <iostream.h>
const double pi = 3.14159;
class box_car {
  public: double height, width, length;
  // Calculate the volume of a box
  double volume() {
    return height * width * length;
  }
};
 class tank_car {
     public: double radious, length;
     // Calculate the volume of a tank
     double volume() {
       return pi * radius * length;
     }
 };
 int main() {
   box_car x; x.height = 10.5; x.width = 9.5; x.length = 40.0;
   tank_car y; y.radius = 3.5; y.length = 40.0;
   cout << “The volume of the box car is “ << x.volume() << endl
          << “The volume of the tank car is “ << y.volume() << endl;
 } // end function main
```

```
The volume of the box car is 3990
The volume of the tank car is 1539.38
```


Recall our box_car and tank_car class examples

Add Information Common to Classes

- We defined two railroad cars: box_car and tank_car
- Want to add information common to all railroad cars
- One way is simply adding it to box_car and tank_car

```
int current_year = 2001;
class box_car {
  public:
    // From a previous definition of the box_car class:
    double height, width, length;
    box_car () {height = 10.5; width = 9.2; length = 40.0;}
    double volume () {return height * width * length;}
    // New member variables:
    int percentage_loaded;
    int year_built;
    // New member function; relies on current_year, a global variable
    int age () {return current_year - year_built;}
};
```

```
class tank_car {
  public:
  // From a previous definition of the box_car class:
  double radius, length;
  tank_car() {radius = 3.5; length = 40.0;}
  double volume() {return pi*radius*radius*length;}
  // New member variables:
  int percentage_loaded;
  int year_built;
  // New member function; relies on current_year, a global variable
  int age () {return current_year - year_built;}
};
```

- What can be the problem?
  - Needless duplication of `percentage_loaded, year_built, age()`
  - Probably need to duplicate in other railroad classes: engine and caboose

Railroad Car Examples

Box car

Tank car

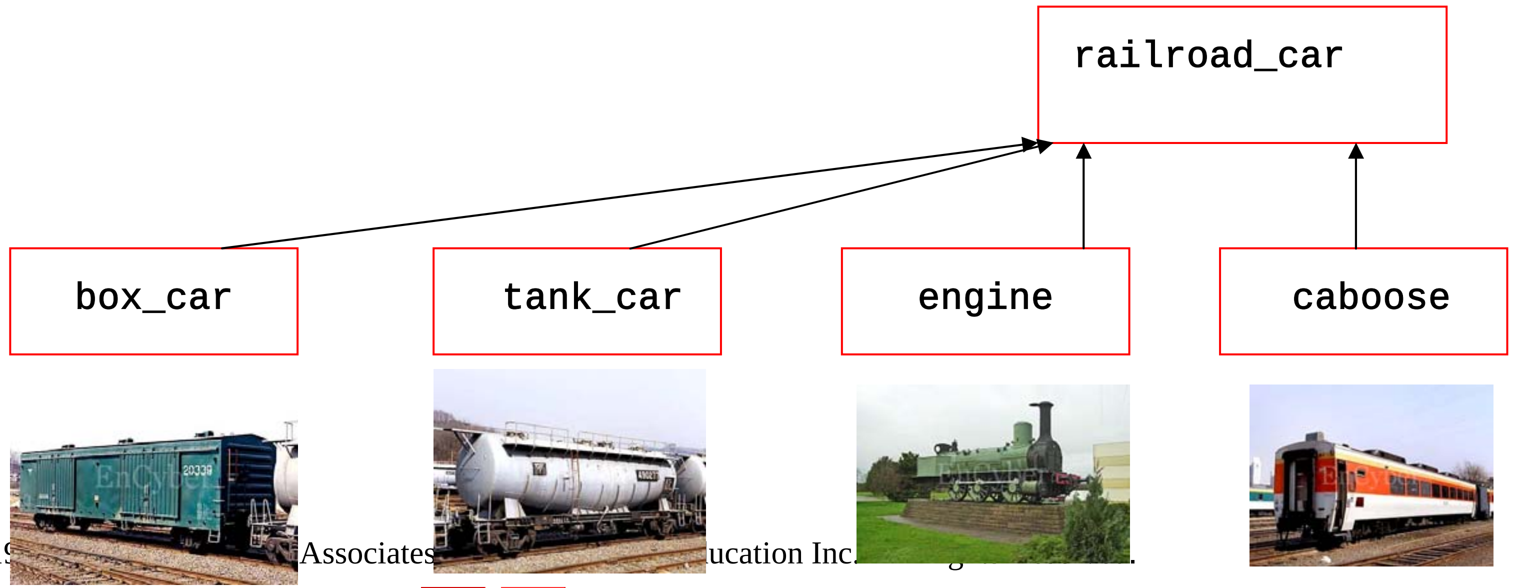
Engine

Caboose

Pictures are from naver.com 이미지 검색

Defining a Super Class

- In C++, we can avoid the duplication via inheritance
  - We can say box car, tank car, engine, and caboose are railroad cars
  - We define a `railroad_car` class where we declare common items
  - Then we define `box_car` class as a subclass of `railroad_car` class
    - Common items will be inherited from `railroad_car` to `box_car`
    - Same for `tank_car`, `engine`, and `caboose` classes



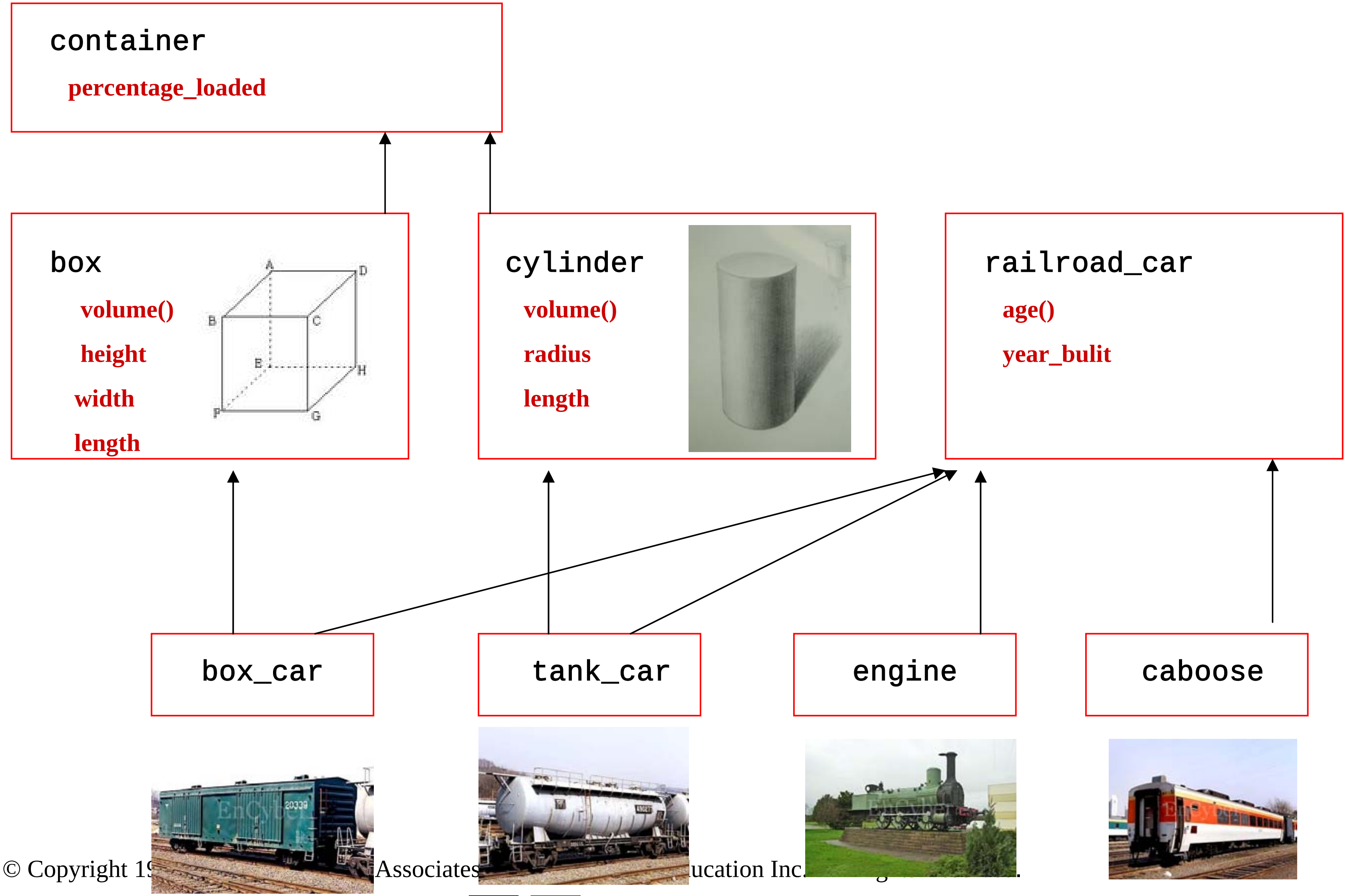
Defining a Super Class

- We need to think more about inheritance
  - We can also say a box car **is** a box and a tank car **is** a cylinder
    - where **is** is different when we say that a box car **is** a railroad car
      - **is** means **is a kind of** while **is** means **usefully can be viewed as**
  - Why do you want to view like this?
      - When introducing box and cylinder classes avoid duplication
      - When you have fully-debugged box and cylinder classes

Our Example Class Hierarchy

- We assume that a box car **usefully can be viewed as** a box and a tank car **usefully can be viewed as** a cylinder
  - `box_car` is a subclass of `box`
  - `tank_car` is a subclass of `cylinder`
- Also, box and cylinder are a kind of container
  - `box` and `cylinder` are subclasses of `container`
- `box_car` and `tank_car` are subclasses of `railroad_car`
- Then we rearrange member variables and functions
  - Move height, width, length, volume() from `box_car` to box class
  - Move radius, length, volume() from `tank_car` to cylinder class
  - Declare age() and year_built in `railroad_car` class
  - Declare percentage_loaded in `container` class

Our Example Class Hierarchy Diagram



Inheriting Member Variables and Functions

- Our class hierarchy shows multiple inheritance
  - Not supported in Java
- Class objects inherit member variables and functions
  - A box_car object has its own copy of all variables declared in
    - box_car, box, container, railroad_car
  - We can work on a box_car object with all functions declared in
    - box_car, box, container, railroad_car
- Criteria for placing member variables and functions
  - No needless duplication
  - Each variable and function should be useful in all subclasses

Defining Class Hierarchy using Inheritance

```
int current_year = 2001;

class container {
  public: int percent_loaded;
          // Default constructor:
          container () {}
};

class railroad_car {
  public: int year_built;
          // Default constructor:
          railroad_car () {}
          // Other member function:
          int age () {return current_year - year_built;}
};
```

Defining Class Hierarchy using Inheritance

```
class box : public container {
  public: double height, width, length;
          // Default constructor:
          box () {}
          // Other member function:
          double volume () {return height*width*length;}
};

const double pi = 3.14159;

class cylinder : public container {
  public: double radius, length;
          // Default constructor:
          cylinder () {}
          // Other member function:
          double volume ()
          {return pi*radius*radius*length ;}
};
```

Defining Class Hierarchy using Inheritance

```
class box_car : public railroad_car, public box {
  public: box_car ()
          {height = 10.5; width = 9.2; length = 40.0;}
}

class tank_car : public railroad_car, public cyliner {
  public: tank_car ()
          {radius = 3.5; length = 40.0;}
}

class engine : public railroad_car {
  public: engine () {}
}

class caboose : public railroad_car {
  public: caboose () {}
}
```

Calling Constructors in Class Hierarchy

- When you create an object in a class hierarchy,
  - All the default constructors in the object's class and superclasses are called automatically
  - Superclasses' constructors are called earlier than subclasses'

```
#include <iostream.h>
int current_year = 2001;
class container {
  public: int percent_loaded;
          // Default constructor:
          container () {
           cout <<
               "Calling container default
   constructor." << endl;
          }
};
```

Calling Constructors in Class Hierarchy

```
class box : public container {
  public: double height, width, length;
          // Default constructor:
          box () {
            cout << "Calling box default constructor." << endl;
          }
          // Other member function:
          double volume () {return height * width * length;}
};
// Cyliner definition goes here ...

class railroad_car {
  public: int year_built;
          // Default constructor:
          railroad_car () { cout <<
             "Calling railroad_car default constructor." << endl;
          }
          // Other member function:
          int age () {return current_year - year_built;}
};
```

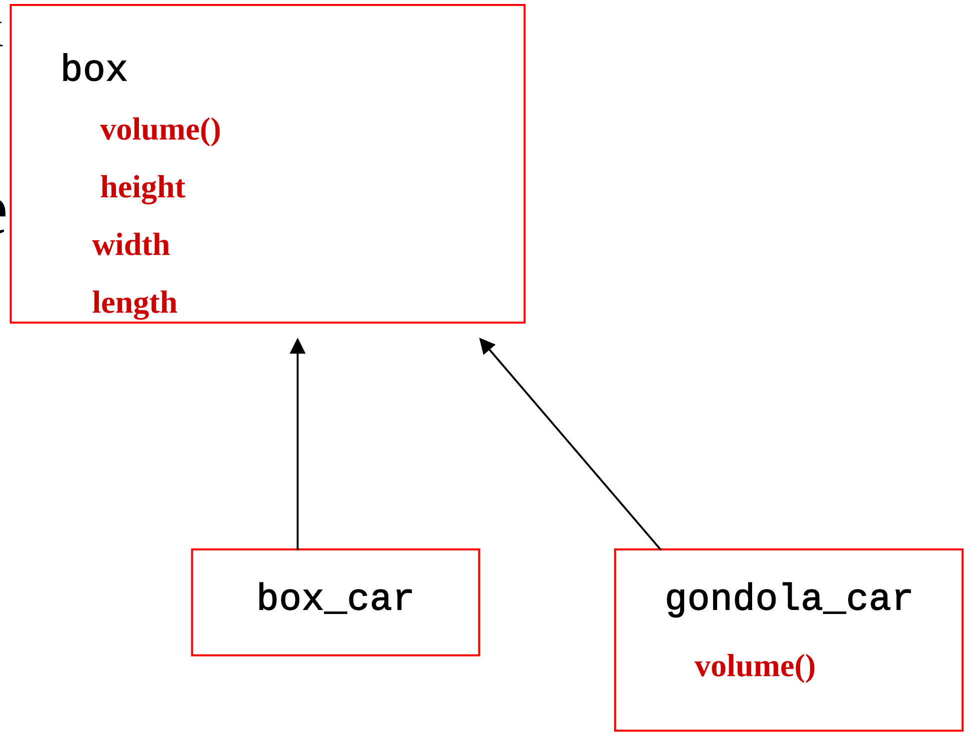
Calling Constructors in Class Hierarchy

```
class box_car : public railroad_car, public box {
  public: box_car () {  cout <<
              "Calling box_car default constructor." << endl;
               height = 10.5; width = 9.2; length = 40.0;}
          };
main() {
  box_car b;
  b.year_built = 1943;  b.percent_loaded = 66;
  cout << "The car is " << b.age() << " years old." << endl;
  cout << "And " << b.percent_loaded << " percent loaded."
   << endl;
  cout << "Its volume is " << b.volume () << " units." << endl;
}
```

```
Calling railroad_class default constructor
Calling container default constructor
Calling box default constructor
Calling box_car default constructor
The car is 58 years old
And 66 percentage loaded
Its volume is 3864 units
```

Member Function Overriding

- Suppose we a `gondola_car` class, a subclass of `box` class
  - whose `volume()` is computed differently, so is defined in `gondola_car`
  - There are two volume() in class hierarchy for a `gondola_car` object
  - Which one to choose? the one in `gondola_car`
    - Which shadows the one in box
  - This is called overriding
- Can also call shadowed one
  - By calling `box::volume()`



Function Overriding Example Code

```
class gondola_car : public railroad_car, public box {
  public: // Default constructor:
          gondola_car () {height=6.0; width = 9.2; length = 40.0;}
          // The gondola volume member function;
          // gondolar cars are loaded above their rims:
          double volume () {return 1.2 * height * width * length;}
};

main() {
  // Construct a gondola car:
  gondola_car g;
  // Display volume; use the gondola class volume function:
  cout << "Viewd as a gondola, the car's volume is "
       << g.volume () << " units." << endl;
  // Display volume; use the box class volume function:
  cout << "Viewd as a box, the car's volume is "
       << g.box::volume () << " units." << endl;
}
```

How to Design Classes and Class Hierarchies

- Principles to design classes and hierarchies
  - Explicit representation principle
    - There should be a class corresponding to a natural category
  - No-duplication principle
    - Avoid duplication of identical code
  - Local-view principle
    - Related program elements should be located close in the code
  - Look-it-up principle
    - Frequently-needed answer must be a variable, not a computed one

How to Design Classes and Class Hierarchies

- Need-to-know principle
  - Restrict access to public interfaces
- Keep-it-simple principle
  - Class definition should be easier to read (e.g., less than 20 lines)
- Modularity principle
  - Related classes should be in a file