What is c

- Most commonly-used language for embedded systems
- Developed in the 1970s – in conjunction with development of UNIX operating system
- In 1972, Invented by Dennies Ritchie at Bell Laboratories.
- Most of the programs of UNIX are written and run with the help of 'C'.
- Many of the important ideas of ‘C' stem are from BCPL by Martin Richards.
- Designed for systems programming
  a. Operating systems
  b. Utility programs
  c. Compilers
  d. Filters
- Evolved from B, which evolved from BCPL

What is C++

- Object-Oriented Programming (OOP).
- Enhanced version of the C language.
- Adds support for OOP without sacrificing any of C's power, elegance, or flexibility.
- C++ is a statically typed, compiled, general-purpose, case-sensitive, free-form programming language that supports procedural, object-oriented, and generic programming.
- C++ is regarded as a middle-level language, as it comprises a combination of both high-level and low-level language features.
-

C++ was developed by Bjarne Stroustrup starting in 1979 at Bell Labs in Murray Hill, New Jersey, as an enhancement to the C language and originally named C with Classes but later it was renamed C++ in 1983.

C++ is a superset of C, and that virtually any legal C program is a legal C++ program.

Facilitates a disciplined approach to program development

Diff between C and C++

- As we know both C and C++ are programming languages and used for application development.
- The main difference between both these languages is C is a procedural programming language and does not support classes and objects, while C++ is a combination of both procedural and object-oriented programming languages.
- In procedural programming, program is divided into small parts called ***functions***.
- In object oriented programming, program is divided into small parts called ***objects***.

- Procedural programming follows ***top down approach***.
- Object oriented programming follows ***bottom up approach***.
- ----------
- In top down approach, main() function is written first and all sub functions are called from main function. Then, sub functions are written based on the requirement.
- Whereas, in bottom up approach, code is developed for modules and then these modules are integrated with main() function.

- Procedural programming does not have any proper way for hiding data so it is ***less secure***.
- Object oriented programming provides data hiding so it is ***more secure***.

- In procedural programming, function is more important than data.
- In object oriented programming, data is more important than function.

- Procedural programming is based on ***unreal world***.
- Object oriented programming is based on ***real world***.

C++ is a statically typed, compiled, general-purpose, case-sensitive, free-form programming language that supports procedural, object-oriented, and generic programming.

C++ is regarded as a middle-level language, as it comprises a combination of both high-level and low-level language features.

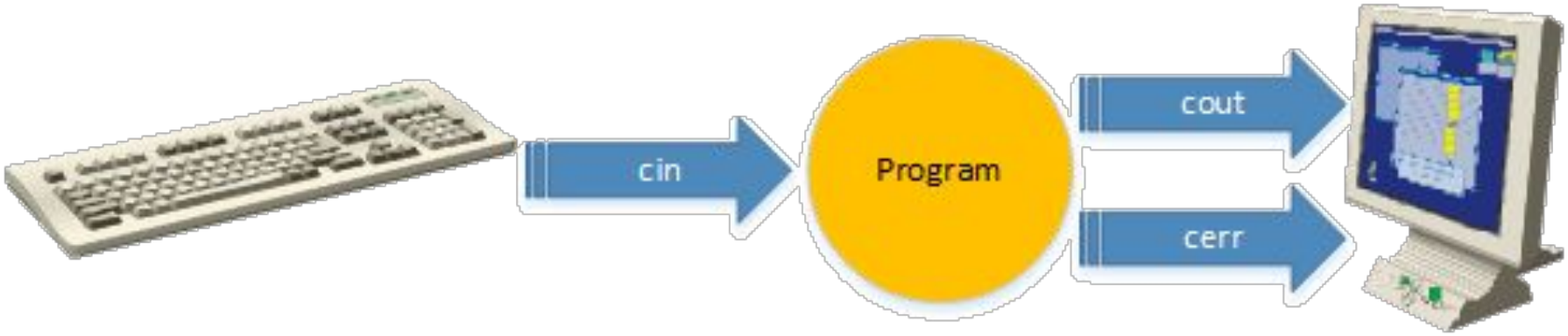
C++ was developed by Bjarne Stroustrup starting in 1979 at Bell Labs in Murray Hill, New Jersey, as an enhancement to the C language and originally named C with Classes but later it was renamed C++ in 1983.

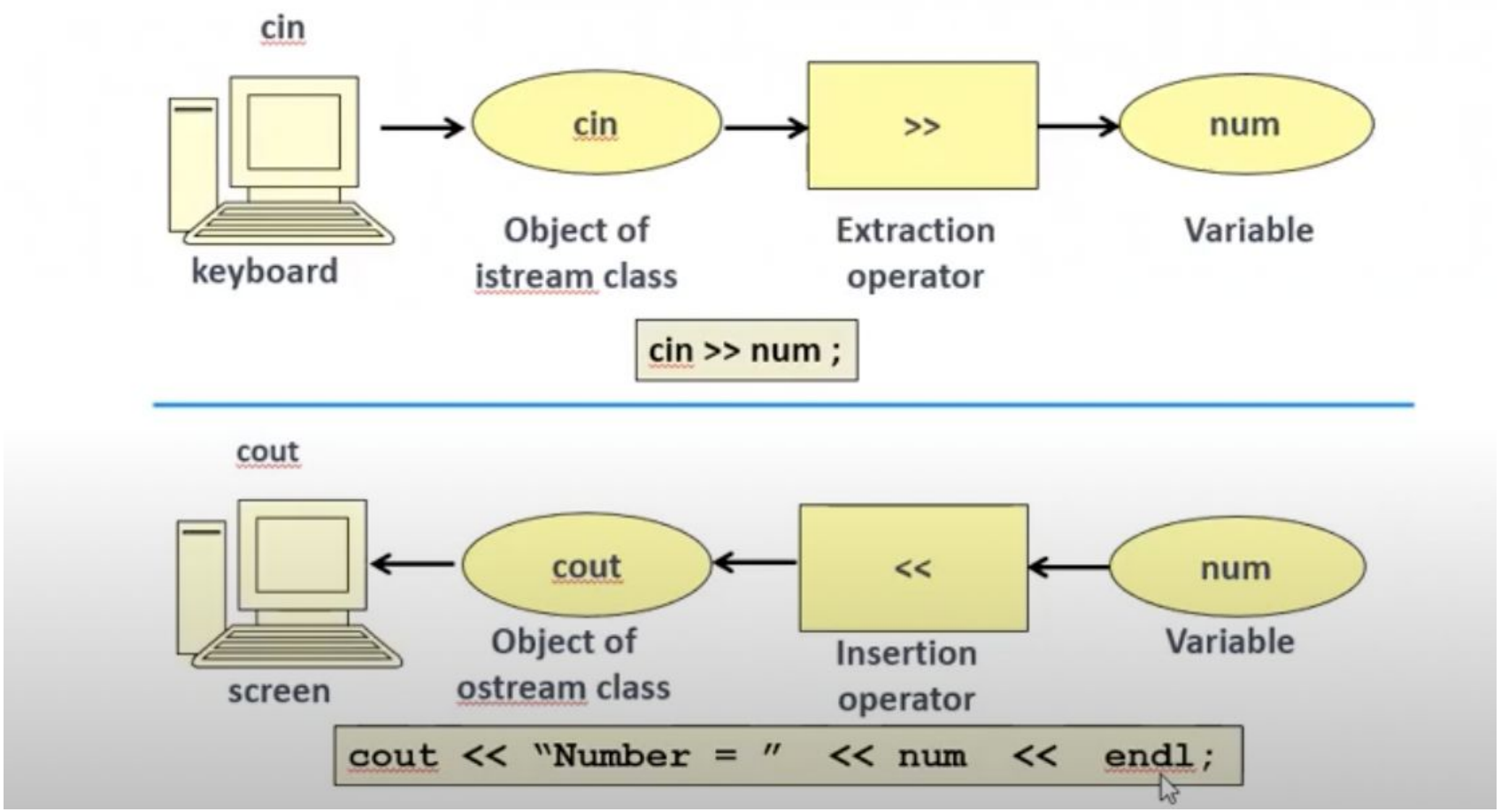
C++ is a superset of C, and that virtually any legal C program is a legal C++ program.

Object-oriented Approach

- Key concepts of object-oriented programming are:
  - Abstraction:
  - Encapsulation:
  - Inheritance:
  - Polymorphism:

cin & cout




cin
cin
>>
num
keyboard
Object of
istream class
Extraction
operator
Variable
cin >> num ;
cout
cout
<<
num
screen
Object of
ostream class
Insertion
operator
Variable
cout << "Number = " << num << endl;

- cin is an object of the input stream and is used to take input from input streams like files, Keyboard, etc.
- cout is an object of the output stream that is used to show output.
- Basically, cin is an input statement while cout is an output statement.

how to write Cpp Program

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

int main()
{
    int n;
    cout<<"Hello"<<endl;
    cout<<”Enter a number”;
    cin>>n;
    cout<<”n=”<<n;
    return 0;
}
```

- “using namespace std” means we use the namespace named std.
- “std” is an abbreviation for standard.
- So that means we use all the things within “std” namespace.
- If we don’t want to use this line of code, we can use the things in this namespace like this. std::cout, std::endl.
- Namespace is grouping of related functionality

- // Code written in the iostream.h file
-
- namespace std {
- ostream cout;
- istream cin;
- // and some more code
- }

- Data Types - 4 types of data types

1. Built in data types- int, char, float, double, long …

2. Derived Data types- pointer, array, function

3. Enum Data type- enum

4. User defined data types -class structure union

Data Types

- **Integer**: Keyword used for integer data types is **int**.

  Integers typically requires 4 bytes of memory space and ranges from -2147483648 to 2147483647.

- **Character**: Character data type is used for storing characters. Keyword used for character data type is **char**.

  Characters typically requires 1 byte of memory space and ranges from -128 to 127 or 0 to 255.

- **Boolean**: Boolean data type is used for storing boolean or logical values

  A boolean variable can store either *true* or *false*.

  Keyword used for boolean data type is **bool**.

- **Floating Point**: Floating Point data type is used for storing single precision floating point values or decimal values.
- Keyword used for floating point data type is **float**.
- Float variables typically requires 4 byte of memory space.

- **Double Floating Point**: Double Floating Point data type is used for storing double precision floating point values or decimal values.
- Keyword used for double floating point data type is **double**.
- Double variables typically requires 8 byte of memory space.

ENUM

- Enumeration is a user defined datatype in C/C++ language.
- It is used to assign names to the integral constants which makes a program easy to read and maintain.
- The keyword “enum” is used to declare an enumeration.
- The following is the syntax of enums.
- enum enum_name{const1, const2, ... };

Demo

- The size of variables might be different , depending on the compiler and the computer you are using.
- Following is the example, which will produce correct size of various data types on your computer.
- use **sizeof()** operator to get size of various data types.

```
cout << "Size of char : " << sizeof(char) << endl;
cout << "Size of int : " << sizeof(int) << endl;
cout << "Size of short int : " << sizeof(short int) << endl;
cout << "Size of long int : " << sizeof(long int) << endl;
cout << "Size of float : " << sizeof(float) << endl;
cout << "Size of double : " << sizeof(double) << endl;
```

- If condition
- For loop
- While loop
- Do while loop
- Switch case
- array

Reference Variable

- A reference variable is an alias, that is, another name for an already existing variable.
- Once a reference is initialized with a variable, either the variable name or the reference name may be used to refer to the variable.
- The main use of references is acting as function formal parameters to support pass-by-reference.
- In an reference variable is passed into a function, the function works on the original copy (instead of a clone copy in pass-by-value).
- Changes inside the function are reflected outside the function.

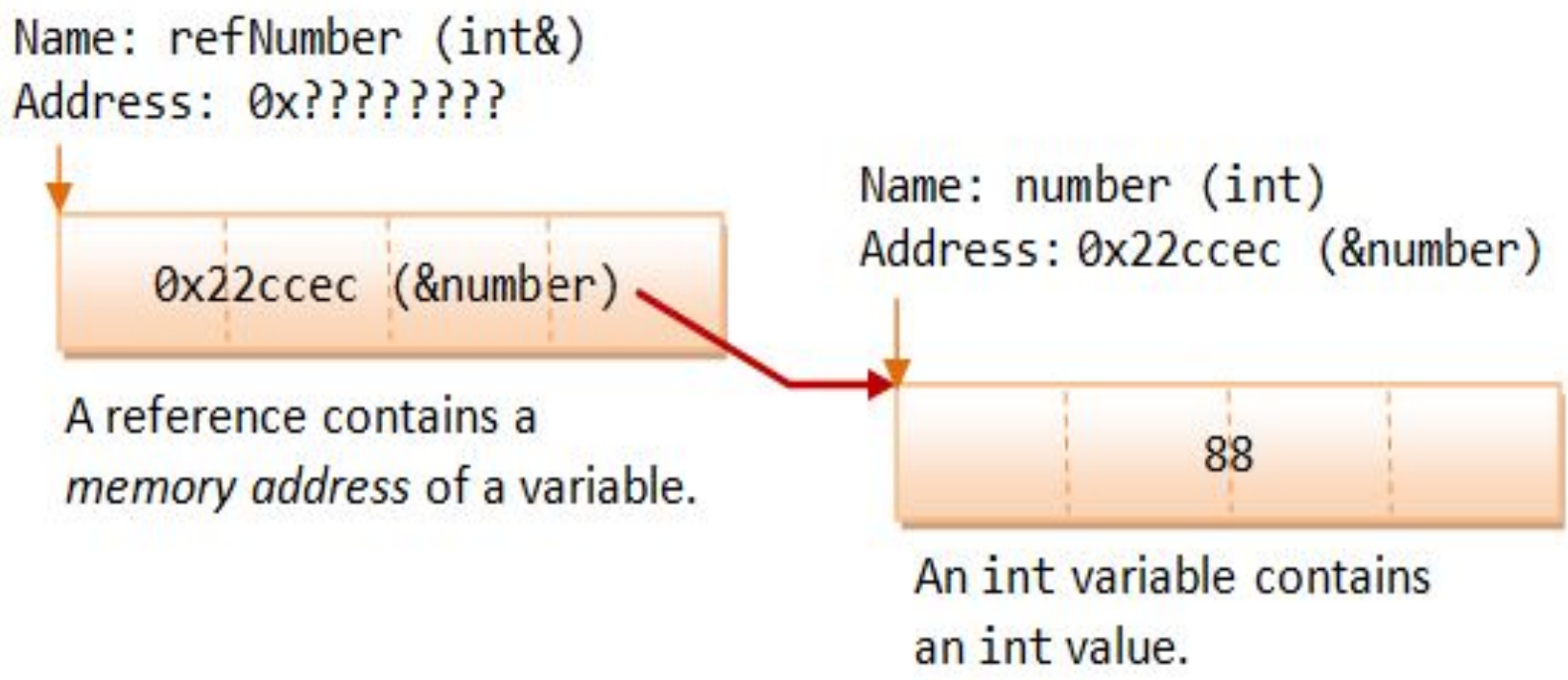
- A reference is similar to a pointer. In many cases, a reference can be used as an alternative to pointer, in particular, for the function parameter.

Demo

- int number = 88; // Declare an int variable called number
- int &refNumber = number; // Declare a reference (alias) to the variable number
- // Both refNumber and number refer to the same value
- cout << number << endl; // Print value of variable number (88)
- cout << refNumber << endl; // Print value of reference (88)
-
- refNumber = 99; // Re-assign a new value to refNumber
- cout << refNumber << endl;
- cout << number << endl; // Value of number also changes (99)
-
- number = 55; // Re-assign a new value to number
- cout << number << endl;
- cout << refNumber << endl; // Value of refNumber also changes (55)

How References Work?

- A reference works as a pointer. A reference is declared as an alias of a variable. It stores the address of the variable, as illustrated:



Using a Reference

- While using references you should know …
  - References have to be initialized.
  - No memory is allocated to references.

Reference: Pass by Reference

- From the function call one cannot make out whether the parameters have been “passed by value” or “passed by reference”.

```
void swapRef(int
  &ref1,
                int
  &ref2)
{
  int temp;
  temp = ref1;
  ref1 = ref2;
  ref2 = temp;
}
```

```
int main()
{
  int n1=10, n2=20;
  //pass by reference
  swapRef(n1, n2);
  cout<<“n1=”<<n1;
  cout<<“n2=”<<n2;
  return 0;
}
```

function

- A function is a group of statements that together perform a task.
- Every C++ program has at least one function, which is main(), and all the most trivial programs can define additional functions.
- A function declaration tells the compiler about a function's name, return type, and parameters.
- A function definition provides the actual body of the function.
- The general form of a C++ function definition is as follows –

```
return_type function_name( parameter list ) {
  body of the function
}
```

Types of functions

call by value

call by reference

call by address

default parameter in function

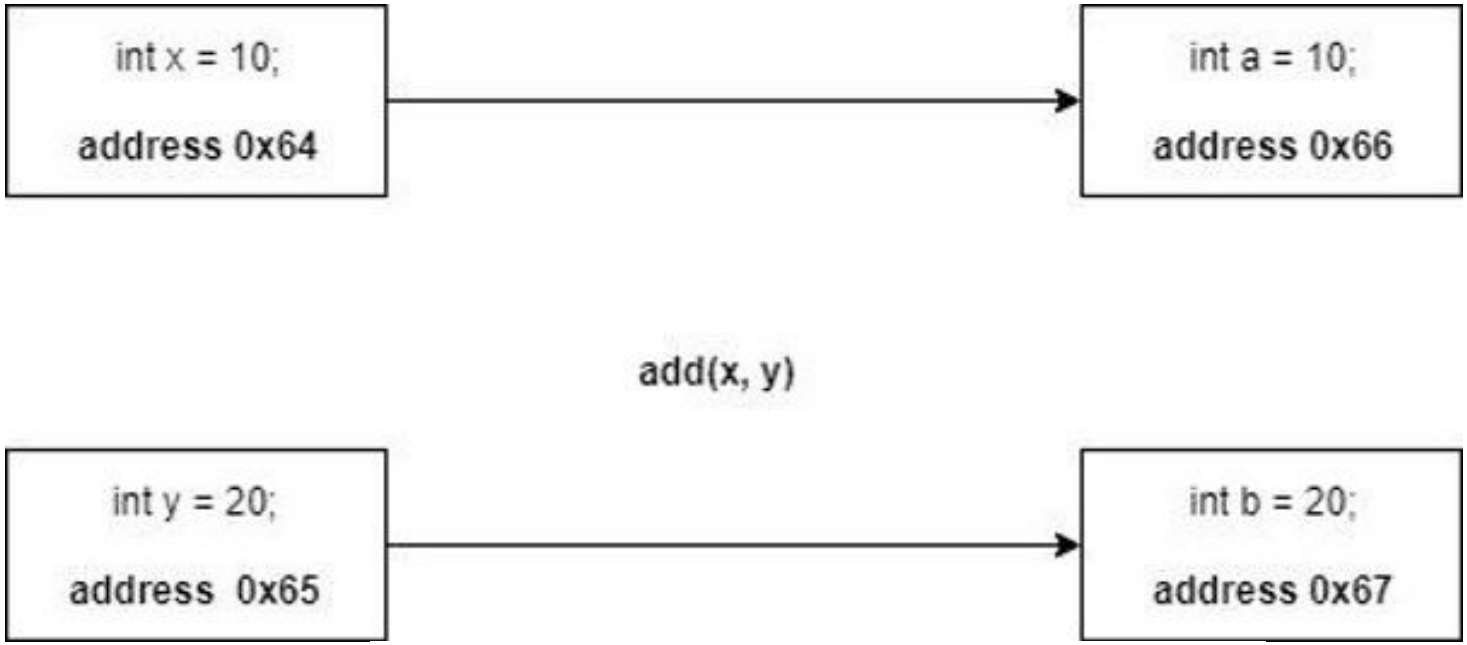
inline function

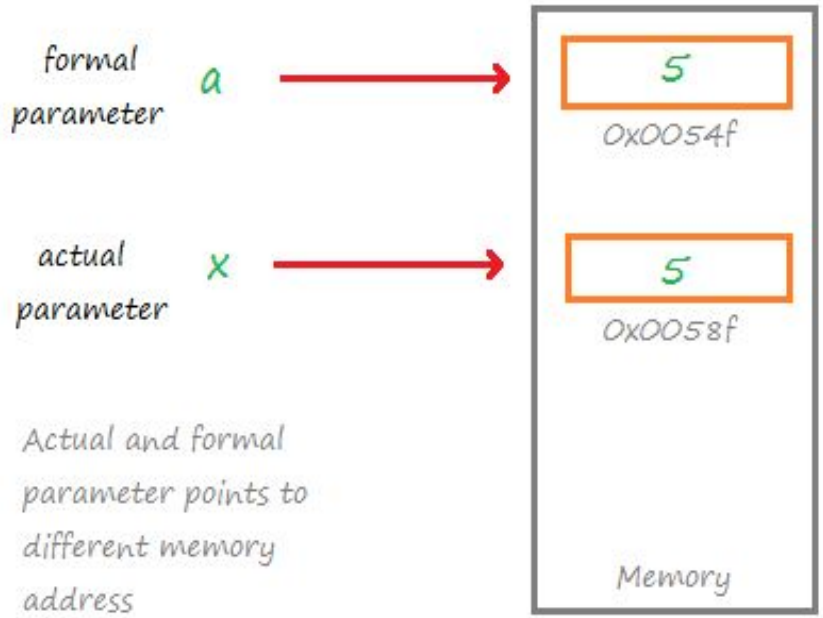
friend function

Call by value

- widely used method.
- you don't want your original values of the variables to be changed.
- only the values of the variables are passed.
- achieved by creating dummy variables in memory.

```
void main(){
    int x=8, y=6;
    int s=add(x,y);        //call by value
    cout<<s;
}
int add(int a, int b){
    return a+b
}
```

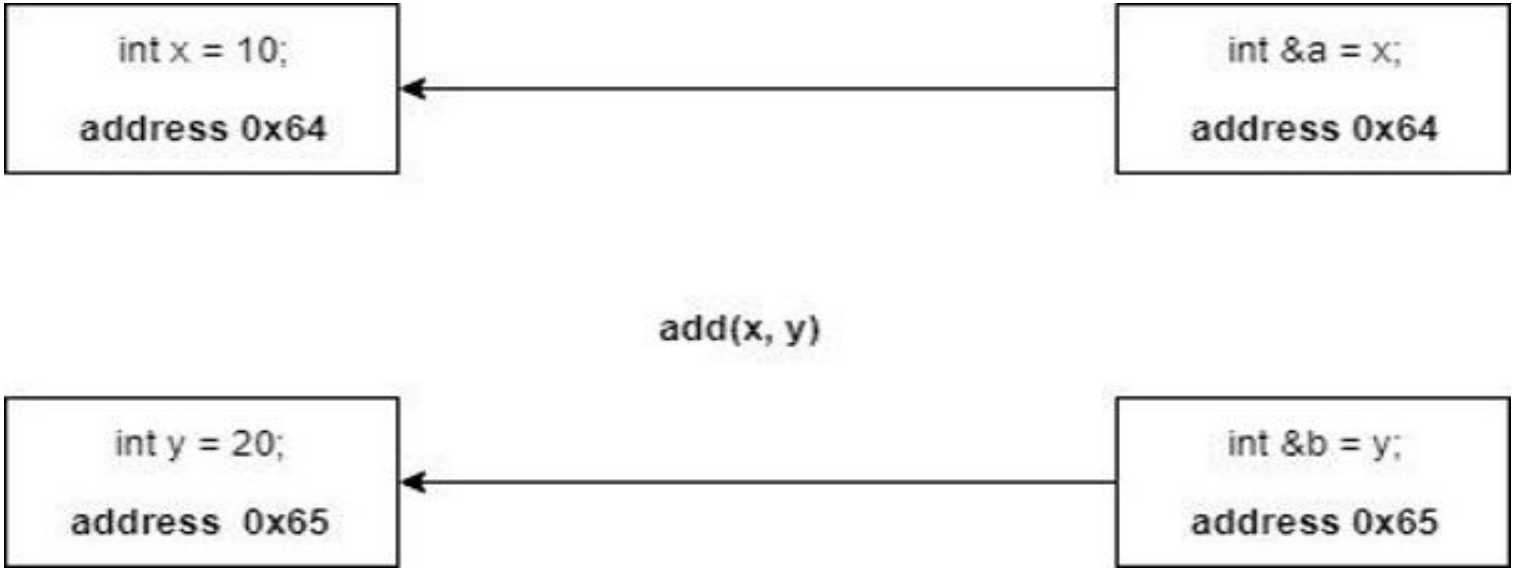


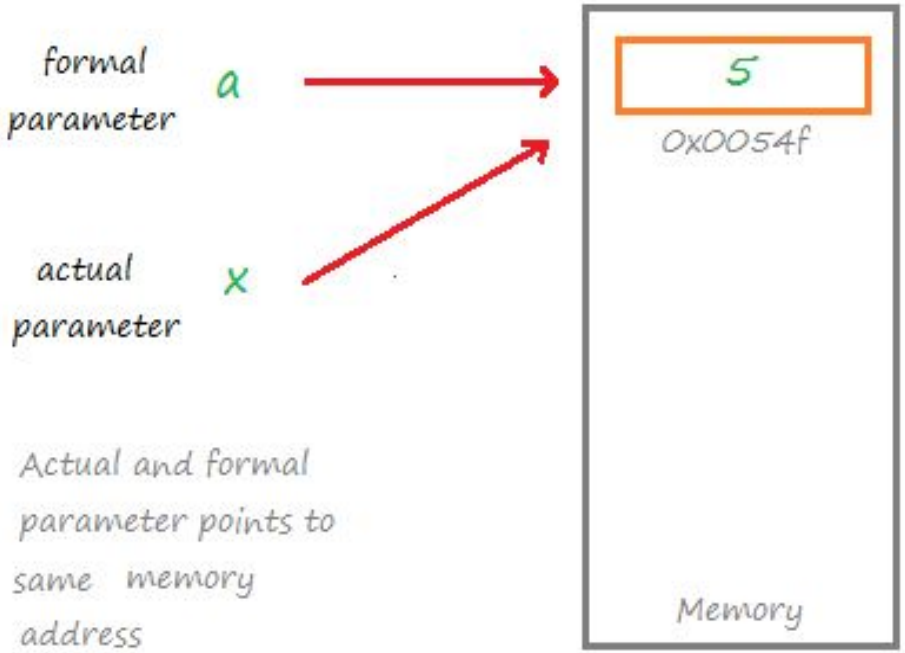


Actual and formal parameter points to different memory address

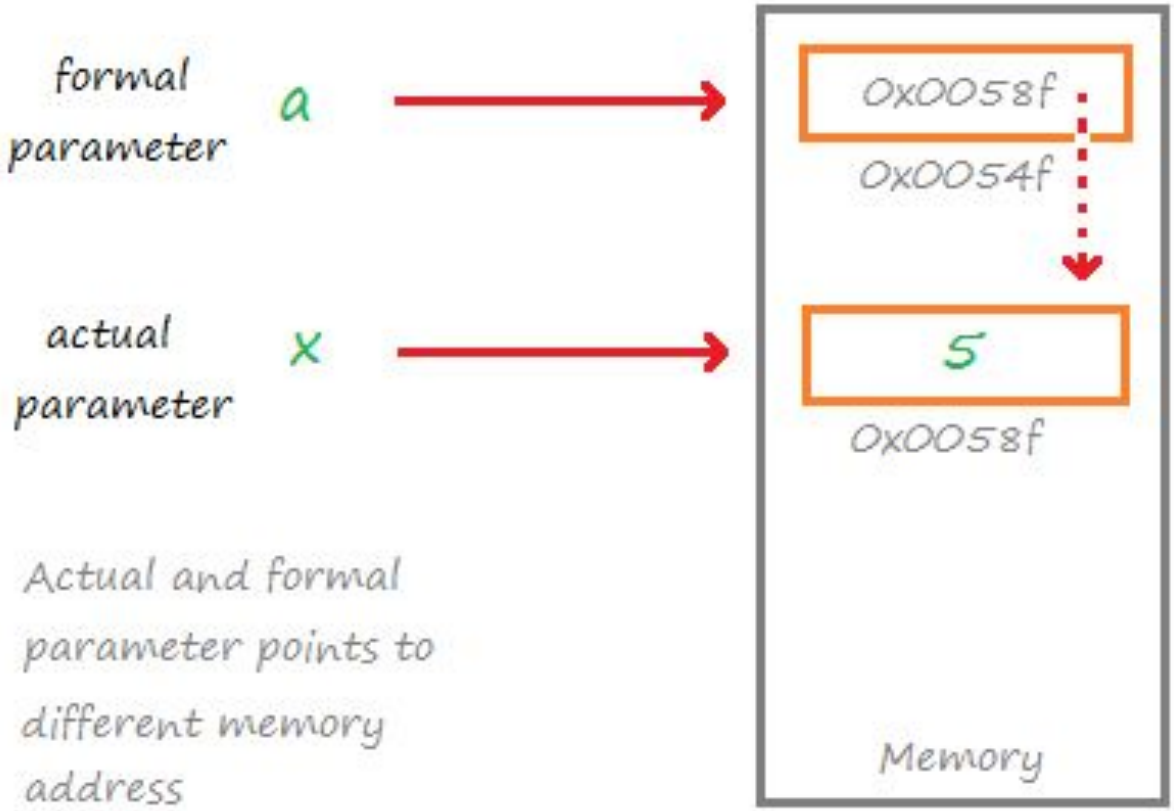
Call by Reference

- Dummy variables are not created,
- A reference of an already existing variable is passed to the method.
- This reference points to the same memory location
- Hence separate copies are not made in the memory.
- The important point to note - changes made in the reference variables are reflected in the actual variable.


int x = 10;
address 0x64
int &a = x;
address 0x64
add(x, y)
int y = 20;
address 0x65
int &b = y;
address 0x65


formal parameter
a
actual parameter
x
5
0x0054f
Memory
Actual and formal parameter points to same memory address

- Call by address / Pass by address
- The function arguments are passed as address.
- The caller function passes the address of the parameters.
- Pointer variables are used in the function definition.
- With the help of the Call by address method, the function can access the actual parameters and modify them.
- pointer variable holds the address of the actual parameter, hence the changes done by the formal parameter is also reflected in the actual parameter.



Actual and formal parameter points to different memory address

Inline Function

- C++ provides an inline functions to reduce the function call overhead.
- Inline function is a function that is expanded in line when it is called.
- When the inline function is called whole code of the inline function gets inserted or substituted at the point of inline function call.
- This substitution is performed by the C++ compiler at compile time.
- Inline function may increase efficiency if it is small.
- inline functions to reduce the function call overhead.

```
inline int cube(int s)
{
   return s*s*s;
}
int main()
{   cout << "The cube of 3 is: " << cube(3) << "\n";
   return 0;
} //Output: The cube of 3 is: 27
```

Remember,

inlining is only a request to the compiler, not a command.

Compiler can ignore the request for inlining.

Compiler may not perform inlining in such circumstances like:

1) If a function contains a loop. (for, while, do-while)

2) If a function contains static variables.

3) If a function is recursive.

4) If a function return type is other than void, and the return statement doesn’t exist in function body.

5) If a function contains switch or goto statement.

Inline functions provide following advantages:

1) Function call overhead doesn’t occur.

2) saves the overhead of push/pop variables on the stack when function is called.

3) It also saves overhead of a return call from a function.

Friend Function

- Non-member functions of a class will not have access to the private data of another class.
- There could be situations where we want two classes to share some functions and the data members.
- In that case, we can make the function a friend of these classes, and that will enable the function to access the private and protected data members of the classes.
- What is a friend function?
- A friend function is a function that is specified outside a class but has the ability to access the class members' protected and private data.

- A friend can be a member's function, function template, or function, or a class or class template, in which case the entire class and all of its members are friends.

Features

- A friend function does not fall within the scope of the class for which it was declared as a friend. Hence, functionality is not limited to one class.
- The friend function can be a member of another class or a function that is outside the scope of the class.
- A friend function can be declared in the private or public part of a class without changing its meaning.
- Friend functions are not called using objects of the class because they are not within the class's scope.

- Without the help of any object, the friend function can be invoked like a normal member function.
- Friend functions can use objects of the class as arguments.
- A friend function cannot explicitly access member names directly. Every member name has to use the object's name and dot operator.
- Syntax

```
class className{

  // Other Declarations

  friend returnType functionName(arg list);

};
```

Case 1: Outsider function as Friend of class

Case 2 : member function of one class will be friend of another class

case 3: one whole class will be friend of other class

class member function friend of other class

```
class className1{
  // Other Declarations
  int functionName1(); // member function
of className1
};


class className2
{
  // Other Declarations
  friend int className1::functionName1();
   //The functionName1() is a friend of
className2
};
```

Friend Class

- A friend class can have access to the data members and functions of another class in which it is declared as a friend.
- They are used in situations where we want a certain class to have access to another class's private and protected members.
- Classes declared as friends to any another class will have all the member functions become friend functions to the friend class.
- Friend functions are used to work as a link between the classes.

- Syntax of friend class:

```
class S; //forward declaration

class P{
  // Other Declarations
  friend class S;
};

class S{
  // Declarations
};
```

C++ OOP Concepts

- The major purpose of C++ programming is to introduce the concept of object orientation to the C programming language.
- Object Oriented Programming is a paradigm that provides many concepts such as inheritance, data binding, polymorphism etc.
- The programming paradigm where everything is represented as an object is known as truly object-oriented programming language.
- Smalltalk is considered as the first truly object-oriented programming language.

OOPs (Object Oriented Programming System)

- **Object** means a real word entity such as pen, chair, table,fan etc.
- **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects.
- It simplifies the software development and maintenance by providing some concepts:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

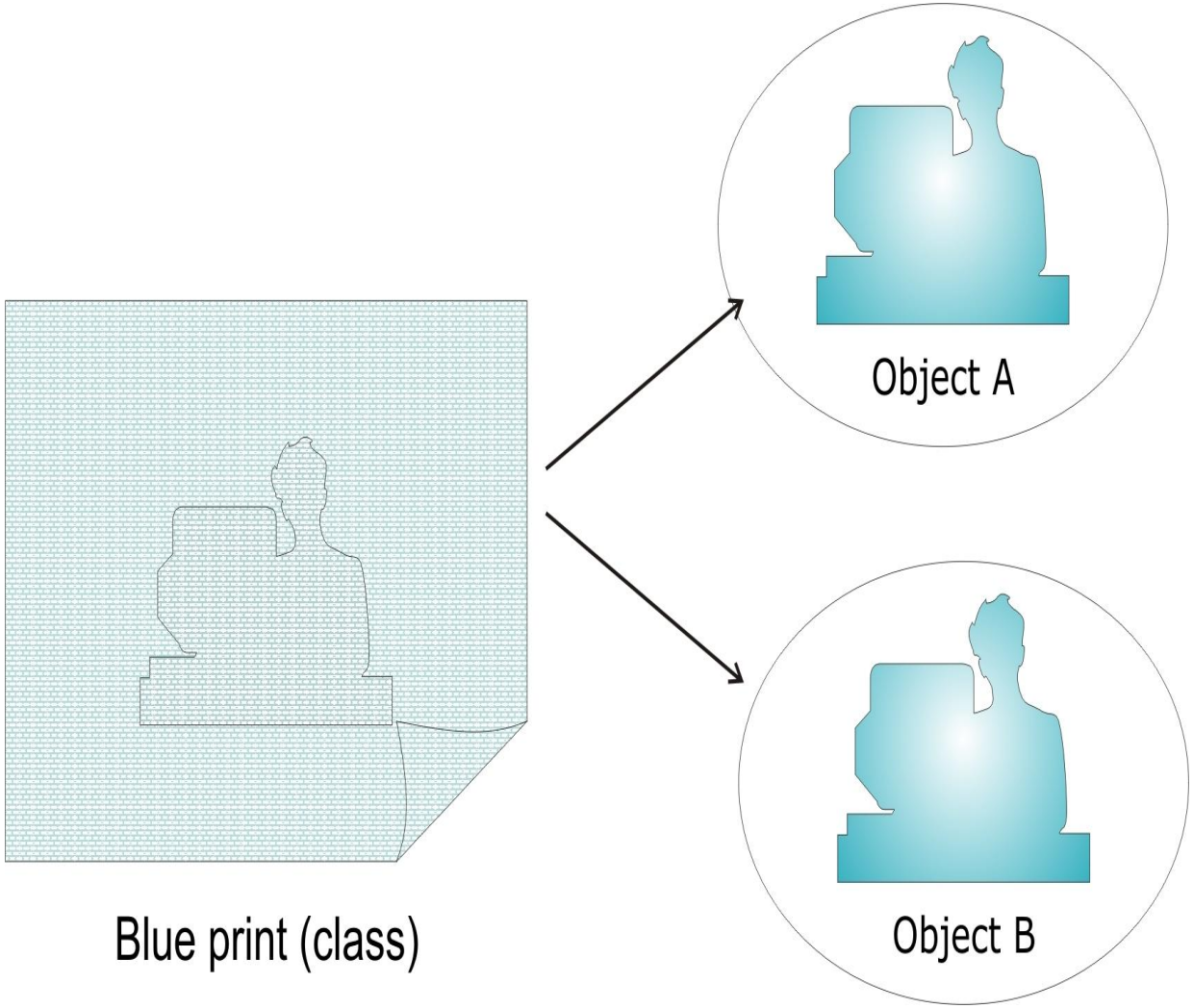
Class

- `struct` keyword can be replaced by `class` keyword.
  - C++ supports `struct` keyword for compatibility.
- Generally `struct` is used in ‘C’ context while `class` is used in C++ context.
- Class and object is C++ terminology.

Class

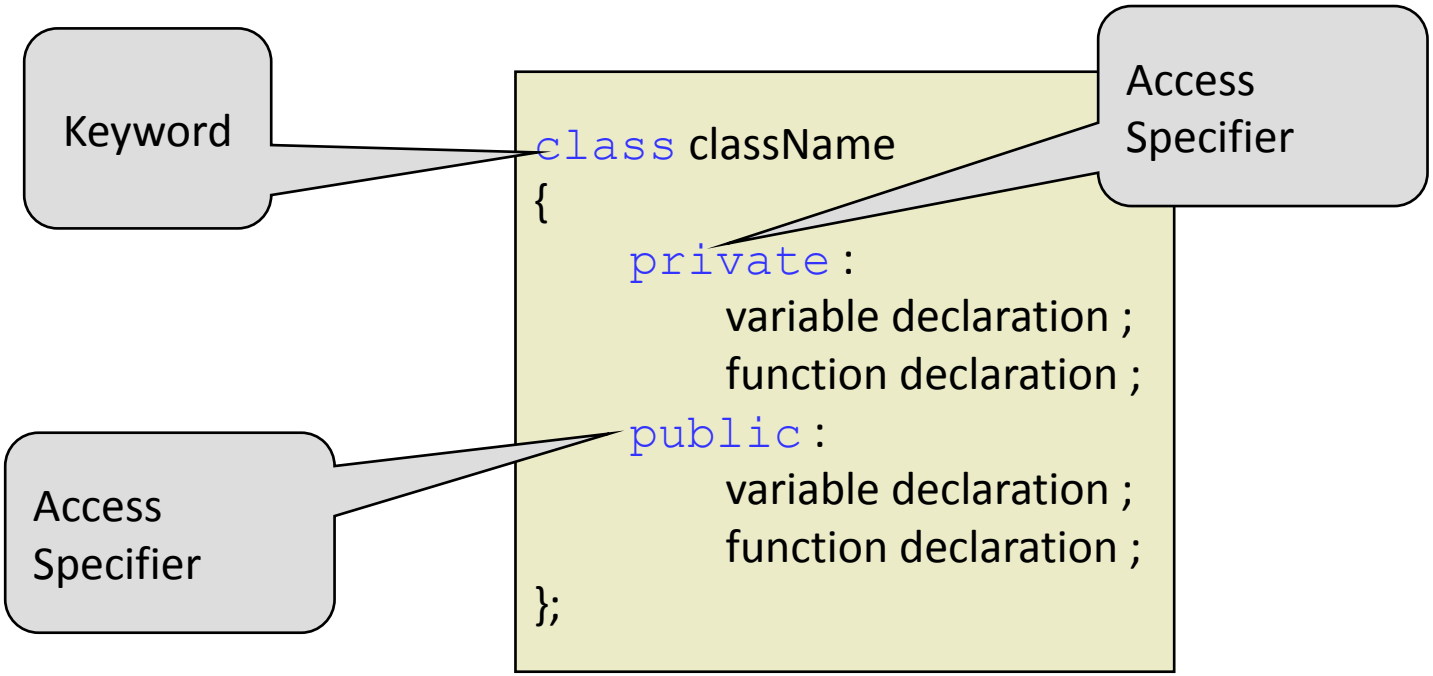
- A template for creating similar objects.
- Maps real world entities into classes through data members and member functions.
- A user defined type.
- An object is an instance of a class.
- By writing a class and creating objects of that class, one can map two concepts of object model, abstraction and encapsulation, into software domain.

Objects and Classes



Class

- Template for the creation of similar objects.
- A class in C++ is an encapsulation of data members and member functions that manipulate the data.



If semicolon is missing, compiler throws an error

- syntax error : missing ';' before 'PCH creation point' Error executing `cl.exe`

Class Components

- A class declaration consists of following components
  - Access specifiers: restrict access of class members
    - `private`
    - `protected`
    - `public`
  - Data members
  - Member functions
    - Constructors
    - Destructors
    - Normal Function

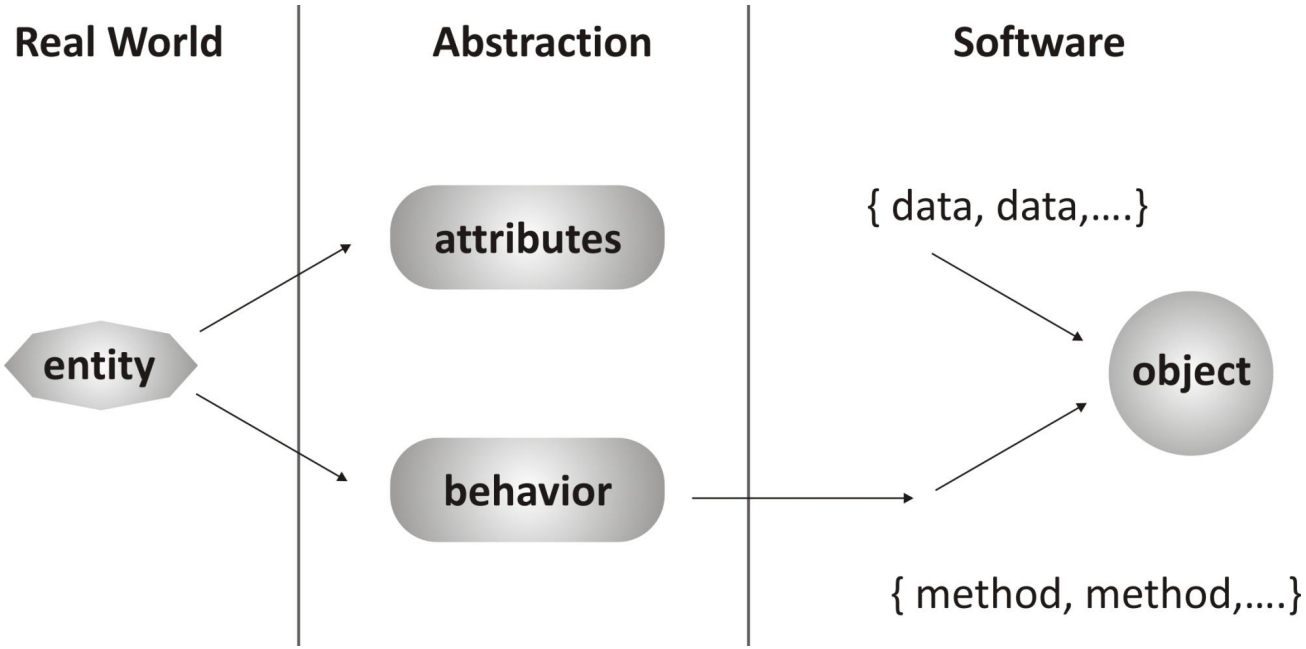
Abstraction

- Abstraction is the process of identifying the key aspects of an entity and ignoring the rest.
- Only those aspects are selected that are important to the current problem scenario.
- Example : Abstraction of a person object
  - Enumerate attributes of a “person object” that need to be created for developing a database
    - useful for social survey
    - useful for health care industry
    - useful for payroll system

Abstraction of a `Person` Object

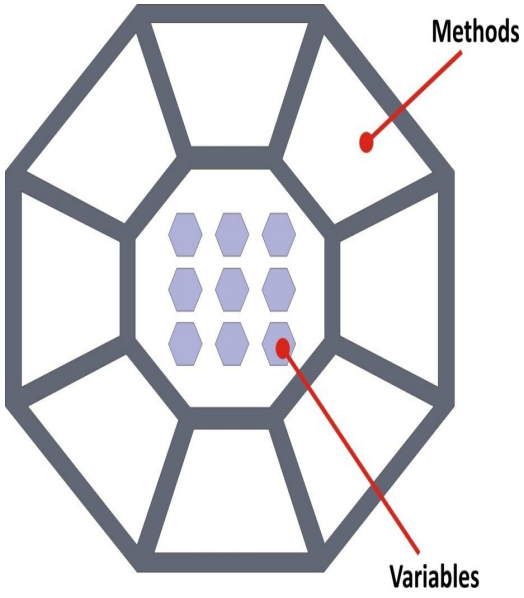
Social Survey	Health Care	Payroll System
name	name	name
Age	age	age
marital status	---	---
religion	---	---
income group	---	---
address	address	address
occupation	occupation	occupation
---	blood group	---
---	weight	---
---	previous record	---
---	---	basic salary
---	---	department
---	---	qualification

Abstraction



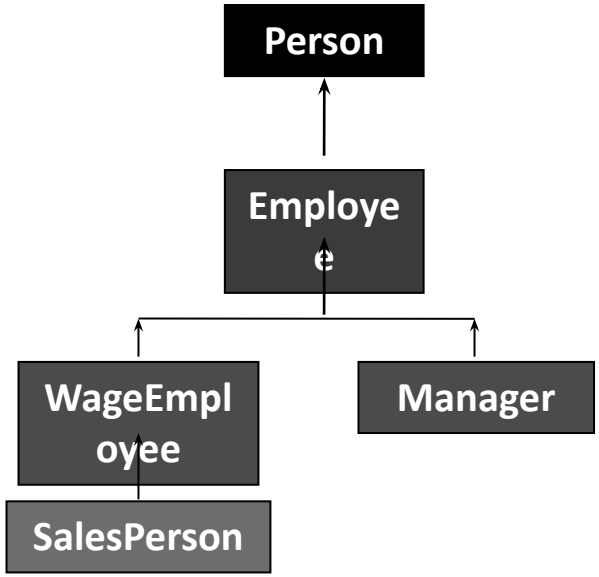
Encapsulation

- Encapsulation is a mechanism used to hide the data, internal structure, and implementation details of an object.
- All interaction with the object is through a public interface of operations.
- The user knows only about the interface; any changes to the implementation does not affect the user.



Inheritance

- Classification helps in handling complexity.
- Inheritance is the process by which one object can acquire the properties of another object.
  - Broad category is formed and then sub-categories are formed.
- “is – a” a kind of hierarchy.

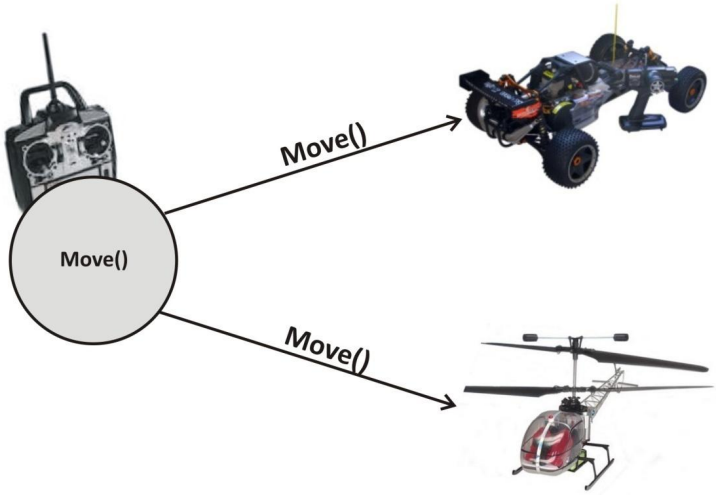


Inheritance

- Generalization
  - Factoring out common elements within a set of categories into a more general category called super-class.
  - Requires good skills of abstraction.
- Specialization
  - Allows to capture specific features of a set of objects.
  - Requires a depth of knowledge of the domain.

Polymorphism

- The ability of different types of objects to respond to the same message in different ways is called polymorphism.
- Polymorphism helps to :
  - Design extensible software; as new objects can be added to the design without rewriting existing procedures.



Containment

- One object may contain another as a part of its attribute
  - Document contains sentences which contain words.
  - Computer system has a hard disk, processor, RAM, mouse, monitor, etc.
- Containment need not be physical
  - E.g. Computer system has a warranty.

Containment Vs Inheritance

- Containment is used:
  - When the features of an existing class are wanted inside a new class, but not its interface.
    - Computer system has a hard disk.
    - Car has an engine, chassis, steering wheel.
- Inheritance is used:
  - When it is necessary that the new type has to be the same type as the base class.
    - Computer system is an electronic device.
    - Car is a vehicle.