Important notice: the final exam will be paper based at campus (closed book)

UNIVERSITY
OF WOLLONGONG
AUSTRALIA

CSCI251
Advanced Programming
OOP in C++

Outline

Object Oriented Programming in C++

OOP in C++

1. What is it?
2. Why OOP?

Class and Object

3. Create a C++ Class
4. Generate a C++ Object
5. Attributes, Method

Encapsulation

6. Struct and Class
7. Public, Private, and Protected

Object Management

8. Constructors
9. *Destructors

Overloading


1. C++ OOP

What is OOP?

Concept: Object-oriented programming (OOP) refers to a type of computer programming (software design) in which programmers define the data type of a data structure, and also the types of operations (functions) that can be applied to the data structure.

Source:
https://www.webopedia.com/TERM/O/object_oriented_programming_OOP.ht

Why OOP?

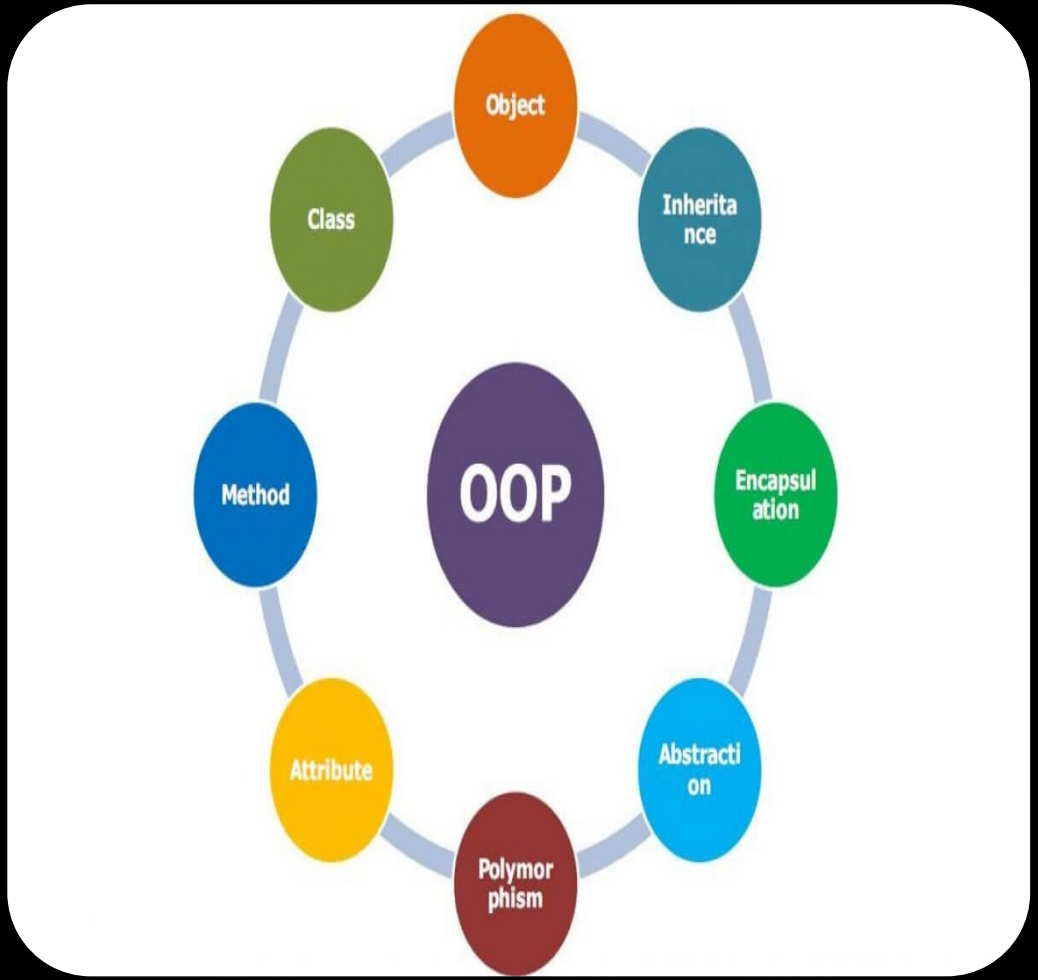
- To create new type of data structure
- Clear structure for the programs
- To reflect things in the real world.
- Faster and easier to execute
- Keep code DRY “Don’t Repeat Yourself”

Source: https://www.w3schools.com/cpp/cpp_oop.asp

OOP Components?

Components relating to OOP

- Object
- Class
- Attributes
- Method
- Abstraction
- Encapsulation
- Inheritance
- Polymorphism




2. Class and Object

What is C++ Class?

Class is most important in OOP in C++.

Class is the new type of data structure that includes attributes and methods to reflect things in the real world.

What is C++ Class?

A class is a user defined data type:

- provide a description for building objects.
- provide prototypes or blueprints for objects.
- provide a way to group related data and the functions which are used to process the data
- you create an object from the class, you automatically create all the related fields

Abstract data type (ADT): a type you define.

Why we use Class?

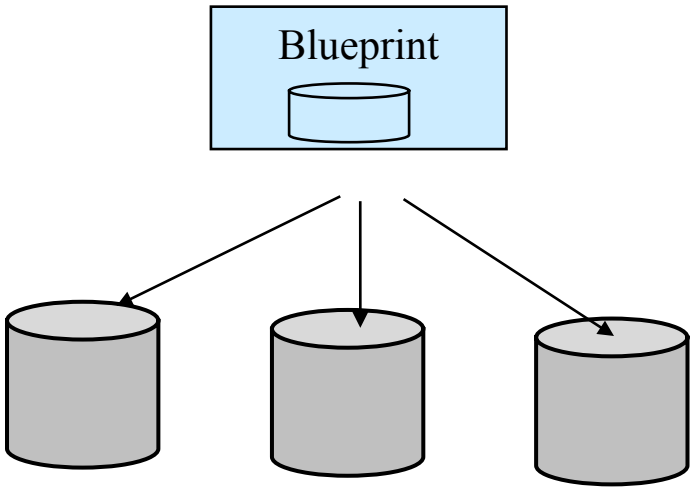
- To create new type of data structure
- To reflect things in the real world.
- To model real world problem.

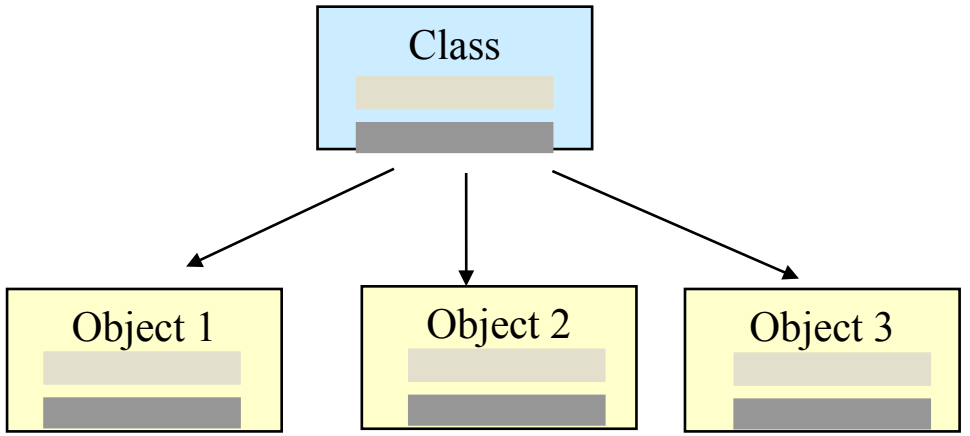
Source: https://www.w3schools.com/cpp/cpp_oop.asp

Class and Objects

Parts of a technical system are produced based upon their description: blueprints.

Parts of a software system are generated based upon their description: classes.

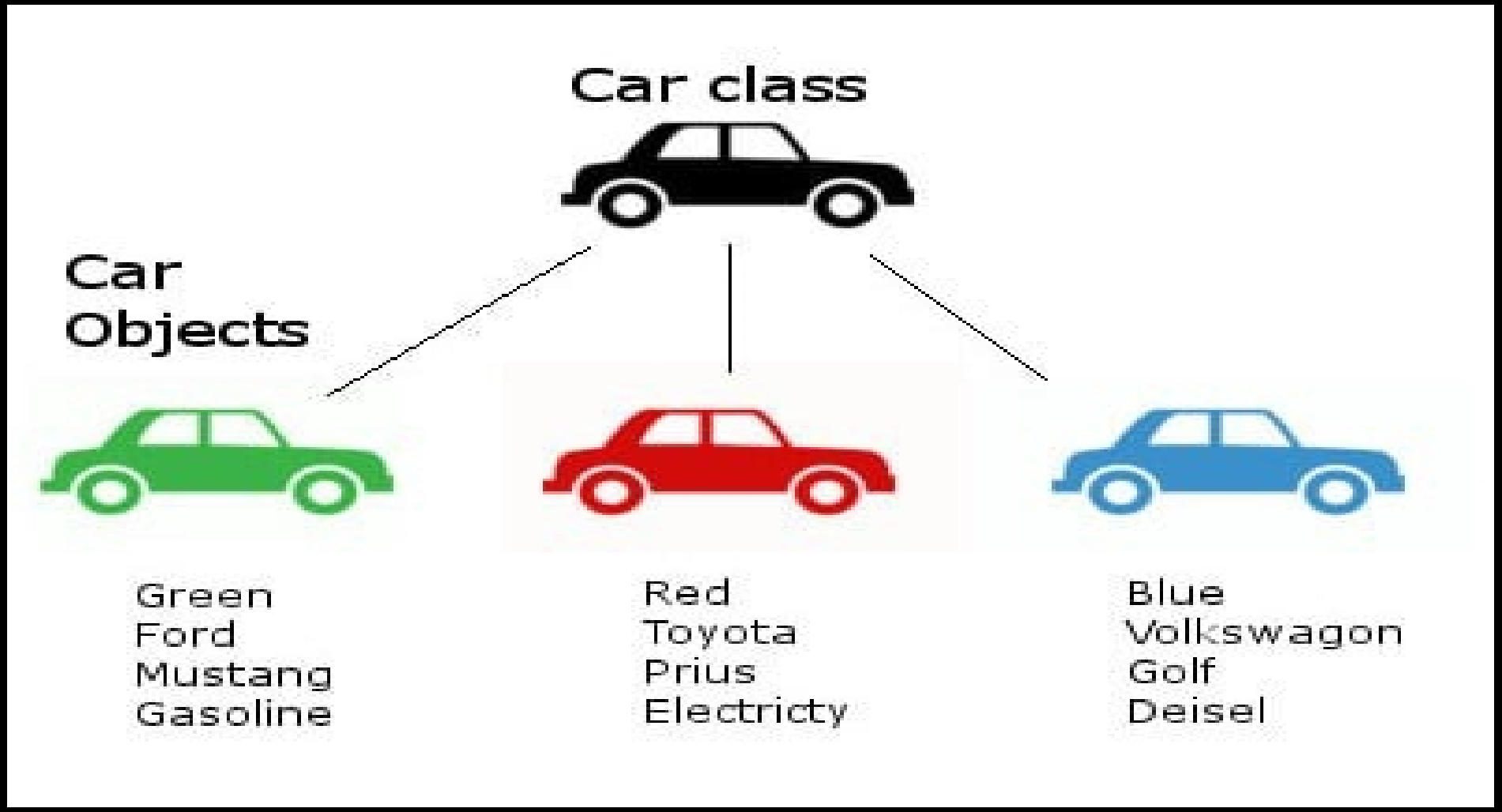




- A class is an abstraction, it is an abstract data type.
- An object is an instance generated and placed in computer memory.

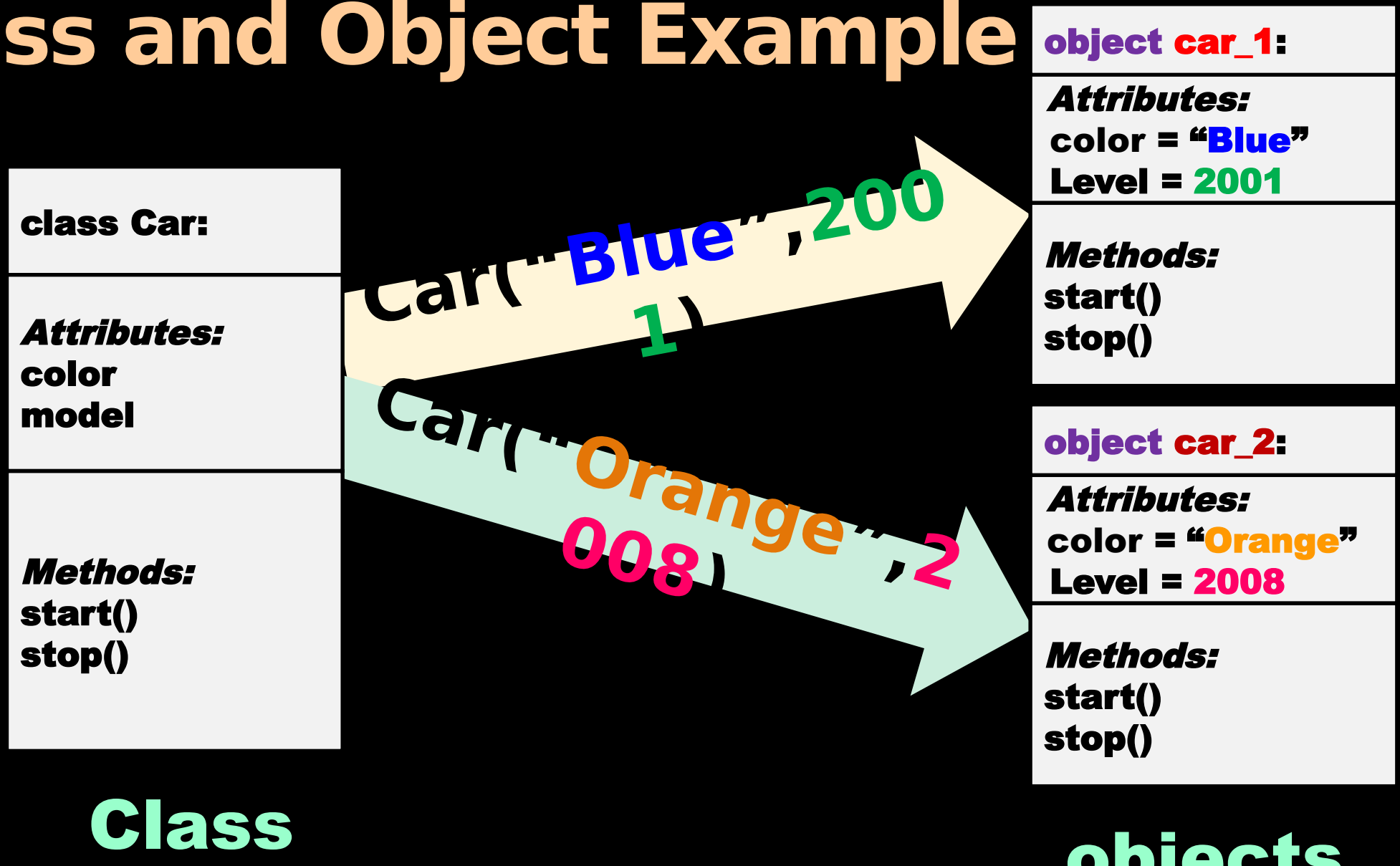
Important notice: the final exam will be paper based at campus (closed book)

Class Example

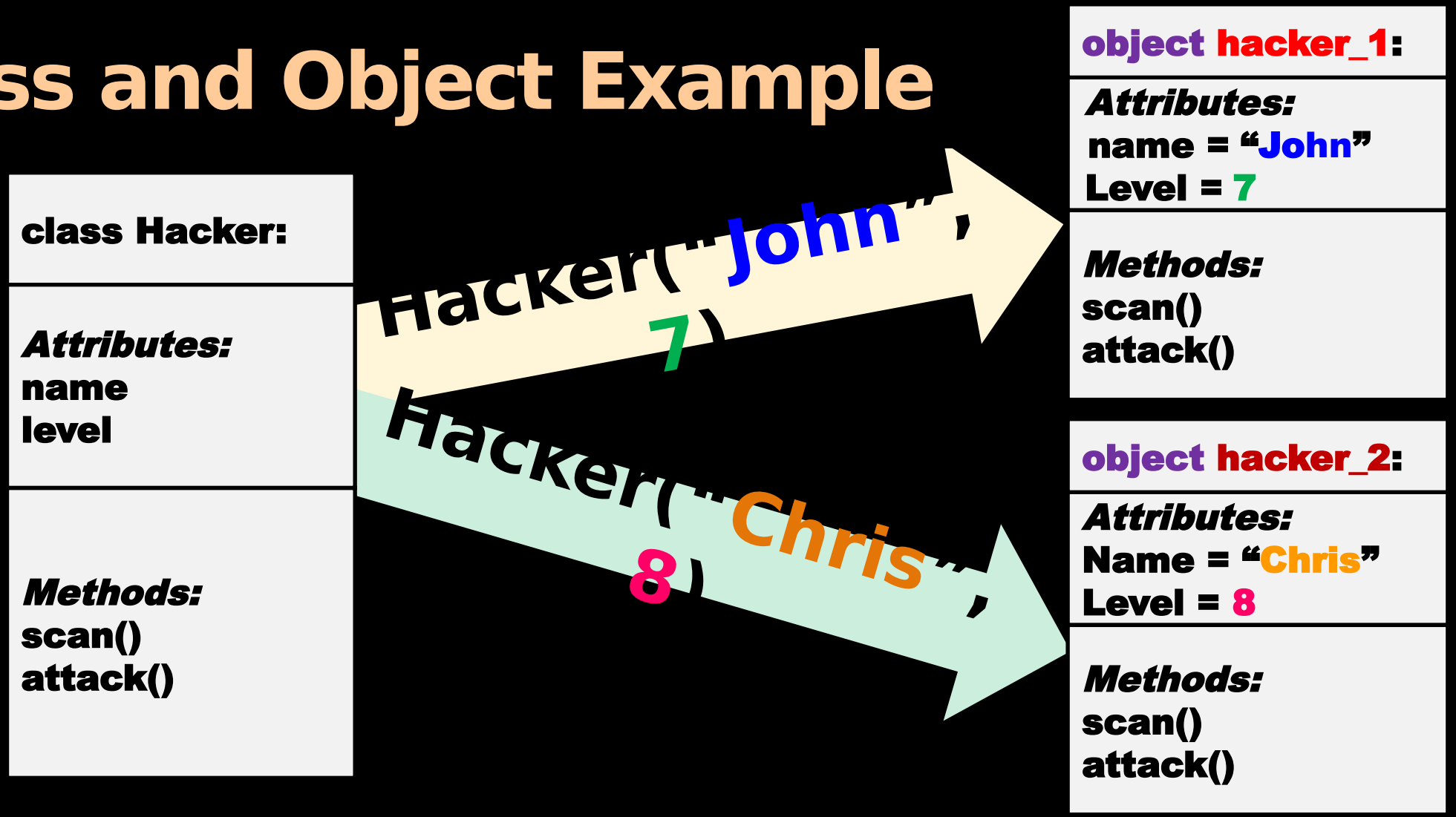


https://medium.com/pongpichs/oop-object-oriented-programing-4dda39dbb745

Class and Object Example



Class and Object Example



Class **objects**

What is object?

An object is an instance of a class.

Source: https://www.w3schools.com/cpp/cpp_oop.asp

How to create C++ Class?

To create a C++ Class

- Syntax:

```
class Class_name{     // The class name
  public:             // Access specifier
    int variable; // Attributes
};                      // Semicolon
```

How to create C++ Class?

To create a Hacker Class with attributes

```
class Hacker {    // The class Hacker
  public:         // Access specifier
    string name;// Attribute (string)
    int level;  // Attribute (int)
};
```

class Hacker:
Attributes: **name** **level**

Instantiate C++ object

To create C++ object

- Syntax: Class_name object;

```
Hacker hacker_1; // Create an Object of Hacker
```

Instantiate C++ object

Set the values and process data

object hacker_1:

Attributes:
name = “John”
level = 7

```
#include <iostream>
using namespace std;

class Hacker {       // The class Hacker
  public:            // Access specifier
    string name;     // Attribute (string variab
le)
    int level;       // Attribute (int variable)
};

int main(){
     Hacker hacker_1; // Create an Object of Ha
cker
     // Access the attributes and set values
    hacker_1.name = "John";
     hacker_1.level = 7;
     // Print to the screen the values
     cout << hacker_1.name << "\n";
     cout << hacker_1.level;
     return 0;
 }
```

Practice 1: Instantiate object

- Quick practice (In Code::Block)
- Create a class Car with two attributes (use public: as access specifier)
  - Attribute color with string type
  - Attribute model with int type
- In main() function:
  - Create an object car_1 of class Car
  - Set the values for color is “Blue” (use dot operator)
  - Set the value for model is 2001
  - Print to the screen the value

Instantiate multiple C++ objects

```cpp
#include <iostream>
using namespace std;
class Hacker {       // The class Hacker
  public:            // Access specifier
    string name;     // Attribute (string variable)
    int level;       // Attribute (int variable)
};
int main(){
     // Create an object of Hacker
     Hacker hacker_1;
     hacker_1.name = "John";
     hacker_1.level = 7;
     // Create another object of Hacker
     Hacker hacker_2;
     hacker_2.name = "Christ";
     hacker_2.level = 8;
     // Print to the screen the values
     cout << "Hacker " << hacker_1.name << ", Level: " << hacker_1.level <<"\n";
     cout << "Hacker " << hacker_2.name << ", Level: " << hacker_2.level;
     return 0;
 }
```

Practice 2- multiple objects

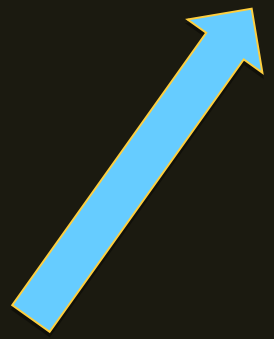
```
#include <iostream>
using namespace std;
class ……. {          // The class Car
  public:             // Access specifier
    string …….;       // Attribute color
    int ……;           // Attribute model
};
int main(){
     // Create an object of Car
     …. car_1;
     car_1… = “Blue";
     car_1…. = 2001;
     // Create another object of Car
     …. car_2;
     car_2… = “Orange";
     car_2…. = 2008;
     // Print to the screen the values
     cout << “Car 1: " << car_1.color << ", Model: " << car_1.model <<"\n";
     cout << “Car 2: " << … << ", Model: " << …..model;
     return 0;
 }
```

```
Car 1: Blue, model: 2001
Car 2: Orange, model: 2008
```

Suggested Answer

```cpp
#include <iostream>
using namespace std;
class Car {          // The class Car
  public:            // Access specifier
    string color;    // Attribute color
    int model;       // Attribute model
};
 int main(){
     // Create an object of Car
     Car car_1;
     car_1.color = "Blue";
     car_1.model = 2001;
     // Create another object of Car
     Car car_2;
     car_2.color = "Orange";
     car_2.model = 2008;
     // Print to the screen the values
     cout << "Car 1: " << car_1.color << ", model: " << car_1.model <<"\n";
     cout << "Car 2: " << car_2.color << ", model: " << car_2.model;
    return 0;
 }
```

```
Car 1: Blue, model: 2001
Car 2: Orange, model: 2008
```



How to create C++ Class?

To create a Hacker Class with **methods**

There are two ways to define functions (methods):

- Inside class (create function inside class)
- Outside class (declare inside and create outside)

Method - inside

```
#include <iostream>
using namespace std;

class Hacker {       // The class Hacker
  public:            // Access specifier
    void scan(){     // Create function/
method inside class
         cout << "I am scanning the target";
    }
};

 int main(){
    Hacker hacker_1; // Create an Object of Hacker
    hacker_1.scan(); // Call the method by "." operato
r
    return 0;
 }
```

Practice 3: Method - inside

- Do in Code::Blocks
- Create a class Hacker with one method (use public: as access specifier)
  - void attack()
  - print to screen “I am attacking for learning”
- In main() function:
  - Create an object hacker_yourname of class Hacker
  - Call the method attack()

Method - outside

```cpp
#include <iostream>
using namespace std;

class Hacker {       // The class Hacker
  public:            // Access specifier
    void scan();     // Declare the function/method
};

// Create function/method outside class
void Hacker::scan(){
        cout << "I am scanning the target";
    }

int main(){
    Hacker hacker_1; // Create an Object of Hacker
    hacker_1.scan(); // Call the method by "." operator
    return 0;
}
```

Remember double colons

Why we use Method - outside?

3. Encapsulation

Struct and Class

Syntactic difference between struct and class

```
struct sStudent {
    string name;
    int id;
};
```

&

```
class cStudent {
    string name;
    int id;
};
```

Struct and Class

Create object for each type

```
sStudent studentS;
```
&
```
cStudent studentC;
```

Set the data

```
studentS.id = 777;
```
&
```
studentC.id = 888;
```

Access Specifier

For modifying the feature of attributes and methods, we have 3 different type of access specifiers:

- Public can be directly accessed from outside the object.
- Private can only be directly accessed by internal methods.
- Protected can be directly accessed by objects of subclasses (inherited class), but not by arbitrary external objects.

We will look at protected later, but private and public allow us to capture the idea of encapsulation.

Access Specifier

What happen?

```
#include <iostream>
using namespace std;
class Hacker {        // The class Hacker
  public:             // Access specifier
    string name;      // Attribute (string variable)
  private:
    int level;        // Attribute (int variable)
};
int main(){
     Hacker hacker_1; // Create an Object of Hacker
     // Access the attributes and set values
    hacker_1.name = "John";
     hacker_1.level = 7;
     // Print the values to screen
     return 0;
}
```

Encapsulation - What is it?

The meaning of Encapsulation, is to make sure that “sensitive” data is hidden from users. To achieve this, we must declare class variables/attributes as private (cannot be accessed from outside the class).

Source:
https://www.w3schools.com/cpp/cpp_encapsulation.asp

Why Encapsulation

- It is considered good practice to declare your class attributes as private (as often as you can).
- Ensures better control of your data, because you (or others) can change one part of the code without affecting other parts
- Increased security of data

How to do Setter and Getter?

If we want others to read or modify the value of a private member, we can provide public `get` and `set` methods.

Encapsulation – Setter and Getter

```
#include <iostream>
using namespace std;
class Hacker {
    private:
        int level;
    public:
        void setLevel(int l){
            level = l;
        }
        int getLevel() const{
            return level;
        }
};
int main(){
    Hacker hacker_1;
    hacker_1.setLevel(7);
    cout << hacker_1.getLevel(
);
    return 0;
}
```

Explain the code:

- Level attribute is private, we cannot access directly
- Public setLevel() take variable l and assign to level attribute.
- Public getLevel() return level value.

So, in the main(), we create object hacker_1, using setLevel to pass 7 as a value. Level attribute will be 7. through function getLevel to return the level attribute is 7.

Practice 4

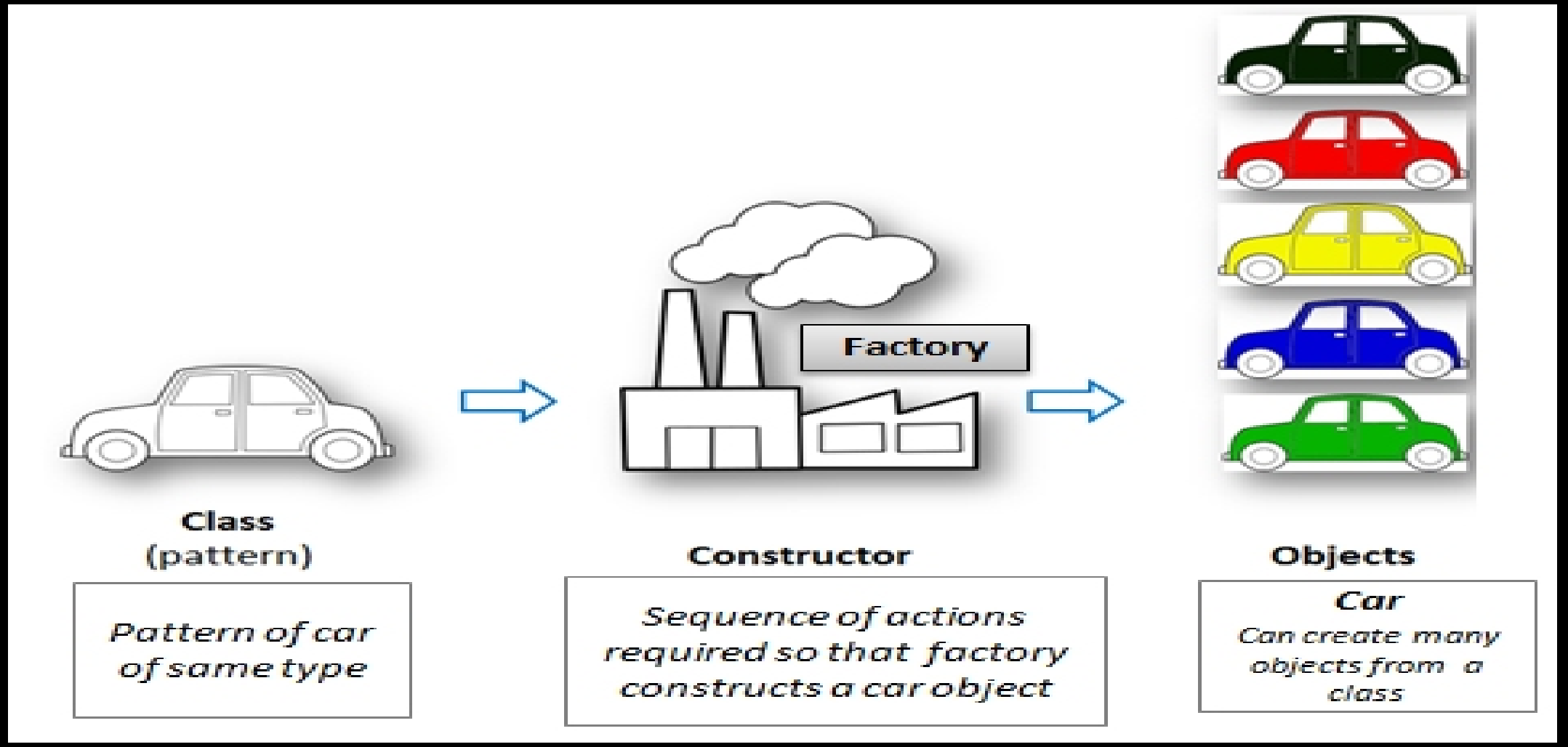
4. Object Management

Constructor and Destructor

Constructors - What is it?

A constructor in C++ is a special method/function/procedure that is automatically called when an object of a class is created.

Constructor - Idea



https://www.listendata.com/2019/08/python-object-oriented-programming.html

Constructors – How to use it

To create a constructor, use the same name as the class, followed by parentheses ():

In this case, it is default constructor without arguments.

```
#include <iostream>
using namespace std;
class Hacker {
    public:
        Hacker ()
{  // Constructor
     cout << "Hi CSCI251";
   }
};
int main(){
    Hacker hacker_1;
    return 0;
}
```

Constructors - Parameters

Constructors can also take parameters like regular functions. We use this technique to set initial values for attributes.

It is called non-default constructor.

Constructors - Parameters

We pass 2 parameter name

and level from creating object

```
#include <iostream>
using namespace std;
class Hacker {
    public:
        string name;
        int level;
        // Constructor passing parameter
        Hacker (string n, int l){
            name = n;
            level =l;
        }
};
int main(){
    // Generate hacker_1 object, call construct
or with parameters
    Hacker hacker_1("John", 7);
    cout<<"Hacker "<<hacker_1.name<<", level: "
<<hacker_1.level;
    return 0;
}
```

Practice 5: Constructors - Parameters

Do on Code::Blocks

- Class Car
- Constructor passing 2 parameters: color and model
- In main() function, create object car_1 and passing 2 variables
  - Color is blue
  - Model is 2001
  - Print to screen: “Car 1 has blue color and is created in 2001”

Constructors – from Outside

It is similar to the regular function. It can be defined outside the Class

```
#include <iostream>
using namespace std;
class Hacker {
    public:
        string name;
        int level;
        // Constructor passing parameter
        Hacker (string n, int l); // Declare constr
uctor
};
// Constructor is defined outside the class
Hacker::Hacker (string n, int l){
    name = n;
    level =l;
}
int main(){
    // Generate hacker_1 object, call constructor w
ith parameters
    Hacker hacker_1("John", 7);
    cout<<"Hacker "<<hacker_1.name<<", level: "<<ha
cker_1.level;
    return 0;
}
```

Multiple Constructors

We can have multiple constructors.

Default and non. So, it gives us the options to use.

In this case, we have hacker_1 with default constructor, hacker_2 with parameters

```
class Hacker {
public:
    string name;
    int level;
    Hacker(){
        cout<<"Hi CSCI251\n";
    }
    Hacker(string n, int l){
        name=n;
        level=l;
    }
};

int main(){
    Hacker hacker_1;
    Hacker hacker_2("John", 7);
    cout<<"Hacker "<<hacker_2.name<<", level: "<<hacker_2.l
evel;
    return 0;
}
```

Copy Constructors

We can copy a constructor.

In this case, hacker_3 has the same values with hacker_2

Practice 6

```
class Hacker {
public:
    string name;
    int level;
    Hacker(){
        cout<<"Hi CSCI251\n";
    }
    Hacker(string n, int l){
        name=n;
        level=l;
    }
};

int main(){
    Hacker hacker_2("John", 7);
Hacker hacker_3 = hacker_2;//
hacker_3.name="John"
    return 0;
}
```

Destructors - What is it?

Destructors are called for an object whenever the object goes out of scope.

Why we use Destructors?

- Think of using the constructor to set up objects and the destructor to remove them nicely.
- Destructors should allow memory to be released, avoiding memory leaks.
- They could be used to write some information about the object to a file or to standard out.

How to use it?

- They have the name as the class but with a leading tilde (~).
- One destructor per class. Default set up by the compiler if none is specified.
- No parameters, no return type.

order of destruction

```
#include<iostream>
using namespace std;

class House {
private:
    int* area;
public:
    House();
    ~House();
};
```

```
House::House()
{
    area = new int(300);
    cout << "House up!" << endl;
    cout << this << endl;
}

House::~House()
{
    cout << "House down!" << endl;
    cout << this << endl;
     // delete area;
}
```

```
int main()
{
    House aHouse[3];
    cout << “======" << en
dl;
}
```

Practice 7

order of destruction

```
House up!
0x61fde0
House up!
0x61fde8
House up!
0x61fdf0
===============
House down!
0x61fdf0
House down!
0x61fde8
House down!
0x61fde0
```

When constructors are created with 3 objects.

We can have each constructor creation of 8 byte data (pointer).

Then when calling destructors, each will delete the memory in the stack following LIFO (Last In First Out)

It means the last created is the first destroyed

Can constructors be private?

- Yes.
- They are usually public but we may have a reason for making a constructor private.
  - We may create an object with certain values only under conditions controlled from within an object.
  - A private constructor can also be used in the context of inheritance.
  - We will see later that a non-public constructor is used to implement a design pattern called a singleton..

Can destructors be private?

- Yes but be careful.
- The destructor is usually public but we can make them private.
- Why would we want to do this?
  - It can give us finality, something we will discuss in the context of inheritance.
  - It can be used to provide safe reference tracking.

```
#include <iostream>
using namespace std
;
class Example {
    private:
        ~Example()
{}
};
int main()
{
    Example ex1;
}
```

&

```
#include <iostream>
using namespace std;
class Example {
    private:
        ~Example(){}
};
int main()
{
    Example *ex2;
}
```

Be careful

```
#include <iostream>
using namespace std
;
class Example {
    private:
        ~Example()
{}
};
int main()
{
    Example ex1;
}
```

&

```
#include <iostream>
using namespace std;
class Example {
    private:
        ~Example(){}
};
int main()
{
    Example *ex2;
}
```

Object ex1 cannot be destructed
The private problem

Ex2 is not an object, it is a pointer
An object has not been constructed yet, so there is no need to destruct it.

Be careful

```
#include <iostream>
using namespace std;
class Example {
    private:
        ~Example(){}
};
int main()
{
    Example *ex2 = new Example;
    delete ex2;
}
```

Cannot be compiled

`delete` cannot get access to the private destructor.

This will compile if you comment out the delete, but there will be a memory leak because we aren't tidying up correctly.

5. Overloading

Overloading

- Constructors with different arguments…

```
eBill();
eBill(double, double, int, string, int)
;
eBill(const eBill &);
```

Practice 8

- This is an example of function overloading.
- Functions are overloaded when they have the same name but different argument lists.
- The parameters determine which one to use.

Overloading - Warning

The overloading is tied to the name being the same and the argument lists differing.

There will be problems if:

We provide default values for the function arguments so don't need them when we call

```
void calculation(int num = 1);
void calculation(char ch = 'A');
```

- We have two functions with the same name and argument list, but different return types.

```
int getMax( int x, int y );
float getMax( int x, int y );
```