

Section 2: Lecture 3

C++ Concepts

Introduction

- Introduction to Objects and Object Oriented Programming, Encapsulation (Information Hiding)
Access Modifiers: Controlling access to a class method/variable (public, protected, private, package),
Other Modifiers,
Polymorphism: Overloading,
Inheritance,
Overriding Methods,
Abstract Classes,
Reusability,
Class's Behaviors.

What is Object Oriented Programming?



An object is like a
black box.

The internal details
are hidden.

- Identifying *objects* and assigning *responsibilities* to these objects.
- Objects communicate to other objects by sending *messages*.
- Messages are received by the *methods* of an object

The two steps of Object Oriented Programming

- *Making Classes*: Creating, extending or reusing abstract data types.
- *Making Objects interact*: Creating objects from abstract data types and defining their relationships.

Example: The Creature class

```
Class Creature {  
private:  
    int yearOfBirth;  
public:  
    void setYearOfBirth(year) {  
        yearOfBirth = year;  
    }  
    int getYearOfBirth() {  
        return yearOfBirth;  
    }  
};
```



Example: The Creature class

```
class Creature {  
private:  
    int yearOfBirth;  
public:  
    void setYearOfBirth(year) {  
        yearOfBirth = year;  
    }  
    int getYearOfBirth() {  
        return yearOfBirth;  
    }  
};
```

The definition of a class:

- The *class* keyword, followed by the class name.
- *private* attributes.
- *public methods*.
- the ; at the end

Example: The Creature class

```
class Creature {  
private:  
    int yearOfBirth;  
public:  
    void setYearOfBirth(year) {  
        yearOfBirth = year;  
    }  
    int getYearOfBirth() {  
        return yearOfBirth;  
    }  
};
```

This class has two (public) methods. One to set the attribute value and the other to retrieve the attribute value.

Example: The Creature class

```
class Creature {  
private:  
    int yearOfBirth;  
public:  
    void setYearOfBirth(year);  
    int getYearOfBirth();  
};  
void Creature::setYearOfBirth {  
    yearOfBirth = year;  
}  
int Creature::getYearOfBirth() {  
    return yearOfBirth;  
}
```

Note that unless the methods are very short, declaration and implementation is usually separated.

The declaration goes into a header file (.h), the implementation in a .cpp file.

Example: The Creature class

```
class Creature {  
private:  
    int yearOfBirth;  
public:  
    void setYearOfBirth(year) {  
        yearOfBirth = year;  
    }  
    int getYearOfBirth() {  
        return yearOfBirth;  
    }  
};
```

This method is an example for a 'modifier' method. It modifies the attribute. The method *changes the state* of the object.

Example: The Creature class

```
class Creature {  
private:  
    int yearOfBirth;  
public:  
    void setYearOfBirth(year) {  
        yearOfBirth = year;  
    }  
    int getYearOfBirth() {  
        return yearOfBirth;  
    }  
};
```

This method is an example for a 'selector' method. It returns information about the attribute but does not change the state of the object.

Classes & Objects

- What may be different for all objects in a class, and what remains the same?
- *All the objects in a class may have different attribute values (state data), but their allowed behaviours are all the same.*

So a class is a blueprint for objects

Objects & Classes

- A class is defined by:
 - A Unique Name
 - Attributes
 - Methods
- An object is defined by:
 - Identity
 - State
 - Behaviour

Instantiating Objects

- An object is instantiated just like any other data type:

```
int x;  
char y;  
Creature z;
```

Declaring z of type 'creature' means we have generated an object with the attributes and methods of the class.

Multiple Objects

- Of course we can create many objects of the same class:

Creature myDog;

Creature theMilkman;

Creature myBestFriend;

Creates three objects.

Sending Messages / Calling Methods.

- A message is send to an object by calling a method of this object. Use the . (dot) for calling a method of an object.

```
int k;
```

```
k = theMilkman.getYearOfBirth();
```

```
myDog.setYearOfBirth(1998);
```

Messages are sent to my dog
and the milkman.

Back to the Instantiation...

- An object is instantiated just like any other data type:

```
int x;
```

```
char y;
```

```
Creature z;
```

Here the “default constructor” of the Creature class is automatically called.

If we don't like this we can specify constructors explicitly!

The Creature class with a user defined default constructor.

```
class Creature {  
private:  
    int yearOfBirth;  
public:  
    // ...  
    Creature() {  
        yearOfBirth = 1970;  
        cout << "Hello.";  
    }  
};
```

The syntax for a constructor is similar as for a method, but:

- It has the same name as the class.
- It has no return value.

The Creature with a parametrized constructor.

```
class Creature {  
private:  
    int yearOfBirth;  
public:  
    // ...  
    Creature(int year) {  
        yearOfBirth = year;  
    }  
};
```

This constructor can be used as follows:

```
Creature theMilkman(1953);
```

instantiates a 49 years old milkman.

The Creature with a copy constructor.

```
class Creature {  
private:  
    int yearOfBirth;  
public:  
    // ...  
    Creature(Creature & otherCreature) {  
        yearOfBirth =  
            otherCreature.getYearOfBirth();  
    }  
};
```

Example:

```
Creature myDog(1995);  
Creature myCat(myDog);
```

creates a cat of the same age as the dog.

Constructors - summary

- A constructor is always called when an object is created.
- We can define our own constructors (Note: a class can have more than one constructor).
- If an object is copied from another object then the copy constructor is called.

Again:

Objects & Classes

- A class is defined by:
 - A Unique Name
 - Attributes
 - Methods
- An object is defined by:
 - Identity
 - State
 - Behaviour

Again:

Objects & Classes

- A class is defined by:
 - A Unique Name
 - Attributes
 - Methods
- An object is defined by:
 - Identity
 - State
 - Behaviour

But: We can give a class state and behaviour with the keyword `static`!

Example: The Creature class

```
class Creature {  
private:  
    int yearOfBirth;  
    static int numberOfAllCreatures = 0;  
public:  
    Creature() { // Constructor - counts the creatures.  
        numberOfAllCreatures++;  
    }  
    static int getNumberOfAllCreatures() {  
        return numberOfAllCreatures;  
    }  
};
```

Note that all objects share the same value of the “class attribute” `numberOfAllCreatures`.

Summary.

- A class is a blueprint for an object.
- Objects are created similar to other data types (int, char, ...).
- The construction of an object can be defined by the user.
- Messages are sent to an object by calling a method.
- *static* messes the concept of classes and objects (but is nevertheless useful).