Object-Oriented Programming

Lecture 5

Agata Półrola
<agata.polrola@wmii.uni.lodz.pl>

Inheritance

- enrichment of existing classes with additional functionality
- ability of modifying the action of methods without modifying the existing old code
- better team support with the code separation
- different types of the inheritance from a base class:
  - **public** – all the public members of the base class become public in derived classes, all the protected members of the base class become protected in derived classes
  - **protected** – all the public and protected members of the base class become protected in derived classes
  - **private** – all the public and protected members of the base class become private in derived classes (private is the default inheritance type)
  - **virtual** – when inheriting from many base classes

Base class

An example of a base class Vehicle:

```
class Vehicle
{
 protected: //accessed by inheriting classes
  int num, wheels;

 public: //public interface
  Vehicle(int _num = 0, int _wheels = 4)
  {
    num = _num; wheels = _wheels;
  }
  void print() const
  {
    cout << "number: " << num << endl;
    cout << "wheels: " << wheels << endl;
  }
};

Vehicle v(1);
v.print(); //public method call
v.wheels = 4; //compile-time error, protected data
```

Derived class

The Car class derived from the base class Vehicle:

```
class Car: public Vehicle //public inheritance
{
 protected: //accessed by inheriting classes
  string brand;
  float capacity; //fields extending the structure

 public: //public interface
  Car(int _num = 0, float _cap = 0, string _brand = ""):
  //initialisation list
  Vehicle(_num, 4), capacity(_cap), brand(_brand) {}
  void print() const //overriden method
  {
    Vehicle::print(); //call of inherited version
    cout << "brand: " << brand << endl;
    cout << "capacity: " << capacity << endl;
  }
};

Car c(2, 2.0, "Ford");
c.print(); //public method call
```

Initialisation list of constructor

- the only way to initialise the derived part (base part) of the derived class
- placed after the constructor argument list and a colon (:)
- the constructor of the base class must be called before the constructor body and it must be the first item of the initialisation list (if ommited, the default constructor will be called implicitly)
- the list may contain also initialisations of any other member fields/objects (copy-constructors are called)

```
Car(int _num = 0, float _cap = 0, string _brand = ""):
  Vehicle(_num, 4), //base class constructor
  capacity(_cap), //member initialisation
  brand(_brand) //member object copy-constructor (for string)
{
  //the body of the constructor
}
```

Rules of inheritance

- objects are constructed starting with the base part and ending with the most-derived part
- objects are destructed starting with the most-derived part and working back to the root (the main base class)
- static members are derived like non-static
- non-derived members: constructors, destructors, the assignment operator (automatically created by the compiler if not defined)
- the copy-constructor calls the derived one by default

```
Car(const Car &c):
  Vehicle(c), //base class copy-constructor
  capacity(c.capacity),
  brand(c.brand)
{
  //the body of the constructor
}
```

Upcasting

- any object of a derived class can be treated as an object of the base class (the reverse implication is false)
- pointers or references to objects of derived classes are also pointers or references to the base class objects
- typical for copy-constructors

```
void repair(Vehicle &v)
{
  //the body
  v.capacity = 3.0; //compile-time error, v has no capacity
  v.print(); //call of Vehicle::print method
}

Car myCar(123, 4.5, "Porsche");
repair(myCar) ; //upcasting
```

Inheritance vs. composition

- the inheritance is used to extend the capabilities of an existing class preserving its common interface
- the composition is used to include the features of an existing class, but not its interface
- use the inheritance to express the «is-a» relationship between classes, i.e. a car is a vehicle, a truck is a car, a student is a person, etc.
- use the composition to express the «has-a» relationship between classes, i.e. a car has a brand, a person has a name, a driver has a car, etc.
- use the composition instead of the private inheritance to hide the interface of an existing class
- do not create very long chains of inherited classes, 3-4 levels of the inheritance are enough (more are inefficient)