Object Oriented Programming in C++

Nepal College of Information Technology

Course Instructor: Er. Rabina Chaudhary

Chapter 4:Inheritance

Subclass, Subtype and Substitutability

The relationship of the data type associated with a parent class to the data type associated with a child class gives rise to the following arguments:

i. Instances of the subclass must possess all data areas associated with parent class
ii. Instances of the subclass must implement thru inheritance at least all functionality defined for the parent class (They can also define new functionality)
iii. An instance of a child class can mimic the behavior of the parent class and should be indistinguishable from an instance of the parent class if substituted in a similar situation

Principal of Substitutability :

- “If we have two classes A and B such that class B is a subclass of A, it should be possible to substitute instances of class B for instances of class A in any situation with no observable effect”

Example program for Principal of Substitutability :

```
#include<iostream.h>
#include<conio.h>
class A
{
        public:
        void printMsg()
         {
                cout <<" I am A!!";
        }
};
class B : public A
{
        public:
        void printMsg()
        {
        cout <<" I am B!!";
        }
};
```

Example program for Principal of Substitutability :

```
void testingA(A  a)
{
        a.printMsg();
}
void testingFunc()
{
        B b;
        cout<<" Passing B"<<endl;
        testingA(b);        // b substituted for object of A.
}

void main()
{
        testingFunc();
        getch();
}
```

Subclass:

- A class which inherits the characteristics and behavior of another class is called a subclass.

Subtype:

- Subtype is subclass relationship in which the principle of substitutability is maintained

Forms of Inheritance:

Description of categorization of the uses of inheritance

1. Inheritance for Specialization
2. Inheritance for Specification
3. Inheritance for Construction
4. Inheritance for Generalization
5. Inheritance for Extension
6. Inheritance for Limitation
7. Inheritance for Variance
8. Inheritance for Combination

1. Inheritance for Specialization:

- Child class is specialized form of Parent class
- Child class satisfies the specification of Parent in all relevant respects
- Child class inherits the properties of the Parent class and has its own additional features also
- Child class can override methods from the Parent in order to specialize the Child class in some way

2. Inheritance for Specification:

- The Parent class is just used to provide behavior or which function should be available but doesn’t provide its implementation
- Parent class is abstract class and just provides specification for child class
- The derived class have the implementation of the function respective to their requirement

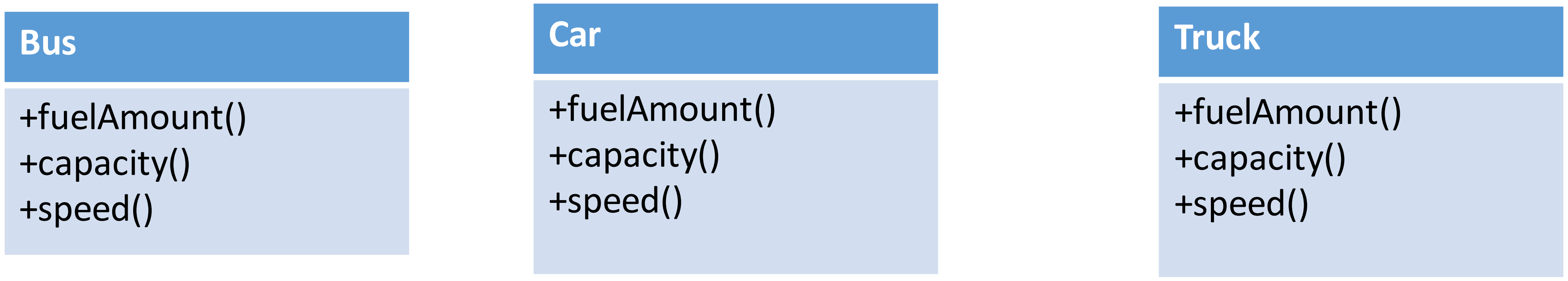
3. Inheritance for Construction:

- If the Parent class is used as a source for the behavior but the child has no is-a relationship to the parent, then the child is using inheritance for construction

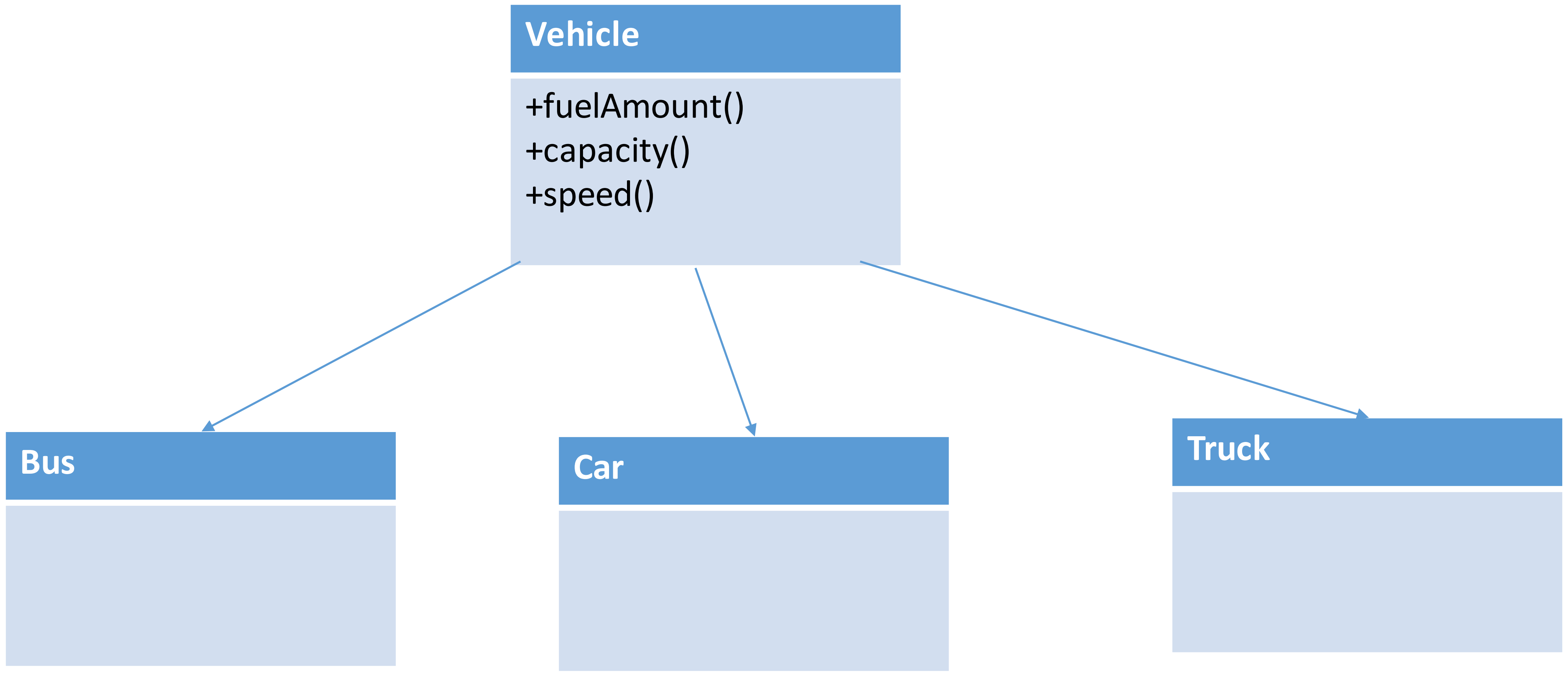
4. Inheritance for Generalization:

- Subclass extends the behavior of Superclass to create more general kind of object but doesn’t override any method with completely new features

Without using inheritance:



Using inheritance for Generalization:



5. Inheritance for Extension:

- Extension simply adds new methods in the Child to those of the Parent

Inheritance for Limitation:

- Inheritance for limitation is used when the behavior of subclass is smaller or more restrictive than the behavior of the Parent class
- The programmer can override the undesired methods inherited from Parent class so that they can limit the features
- The methods override existing methods and eliminate or limit their functionality

7. Inheritance for Variance:

- Used when two or more classes have similar implementations but do not seem to possess any hierarchical relationships between concepts represented by the classes
- e.g. code required to control a mouse and a tablet are identical
- In this case one of the two classes can be chosen as the parent by inheriting the common code and overriding the device specific code

8. Inheritance for Combination:

- The subclass represents a combination of features from two or more parent classes
- The ability of a class to inherit from two or more parent classes is known as multiple inheritance

Advantages of inheritance:

- Software Reusability : The child class need not rewrite the behavior inherited from its parent class
- Code Sharing : many user or projects can use the same class, code sharing through inheritance
- Software Components : Inheritance facilitates building of reusable software components
- Rapid Prototyping : usage of reusable components cuts down time required for coding which can be spent on understanding the new and unusual portions of the system. Thus software can generated quickly and early leading to a style of programming called Rapid Prototyping.
- Information Hiding : Programmer reusing a software component needs only to understand the functionality of the component and its interface to reuse it

IS – A relationship:

- In OOP “is-a” relationship is a relation between two classes where one class is a specialized form of second class
- X is a specialized instance of concept Y if the assertion in plain English “X is a Y” sounds correct or logical
- Examples of inheritance satisfying the “is-a” relationship
    - student is a person, apple is a fruit, car is a vehicle
- The concept of “is-a” relationship is represented by Inheritance

HAS - A relationship:

- If the one concept is a component of another concept then there exist “has-a “ relationship
- the two concepts are not the same thing
- e.g. a car has a engine , fire station has fire fighter
- Also called a “part of” relationship e.g. an engine is a part of car

- Testing the relationship:
  - Form a simple sentence “ X has a Y”,
  - if the result sounds reasonable then the relationship hold

Example of “Car has a engine”:

```
class Engine
	{
	};
class Car
	{
	Engine e;    // a car is composed of an engine
	};
```

Example program of “has-a “ relationship:

```
#include<iostream.h>
#include<conio.h>
class Engine
{
        public:
          void startEngine()
                {
                        cout<<" Engine started"<<endl;
                }

        void stopEngine()
        {
                cout<<" Engine Stopped"<<endl;
        }
};
```

Example program of “has-a “ relationship:

```
class Car
{
        Engine e;
        public:
        void startCar()
        {
                e.startEngine();
                cout<<endl<<" The car is moving"<<endl;
        }
        void stopCar()
        {
                e.stopEngine();
                cout<<endl<<" The car has stopped"<<endl;
        }
};
```

Example program of “has-a “ relationship:

```
void main()
{
        Car car;
        car.startCar();
        cout<<endl<<endl<<endl;
        car.stopCar();

        getch();
}
```

Has-a relationship:

Here, since one object is composed of or contained in another one, this relationship is also called containership.

This relationship between classes is also referred to as aggregation.

Q1. Define a class **Address** that has attributes country, province number, municipality name and ward number. Define a class Student with attributes relative to student. Write a program to use aggregation to represent “Student has Address”.