1(a). A structure is the same as a class except for a few differences. A structure has a default access specifier of public while class is private. A Structure is not secure and cannot hide its implementation details from the user while a class is secure and can hide its programming and designing details.

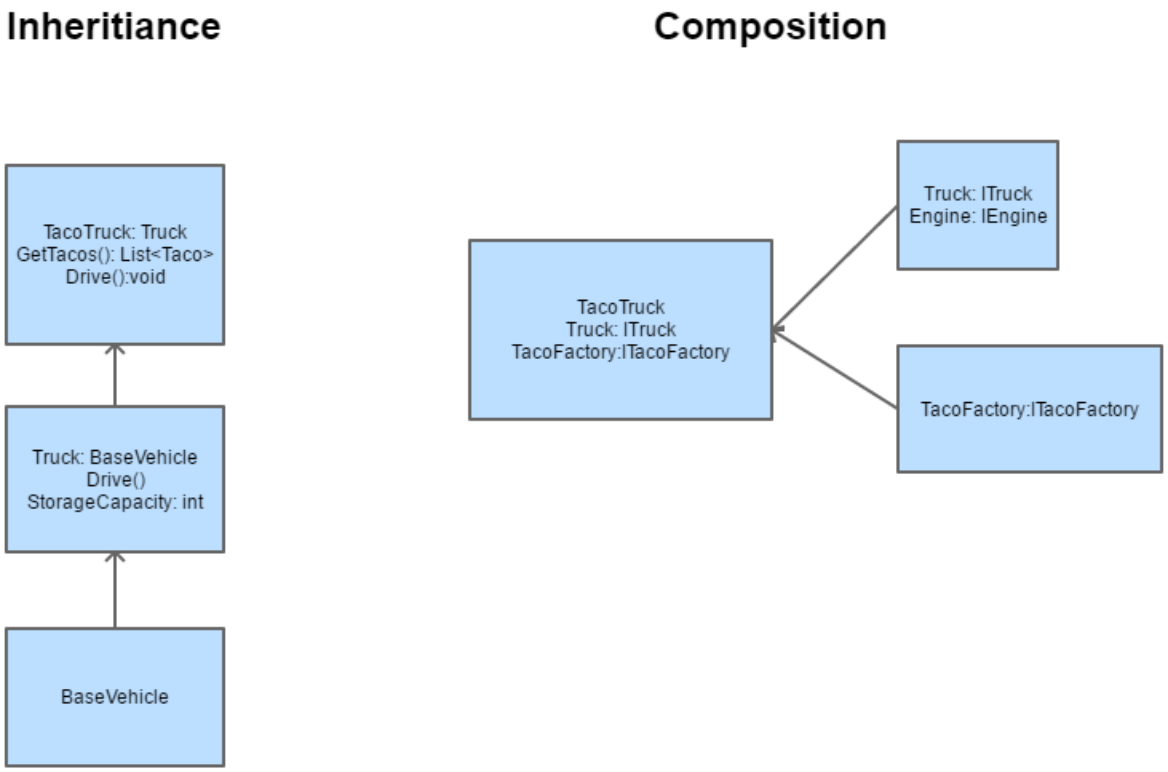
```
class Test {
    int x; // x is private
};
int main()
{
  Test t;
  t.x = 20; // compiler error because x is private
  getchar();
  return 0;
}
```

```
struct Test {
    int x; // x is public
};
int main()
{
  Test t;
  t.x = 20; // works fine because x is public
  getchar();
  return 0;
}
```

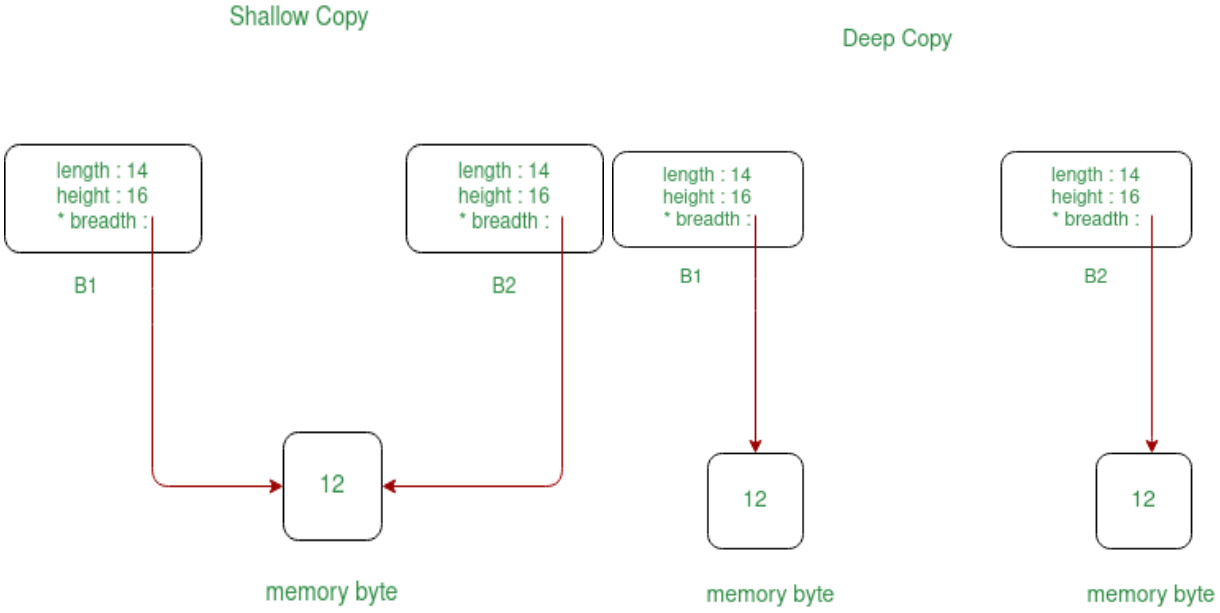
1(b). A class is a template for creating objects while an object is an instance of a class. You can only declare one class but you can declare multiple objects using the same class. Classes can't be manipulated because they are not in memory but an object can.

```
class azizz
{ public:
string name;
int age;
return 0;
}

// object example of class above
azizz mills;
mills.name;
mills.age;
```

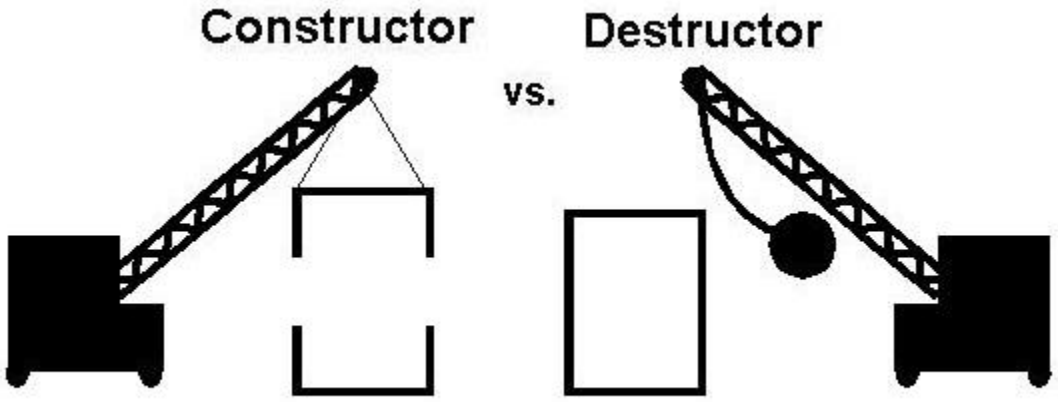
1(c). Composition is when a class can have an instance of another class as a field in that class. Inheritance is when an object can acquire the properties and behavior of the parent object by extending a class. Basically for composition, an object of one class is placed in another class which can be altered and manipulated. For inheritance, memory is only obtained and can only be manipulated in the current class.



1(d). Shallow copy stores the references of objects to the original memory address while Deep copy stores copies of the object's value. Shallow copy uses pointers and deep copy does not. Shallow copy is comparatively faster than deep copy.

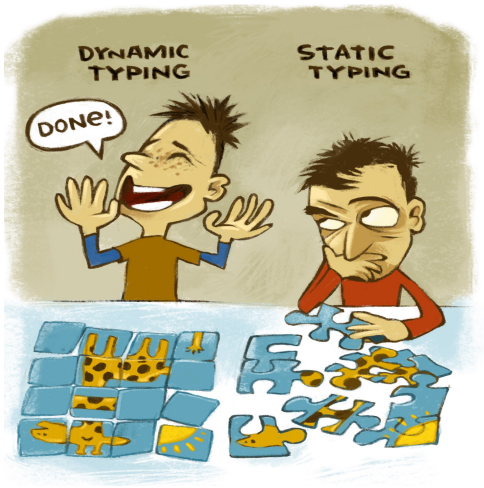


1(e). Constructor helps to initialize the data members of a class while destructor is used to destroy the instances of a class. A constructor can either accept arguments or not while a destructor cannot have any arguments. In a class, there can be multiple constructors while in a class, there is always a single destructor. None of them are necessary since a constructor is used to initialize data members but if a constructor is not present then a destructor is not necessary especially for dealing with deallocated memory.



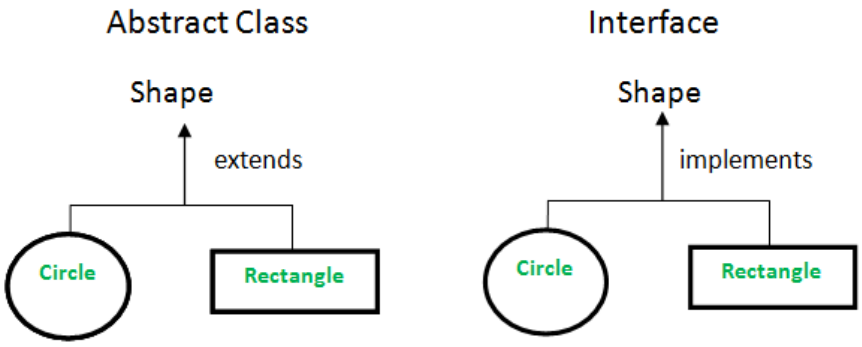
1(f). A language is statically typed if the type of a variable is known at compile time. For some languages (C++) this means that you as the programmer must specify what type each variable is while a dynamically typed language (Python) is associated with run-time values, and not named variables. This means that you as a programmer can write a little quicker because you do not have to specify data types every time.

Azizz Mills



1(g). Encapsulation is wrapping data together to make a single unit while abstraction is displaying the essential details to the user and hiding the rest.In abstraction, problems are solved at the design or interface level (meaning when writing the code problem is solved) while in encapsulation, problems are solved at the implementation level (the code must receive some input to fully resolve the problem). Examples of encapsulation is any program where a data member is made private and setters and getters are used to alter and access. A real world example for encapsulation would be a company with two sectors has one person who is the head of each sector. If one head of the sector wants information from another sector then he would have to go to the head of that sector and vice versa. A real world example for abstraction is that a person can turn a TV off and on but does know the full details or process as to how the TV turns off and on.

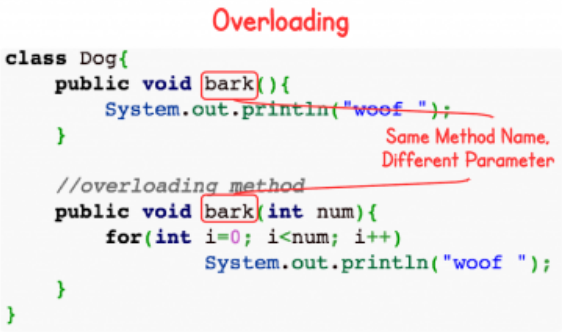
1(h). An interface tells the scope of a class without seeing an implementation of that class while an abstract class can have abstract and non-abstract methods. Abstract class can provide the implementation of the interface while interface can't provide the implementation of an abstract class.



1(i) A virtual function is one that is declared within a base class and is overwritten by a derived class while a pure virtual function is a virtual function that does not have an implementation. Classes of virtual functions are not abstract while classes of pure functions turn abstract. Virtual functions are basically selected based on the situation while pure virtual functions are stand alone.

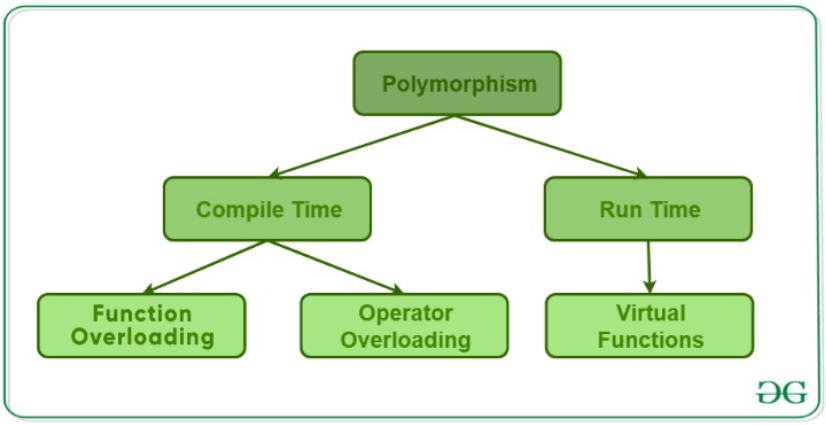
```
// virtual function
virtual<func_type><func_name>()
{
   // code
}
// pure virtual function
virtual<func_type><func_name>()
   = 0;
```

1(j). Function overloading is multiple definitions of the same function using different parameters. Function overriding is the redefining of a function with the same parameters.Overriding of functions can happen when one class is inherited from another class. Overloading can occur without inheritance. Overloaded functions must have different parameters. In overriding, function parameters are the same.

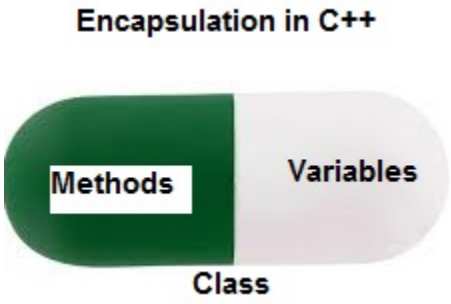


2(a). Polymorphism is when many classes that are related to each other by inheritance. Any example of this is a class called Animal has a method called

animal_sound. Then other classes derived from Animal for example dog, cat and pig will be able to use their own instance of animal_sound. There are two different types compile time which done by function overloading or operator overloading. Run time polymorphism which is done by function overriding.



2(b). Encapsulation is basically bundling data into one unit. An example of that is in the Admission office there are many departments. Each department there is someone in charge. If a head of department wants to get info from another department. They must go to the head to get it.



2(c). Abstraction is only displaying the data that is useful to the user and hiding the rest. For example, when you use a blender all you do is place the contents into it and switch it on. You do not know how exactly the motors in the blender work for or how long it remains working before it stops

3(a). A default constructor is a member function of a class which has the same name as the class, that can be called with no arguments, and is used to instantiate an object of a class using default values (or parameters). Default constructors are

called when an object is instantiated without any parameters in the parameter list at that point in the program.

3(b). An overloaded constructor is a member function of a class that is used to instantiate an object of that class using a different number of parameters than the default constructor. The number and type of parameters that are used to instantiate an object determine which constructor is called. For example, while a car class can be instantiated without passing any arguments, passing the string color = “red” and string type = ‘Toyota’ arguments into the call to create the person object will call the overloaded constructor that will use those two arguments to instantiate an object with those two data members in their definition.

3(c). A copy constructor is a constructor used to copy the contents from one object to the next in the same class. We do not need a copy constructor for every class. We only need one when there is dynamic memory involved and there needs to be a deep copy.

3(d). A deep copy is a way of creating an exact replica of the object having the same data. A deep copy is needed if the variables of an object have been dynamically allocated, in order to create a copy of the object.

3(e). Copy constructors (made by programmers) are required to allow for deep copying. This is because the programmer needs to create a constructor that dynamically allocates memory for the member variables in order to allow deep copying of an object.

3(f). The purpose of the copy constructor and the assignment operator are almost the same except that the copy constructor initializes new objects while the assignment operator replaces the contents of existing objects.

3(h). One major reason a class needs a destructor is dynamic memory management. When objects are instantiated, they take up memory until they are destroyed. Without destructors, memory from objects that are not being used and have not been destructed is still taken up, which consumes computer memory, slowing down performance.

Classes may also need destructors if the programmer intends to keep track of how many objects have existed and have been destroyed during runtime.

4. Abstract classes cannot be instantiated and is usually implemented as a class that has one or more pure virtual functions while interface class specifies the polymorphic interface (pure virtual function declarations into a base class)

```
class AbstractClass {

public:
  virtual void AbstractMemberFunction() = 0; // Pure virtual function makes
                                             // this class Abstract class.
  virtual void NonAbstractMemberFunction1(); // Virtual function.


  void NonAbstractMemberFunction2();
};
class shape   // An interface class
{
  public:
    virtual ~shape();
    virtual void move_x(int x) = 0;
    virtual void move_y(int y) = 0;
    virtual void draw() = 0;
//...
};
```

5. The term public means that its members can be accessed from outside the class. The term private means that it's members cannot be accessed from outside the class.
The term protected means that members cannot be accessed from outside the class but they can be accessed in inherited classes.

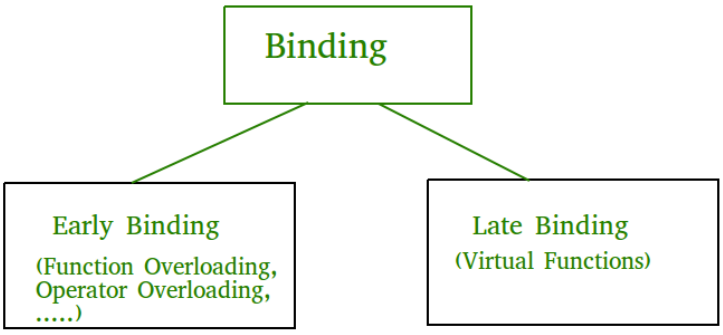
```
// example code for private and public
class MyClass {
  public:    // Public access specifier
```

```
    int x;
  private:   // Private access specifier
    int y;
};

int main() {
  MyClass myObj;
  myObj.x = 25;  // Allowed (public)
  myObj.y = 50;  // Not allowed (private)
  return 0;
}


// example code for protected
class Shape {
   public:
      void setWidth(int w) {
         width = w;
      }
      void setHeight(int h) {
         height = h;
      }

   protected:
      int width;   // variables to
      int height;  // be accessed
};

// Base class PaintCost
class PaintCost {
   public:
      int getCost(int area) {
         return area * 70;
      }
};
```

```
// Derived class
class Rectangle: public Shape, public PaintCost {
  public:
    int getArea() {
      return (width * height); // these variables are accessed
.                              //from the parent class
    }
};
```

6. The diamond problem is when two derived classes have a common parent class and those classes are the parents to a derived class. This causes an issue as the two derived classes may have data members of the same type but different values. Thus when the derived class tries to retrieve information from the two parent classes an error may occur. An example of this is at a university, a person(base class) will have a name and an age. He will also be a student(derived class) and belong to a faulty(derived class). The problem comes around if the student is also a TA(new derived class) then he will also need name and age. At this the TA would be receiving the information twice. This causes issues.

7. Early binding is when the compiler mapps out what function call goes to which function and turns that communication into machine code. Late binding happens during run time when the compiler adds code that identifies the kind of object it then mapps in machine code the object to the function call. By default early binding is happening (in C++ at least), where a function call is made and executed.By this time the compiler has already gone through the code and has mapped the calls to the function. Since virtual functions are altered at runtime, they are used for late binding. This is mainly because the corrected version of the virtual function must be selected first then the mapping from object to function can be done in machine code.



8. Base* bptr; // pointer for first class
bptr.show(); // displayed contents of show function from Base class
Derived d; // object from Derived class
bptr = &d; // making the Base pointer equal to that of the address of the Derived object.
bptr.show(); // displayed contents of show function from Derived class

9. // the following is for explanation only. No concatenation will be done to these variables.

```
class Automobile
{
 public:
 string “sportscar”;
};

class engine: public Automobile
{
public:
string “2.4-liter_twin_turbo_V-6_engine”;
string “Pennzoil”;
sportscar = Pennzoil + “2.4-liter_twin_turbo_V-6_engine”;
};
```

Inheritance would be the way to go in this case since the problem only asks to utilize a variable. If there was a demand for more complex functionality then composition would be the go to.

10. class Life
{
static int numBorn;

public:

Life () { numBorn++;}

void testmessage()
 {
“This line of string is printed out for each object”;
}
static void totalobj()
{ numLiving = numBorn
return numLiving ;
}

};

11(a). class Quad
{
public:
int length1;
int width1;
int length2;
int width2;
setlenght1();
setlenght2();
setwidth1();
setwidth2();
virtual int Area() {length1 *width1};

```
};
```

11(b). class Rectangle: public Quad

```
{
public:
int Area() {length1 *width1};
};
```

12. When there is no copy constructor and assignment operator for an object, the destructor runs only once to destroy an object. However there is a copy constructor and assignment operator the objects are altered via deep copy. Thus there are two copies of the same value in the objects. This causes the destructor to run twice.

13. Some of the cons of OOP is that:

It can be inefficient:
What is meant by this is that the processing power used to run the code places too much strain on the CPU as opposed to other code standards. For example dynamic language (Python).

It can be too scalar:
A great deal of unnecessary code can be accumulated if OOP is left unchecked. This waste spaces making it difficult to keep costs down.

It can make duplicates:
OOP projects are easier to design than to implement. This is because these classes are flexible in application. Projects may be up and running quickly but there is a large possibility that the project may seem cloned.

Bibliography

Structure vs class in C++
Last Updated : 19 Apr, 2021
From: https://www.geeksforgeeks.org/structure-vs-class-in-cpp/

C++ Classes and Objects
Last Updated : 08 Nov, 2019
From: https://www.geeksforgeeks.org/c-classes-and-objects/

Inheritance vs. Composition

Last updated:May 10, 2020
From: https://betterprogramming.pub/inheritance-vs-composition-2fa0cdd2f939

Shallow Copy and Deep Copy in C++
Last updated: 29 Dec, 2020
From: https://www.geeksforgeeks.org/shallow-copy-and-deep-copy-in-c/

Dynamic typing vs. static typing
Last updated: March 2015
From:
https://docs.oracle.com/cd/E57471_01/bigData.100/extensions_bdd/src/cext_transform_typing.html#:~:text=static%20typing,-This%20topic%20is&text=There%20are%20two%20main%20differences,type%20checking%20at%20compile%20time.

Difference between Abstraction and Encapsulation in Java with Examples
Last updated:08 Nov, 2019
From:
https://www.geeksforgeeks.org/difference-between-abstraction-and-encapsulation-in-java-with-examples/

Difference between Abstract Class and Interface in Java
Last updated: 31 Mar, 2021

From:
https://www.geeksforgeeks.org/difference-between-abstract-class-and-interface-in-java/

Difference between Virtual function and Pure virtual function in C++
Last updated: 09 Jun, 2020
From:
https://www.geeksforgeeks.org/difference-between-virtual-function-and-pure-virtual-function-in-c/#:~:text=A%20virtual%20function%20is%20a,derived%20class%20also%20becomes%20abstract.

Polymorphism in C++
Last Updated : 03 Jul, 2020

From: https://www.geeksforgeeks.org/polymorphism-in-c/

Encapsulation in C++
Last Updated : 08 May, 2019
From: https://www.geeksforgeeks.org/encapsulation-in-c/

Abstraction in C++
Last Updated : 30 May, 2018
From: https://www.geeksforgeeks.org/abstraction-in-c/

Multiple Inheritance in C++
Last Updated : 06 Sep, 2018
From: https://www.geeksforgeeks.org/multiple-inheritance-in-c/

Early binding and Late binding in C++
Last Updated : 05 Feb, 2018
From: https://www.geeksforgeeks.org/early-binding-late-binding-c/

6 Pros and Cons of Object Oriented Programming
Last updated: Nov 1, 2017

Azizz Mills

From:
https://greengarageblog.org/6-pros-and-cons-of-object-oriented-programming