Lecture One

# Basic Features of
# Object Oriented Programming (OOP)

© **Dr. Mohammad Mahfuzul Islam, PEng**
Professor, Dept. of CSE, BUET

# C/C++ Basic I/O

General form of C++ Console I/O

- **Input Command**:    **cin >> variable;**
- **Output Command**:    **cout << expression;**

| C Basic I/O | C++ Basic I/O |
|---|---|
| **a) Input Statements** | |
| scanf("%s", strName); | cin >> strName; |
| scanf("%d", &iCount); | cin >> iCount; |
| scanf("%f", &fValue); | cin >> fValue; |
| scanf("%d %d %d", &day, &month, &year); | cin >> day >> month >> year; |
| **b) Output Statements** | |
| printf("%s%c%s%c", "Hello", ' ', "World", '!'); | cout << "Hello" << ' ' << "World" << '!'; |
| printf("Value of iCount is: %d", iCount); | cout << "Value of iCount is: " << iCount; |
| printf("Enter day, month, year"); | cout << "Enter day, month, year: "; |

# I/O in C++ Programming

**C Code**

```c
#include <stdio.h>
int main() {
    int a, b, sum;
    char str[16];

    printf("Enter number 1: ");
    scanf("%d", &a);
    printf("Enter number 2: ");
    scanf("%d", &b);
    printf("Enter a string: ");
    scanf("%s", str);
    sum = a + b;
    printf("The sum is %d : %s", sum, str);

    return 0;
}
```

**Output:**

```
Enter number 1: 12
Enter number 2: 23
Enter a string: Love C Programming.
The sum is 35 : Love
```

**Why? How to solve this?**

**C++ Code**

```cpp
#include <iostream>
using namespace std;
int main() {
    int a, b, sum;
    char str[16];

    cout << "Enter number 1: ";
    cin >> a;
    cout << "Enter number 2: ";
    cin >> b;
    cout << "Enter a String: ";
    cin >> str;
    sum = a + b;
    cout << "The sum is "<< sum << " : " << str;

    return 0;
}
```

**Output:**

```
Enter number 1: 14
Enter number 2: 25
Enter a String: Love C++ Prog.
The sum is 39 : Love
```

3

# Programming with C++

## Two versions of C++:

### Old version of C++:

```
#include <iostream.h>

int  main(){
    /*  program code */
    return 0;
}
```

➢ Includes **filename**

### New version of C++:

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

int main(){
    /* program code */
    return 0;
}
```

➢ Includes **stream** which is mapped to file by compiler

Bjarne Stroustrup
(1979)

| Filename used in<br>Old Version | File stream used in<br>New Version |
|---|---|
| iostream.h | iostream |
| string.h | cstring |
| math.h | cmath |
| graphics.h | cgraphics |

# Comments

➤ Multi-line comments

/* one or more lines of comments */

➤ Single line comments

// …

# Some differences between C and C++

| SL# | Area | C | C++ |
|-----|------|---|-----|
| 1. | Empty parameter list | **void** is mandatory.<br>`char fl(void);` | **void** is optional.<br>`char fl();` |
| 2. | Function prototype | Function prototype is optional but recommended. | All functions must be prototyped. |
| 3. | Returning a value | ➢A **non**-**void** function in not required to actually return a value. If it doesn't, a **garbage value** is returned.<br>➢ "**Default-to-int**" rule: If a function does not explicitly specify the return type, an integer return type is assumed. | ➢If a function is declared as returning a value, it **must** return a value.<br>➢ C++ has **dropped** the "default-to-int" rule. |
| 4. | Local variable declaration | Local variables are declared at the **start of a block**, prior to any action statement. | Local variables can be declared **anywhere**. |
| 5. | **bool** data type | - | C++ defines the **bool** data type and also **keywords true** and **false**. |

# I/O in Java Programming

✓ **System** class of java provides facilities like standard input, standard output and standard error streams. *System class can't be instantiated*.

✓ Java **Scanner** is a utility class to read user input or process simple regex-based parsing of file or string source. *Regex is a short form of regular expression.*

**JAVA Code**

```java
import java.util.Scanner;

public class Program7 {
    public static void main(String[] args) {
        Scanner myObj = new Scanner(System.in);

        System.out.print("Enter the first number: ");
        int a = myObj.nextInt();
        System.out.print("Enter the second number: ");
        int b = myObj.nextInt();

        int sum = a + b;
        System.out.println("The sum is "+ sum+ ".");
    }
}
```

**Scanner Methods:**
- `nextBoolean()`
- `nextByte()`
- `nextDouble()`
- `nextFloat()`
- `nextInt()`
- `nextLine`
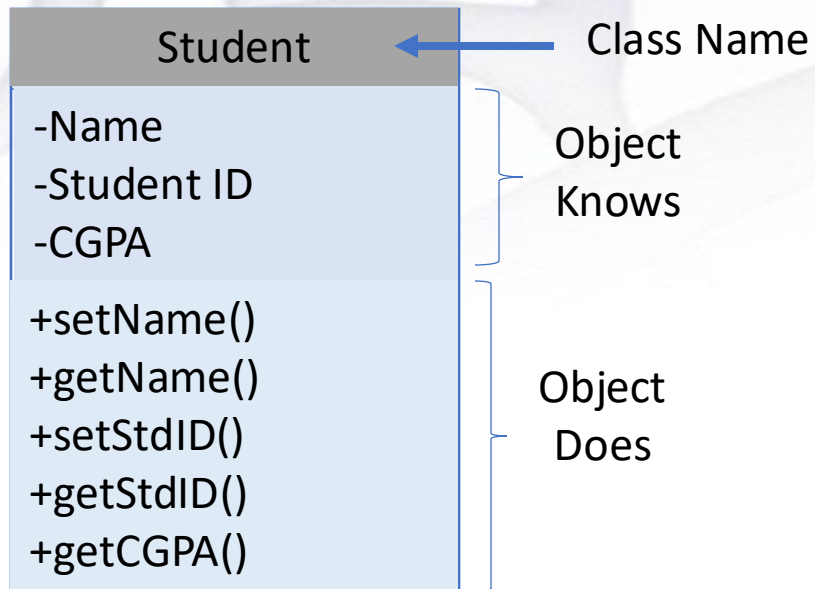- `nextLong()`
- `nextShort()`

**Output:**
```
Enter the first number: 12
Enter the second number: 34
The sum is 46.
```

**Notes for Java:**

**1.** Java does **not allow** any variable or function out of a class.

**2. main()** method must be within a class.

**3.** main() method must be **public** and **static**. **Why?**

**4.** A source file may contain multiple class or interface; but only one of them is public.

**2.** The source file name must be same with public class or interface name.

## Class vs. Object

| Student |
|---|
| -Name<br>-Student ID<br>-CGPA |
| +setName()<br>+getName()<br>+setStdID()<br>+getStdID()<br>+getCGPA() |

Class Name

Object Knows

Object Does

- means **private**
+ means **public**

**Examples**

Class: Student
Object: Lisa, Latif, Mahmud, Habiba

**Class**: A template or blueprint that defines the properties and behaviors of a type of objects.

**Object:** A specific instance of a class.

8

Program8.c

```c
#include <stdio.h>

typedef struct xx{
        char name[20];
        int rollno;
        double cgpa;
} Student;

int main() {
    Student karim;
    printf("Enter the name: ");
    gets(karim.name);
    printf("Enter Rollno: ");
    scanf("%d", &karim.rollno);
    printf("Enter CGPA: ");
    scanf("%lf", &karim.cgpa);

    printf("\nName: %s\n", karim.name);
    printf("Rollno: %d\n", karim.rollno);
    printf("CGPA: %.2lf\n", karim.cgpa);
}
```

C Code

**Output:**
Enter the name: Karim Ahmed
Enter Rollno: 12
Enter CGPA: 3.85

Name: Karim Ahmed
Rollno: 12
CGPA: 3.85

**Program8.cpp**

```cpp
#include <iostream>
#include <string.h>
using namespace std;

typedef struct xx{
private:
        char name[20];
        int rollno;
        double cgpa;
public:                          //setter - getter
        void setName(char *N){ strcpy(name, N);}
        char *getName(){return name;}
        void setRollno(int R){ rollno = R; }
        int getRollno(){ return rollno;};
        void setCGPA(double CGPA){ cgpa = CGPA;}
        double getCGPA(){ return cgpa;}
} Student;
```

**Output:**
Enter the name: Karim
Enter Rollno: 12
Enter CGPA: 3.85
Karim 12 3.85

Does it allow a name as **"Karim Ahmed"? Why?**

C++ Code

```cpp
int main() {
    Student karim;
    char iname[20];
    int irollno;
    double icgpa;

    cout << "Enter the name: ";
//  cin >> karim.name;   //Error – Why?
    cin >> iname;
    karim.setName(iname);

    cout << "Enter Rollno: ";
//  cin >> karim.rollno; //Error – Why ?
    cin >> irollno;
    karim.setRollno(irollno);

    cout << "Enter CGPA: ";
    cin >> icgpa;
    karim.setCGPA(icgpa);

    cout << karim.getName() << " ";
    cout << karim.getRollno();
    cout << " " << karim.getCGPA();
}
```

# Structure vs. Class

❖ By default, all members of a structure are **Public**

❖ By default, all members of a class are **Private**

**C++ Code**

```cpp
#include <iostream>
#include <string.h>
using namespace std;

typedef struct xx{
private:
        char name[20];
        int rollno;
        double cgpa;
public:                          //setter - getter
        void setName(char *N){ strcpy(name, N);}
        char *getName(){return name;}
        void setRollno(int R){ rollno = R; }
        int getRollno(){ return rollno;};
        void setCGPA(double CGPA){ cgpa = CGPA;}
        double getCGPA(){ return cgpa;}
} Student;
```

```cpp
#include <iostream>
#include <string.h>
using namespace std;

class Student{
        char name[20];
        int rollno;
        double cgpa;
public:                          //setter - getter
        void setName(char *N){ strcpy(name, N);}
        char *getName(){return name;}
        void setRollno(int R){ rollno = R; }
        int getRollno(){ return rollno;};
        void setCGPA(double CGPA){ cgpa = CGPA;}
        double getCGPA(){ return cgpa;}
};
```

# Use of Class in Java

- ❖ Java doesn't support **Structure, Pointer or Union.**
- ❖ **main** method – static, public, inside class

**StudentDemo.java**

**Java Code**

```java
import java.util.Scanner;

class Student{
    private String name;
    private int rollno;
    private double cgpa;

    //setter - getter
    public void setName(String N){name = N;}
    public String getName(){return name;}
    public void setRollno(int R){ rollno = R; }
    public int getRollno(){ return rollno;};
    public void setCGPA(double CGPA){ cgpa = CGPA;}
    public double getCGPA(){ return cgpa;}
}
```

**Output:**
```
Enter name: Abdur Rahim
Enter Rollno: 12
Enter CGPA: 3.85
Name: Abdur Rahim Rollno: 12 CGPA: 3.85
```

```java
public class StudentDemo{
    public static void main(String[] args){
        Student std = new Student();
        String iname = new String();
        Scanner obj = new Scanner(System.in);
        int irollno;
        double icgpa;

        System.out.print("Enter name: ");
        iname = obj.nextLine();
        std.setName(iname);

        System.out.print("Enter Rollno: ");
        irollno = obj.nextInt();
        std.setRollno(irollno);

        System.out.print("Enter CGPA: ");
        icgpa = obj.nextDouble();
        std.setCGPA(icgpa);

        System.out.print("Name: " + std.getName()+
            " Rollno: " + std.getRollno() +
            " CGPA: " + std.getCGPA());
    }
}
```

# Features of OOP

**Three Key Features of OOP**

| | |
|---|---|
| **Encapsulation** | ❖ **Wrap up data** and **functions/methods** together<br>❖ **Insulation** of data from **direct access** by the program – **data hiding** |

**Polymorphism**

**One Interface, multiple methods**
- ❖ **Function overloading**:
  - (1) Use a **single name** to **multiple methods**;
  - (2) **different** number and types of **arguments**.
- ❖ **Operator overloading**:
  - Use of **single operator** for **different** types of operands

**Inheritance**
- ❖ One class **inherits** the **properties** of another class
- ❖ provide **hierarchical classifications**
- ❖ Permits **reuse** of common **code** and **data**

- ❖ Representing essential features **without details**
- ❖ Class defines a list of **abstract attributes** (**data members**) and **methods** to operate on these attributes

**Abstraction**

# Encapsulation / Data Hiding

❖ **Wrap up** **data** and **functions/methods** together



**Class**

**Interface**
(Public method/data)
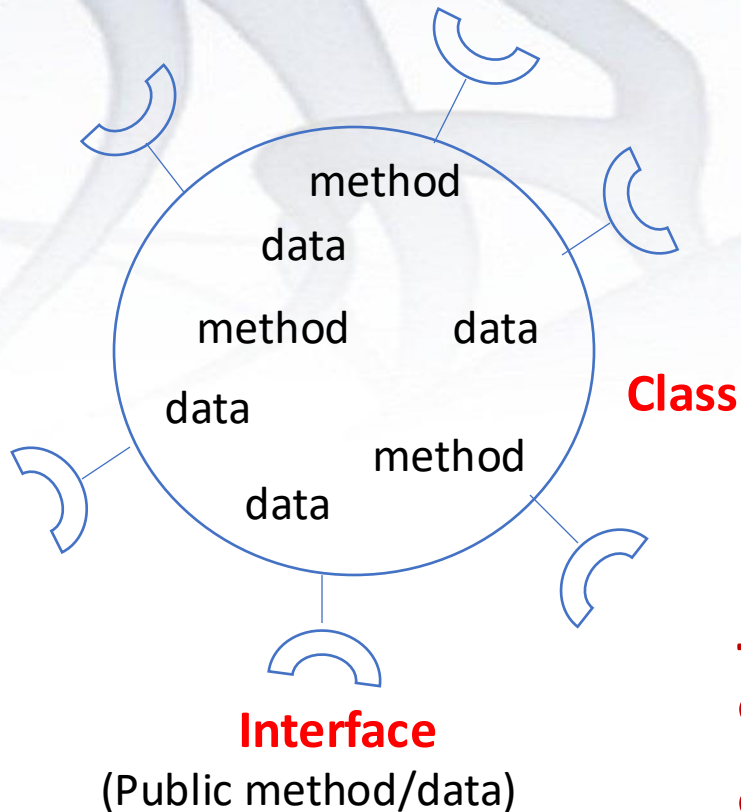
**C++ Code**

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

class myclass {
    int  a;
public:
    myclass();  // constructor
    int geta() { return a;}
};

myclass::myclass(){
    cout << "In constructor\n";
    a = 10;
}

int main(){
    myclass ob;

 //     cout << a;  // wrong -Why?
    cout << ob.geta();    // OK
    return 0;
}
```

**Java Code**

```java
class MyClass{
   private int a;
   MyClass(){
      System.out.println("In Constructor");
      a = 10;
   }
   public int getA(){return a;}
}

public class Encapsulation{
   public static void main(String[] args){
      MyClass ob = new MyClass();
//     System.out.println("a: "+a);  Error - Why?
      System.out.println(ob.getA());
   }
}
```

**Output:**
**In Constructor**
**10**

# Polymorphism

**One Interface, multiple methods**

❖ **Function overloading**: (supported by both C++ & Java)

   (1) Use a **single name** to **multiple methods**;

   (2) **different** number and types of **arguments**.

❖ **Operator overloading**: (Java doesn't support customized operator overloading)

   Use of **single operator** for **different** types of operands

**Function / Constructor Overloading**

**C++ Code**

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

class date {
    int month, day, year;
public:
    date(char *str);
    date(int d, int m, int y){
        day = d;
        month = m;
        year = y;
    }
    void show(){
        cout << day << '/' << month << '/' ;
        cout << year << '\n';
    }
};
```

```cpp
date::date(char *str){
        sscanf(str, "%d%*c%d%*c%d", &day, &month, &year);
}

int main() {
    date sdate("31/12/99");
    date idate(31, 12, 99);

    sdate.show();
    idate.show();
    return 0;
}
```

**Output:**

31/12/99

31/12/99

@Moh

# Polymorphism

```java
import java.time.LocalDate;

class MyDate{
    private int day;
    private int month;
    private int year;

    MyDate(String str){
        LocalDate date = LocalDate.parse(str);
        day = date.getDayOfMonth();
        month = date.getMonthValue();
        year = date.getYear();
    }

    MyDate(int newDay, int newMonth, int newYear){
        day = newDay;
        month = newMonth;
        year = newYear;
    }

    public void showMyDate(){
        System.out.println(day+"/"+month+"/"+year);
    }
}
```

**PolymorphismDemo.java**

**JAVA Code**

```java
public class PolymorphismDemo {
    public static void main(String[] args) {
        MyDate sDate = new MyDate("2024-05-12");
        MyDate iDate = new MyDate(23, 7, 2025);

        sDate.showMyDate();
        iDate.showMyDate();
    }
}
```

**Output:**
12/5/2024
23/7/2025

**Operator overloading:** **Shall be discussed in next.**
a = 4 + 6;
ob1 = ob2 + ob3; → **Not Supported in Java.**

# Inheritance

- ❖ One class **inherits** the **properties** of another class
- ❖ provide **hierarchical** **classifications**
- ❖ Permits **reuse** of common **code** and **data**

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

class base {          Superclass
    int x;
public:
    void setx(int n) { x = n; }
    void showx() { cout << x << '\n'; }
    int getx(){ return x; }
};

class derived: public base {    Subclass
    int y;
public:
    void sety(int n) { y = n; }
    void showy() {
        cout << y << '\n';
        cout << y+getx() <<'\n';
    }
};
```
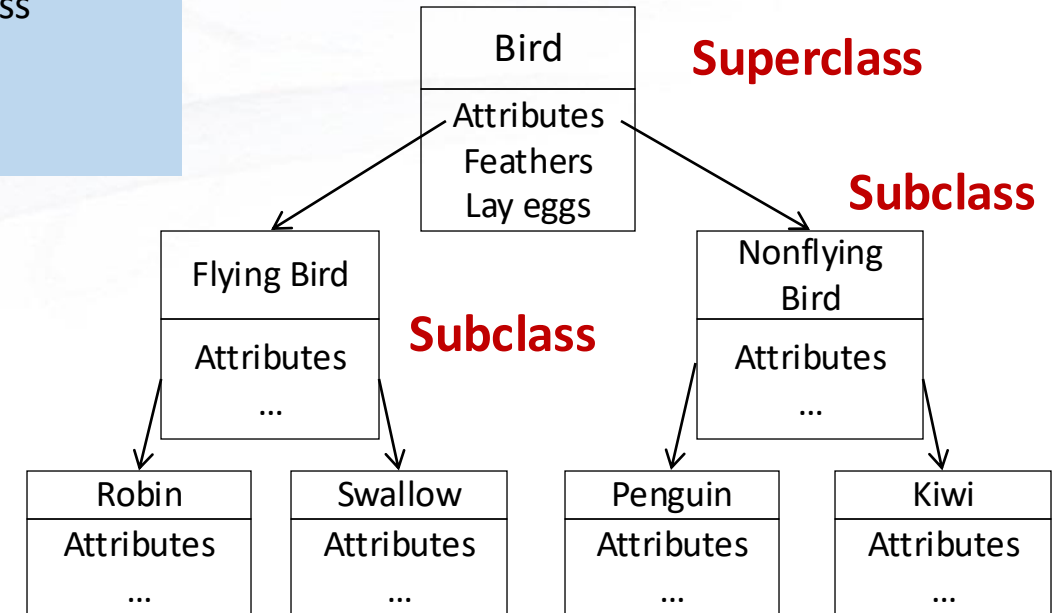
**C++ Code**

```cpp
int main(){
    derived ob;

    ob.setx(10);
    ob.sety(20);
    ob.showx();
    ob.showy();
    return 0;
}
```

**Output:**
10
20
30



Bird — **Superclass**
Attributes
Feathers
Lay eggs

**Subclass**

Flying Bird — **Subclass**
Attributes
…

Nonflying Bird
Attributes
…

Robin
Attributes
…

Swallow
Attributes
…

Penguin
Attributes
…

Kiwi
Attributes
…

17

```java
class Base{
    private int x;

    public void setX(int newX){ x = newX;}
    public int getX(){ return x;}
    public void showX(){System.out.println("X = "+x);}
}


class Derived extends Base{
    private int y;
    public void setY(int newY){ y = newY;}
    public void showY(){
        System.out.println("Y= "+y + "  X= "+ getX());
    }
}
```

**Java Code**

```java
public class InheritanceDemo {
    public static void main(String[] args) {
        Derived obj = new Derived();

        obj.setX(10);
        obj.setY(20);
        obj.showX();
        obj.showY();
    }
}
```

**Output:**
```
X = 10
Y= 20   X= 10
```

# Abstraction

**Abstract Class and Method:**

- **Abstract class** is a superclass without a complete implementation of every method.
  - There can be **no objects** of an abstract class.
  - Abstract can be used to create **object references**.
- **Abstract method** refers to **subclasser responsibility** to override it, otherwise, it will report a **warning message**.
  - **Constructor** and **static method** cannot be **Abstract.**

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

class Figure{
protected:
    double dim1, dim2;
public:
    Figure(double d1, double d2){
        dim1 = d1;
        dim2 = d2;
    }
    virtual double area() = 0;  // Pure virtual function
};
```

**C++ Code**

```cpp
class Rectangle: public Figure{
public:
    Rectangle(double d1, double d2): Figure(d1, d2){}
    double area(){
        return dim1 * dim2;
    }
};

class Triangle: public Figure{
public:
    Triangle(double d1, double d2): Figure(d1, d2){}
    double area(){
        return dim1 * dim2 / 2;
    }
};

int main(){
    Figure *p;
    Rectangle r(10, 7);
    Triangle t(10, 5);

    p = &r;
    cout << "Rectangle Area: " << p->area() << endl;

    p = &t;
    cout << "Triangle Area: " << p->area() << endl;

    return 0;
}
```

**Java Code**

```java
abstract class Figure {
    double dim1, dim2;
    Figure(double a, double b){ dim1 = a; dim2 = b;}
    abstract double area();
    void show(){System.out.println("Abstract");}
}

class Rectangle extends Figure {
    Rectangle(double a, double b) { super(a, b);}
    double area(){ return dim1*dim2;}
    void show(){
        System.out.println("Rectangle Area: "+area());
    }
}


class Triangle extends Figure {
    Triangle(double a, double b) {super(a, b);}
    double area(){ return 0.5*dim1*dim2;}
    void show(){
        System.out.println("Triangle Area: "+area());
    }
}
```

```java
public class Main {
    public static void main(String[] args) {
        Rectangle r = new Rectangle(10,7);
        Triangle t = new Triangle(10, 5);
        Figure figref;

        figref = r;
        figref.show();

        figref = t;
        figref.show();
    }
}
```

**Output:**
Rectangle Area: 70.0
Triangle Area: 25.0

**Abstract class**: private /public /protected variable and method **declaration / implementation** is allowed.
**Interface**: only static final variable and method **declaration** is allowed. No method implementation.

### Fly.java

```java
public interface Fly {
    public String fly();
}

class Dog implements Fly{
    public String fly(){ return "I cannot fly."; }
    public String MakeSound(){ return "I sound berk.";}
}

class Bird implements Fly{
    public String fly() { return "I fly in the sky.";}
    public String MakeSound(){ return "I sound chi-chi.";}
}

class Biman implements Fly{
    public String fly(){
        return "I fly too high in the sky.";
    }
}
```

### InterfaceDemo.java

```java
public class InterfaceDemo {
    public static void main(String[] args) {
        Fly doggy = new Dog();
        Bird sweety = new Bird();
        Fly gogon = new Biman();

        System.out.println("I am a dog."+ doggy.fly()+ ((Dog) doggy).MakeSound());
        System.out.println("I am a bird."+sweety.fly() + sweety.MakeSound());
        System.out.println("I am a Biman." + gogon.fly());
    }
}
```

**Output:**
I am a dog.I cannot fly.I sound berk.
I am a bird.I fly in the sky.I sound chi-chi.
I am a Biman.I fly too high in the sky.

21

**Scope Resolution Operator** is used for **two purposes** in C++:

To access a **hidden global variable**

**C++ Code**

**Program5.cpp**

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

int count = 0;

int main(void) {
    int count = 0;
    ::count = 1;
    count = 2;
    cout << "Global: " << ::count << endl;
    cout << "Local: " << count;
    return 0;
}
```

**Output:**
```
Global: 1
Local: 2
```

To access a **hidden member class** or **member variable** with a **class**

**C++ Code**

**Program6.cpp**

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

class X {
public:
    static int count;
};

int X::count = 10;

int main () {
    cout << "count: " << X::count;
}
```

**Output:**
```
count: 10
```

22

# Namespaces in C++

- A **namespace** is a **declarative region** that **localizes** the names of identifiers to **avoid name collisions.**
- **Three ways** of using **namespace.**

```cpp
#include <iostream>
#include <string>

int main(){
    std::string str;

    std::cout << "Enter a string: ";
    getline(std::cin, str);
    std::cout << str << std::endl;

    return 0;
}
```

**Output:**
```
Enter a string: Love C++
Love C++
```

**C++ Code**

```cpp
#include <iostream>
#include <string>
using std::cin;
using std::cout;
using std::endl;
using std::string;

int main(){
    string str;

    cout << "Enter a string: ";
    getline(cin, str);
    cout << str << endl;

    return 0;
}
```

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

int main(){
    string str;

    cout << "Enter a string: ";
    getline(cin, str);
    cout << str << endl;

    return 0;
}
```

- **Java programming uses separate namespace for each package.**

➤ **C++ library** is defined within its **own namespace**, **std**.

➤ The general form of defining namespace is shown as:

```
namespace name{

}
```

**Unnamed namespace** is declared with the scope of **a single file**.

```
namespace{

}
```

➤ There are **two general form** of using statement:

```
using namespace name;
using name::member;
```

➤ Declaration of new namespaces are required for creating library of **reusable code** or code that requires **widest portability**.

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

namespace {
    void displayMsg() {
        cout << "unnamed namespace." << endl;
    }

    int add(int a, int b) {
        return a + b;
    }
}
```

```
int main() {
    displayMsg();
    int result = add(5, 3);
    cout << "Result: " << result << endl;
    return 0;
}
```

**Output:**
```
unnamed namespace.
Result: 8
```

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

namespace firstNS{
    class MyClass {
        int i;
    public:
        MyClass(int n) { i = n; }
        void setI(int n) { i = n; }
        int getI() { return i; }
    };

    const char* str = "Hello from firstNS!";
    int counter = 0;
}

namespace secondNS{
    int x, y;
}
```

**Output:**

```
I: 99
Hello from firstNS!
10 9 8 7 6 5 4 3 2 1
X: 10
Y: 20
```

```cpp
int main() {
    firstNS::MyClass ob(10);

    ob.setI(99);
    cout << "I: " << ob.getI() << endl;
    using firstNS::str;
    cout << str << endl;

    using namespace firstNS;
    for( counter = 10; counter; counter--)
        cout << counter << " ";
    cout << endl;

    secondNS::x = 10;
    secondNS::y = 20;

    cout << "X: " << secondNS::x << endl;
    cout << "Y: " << secondNS::y << endl;

    return 0;
}
```

➤ **There can be more than one namespace declaration of the same name in the same file or different files.**

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

namespace Demo{
    int a;
}
int x;

namespace Demo{
    int b;
}
```

```cpp
int main(){
    using namespace Demo;

    a = b = x = 100;
    cout << "a: " << a << endl;
    cout << "b: " << b << endl;
    cout << "x: " << x << endl;

    return 0;
}
```

**Output:**
```
a: 100
b: 100
x: 100
```

26

# Packages in Java

```java
import java.time.LocalDate;

class MyDate{
    private int day;
    private int month;
    private int year;

    MyDate(String str){
        LocalDate date = LocalDate.parse(str);
        day = date.getDayOfMonth();
        month = date.getMonthValue();
        year = date.getYear();
    }

    public void showDate(){
        System.out.println("Date: "+ day+"/"+month+"/"+year);
    }
}
```

```java
public class Main {
    public static void main(String[] args) {
        MyDate date = new MyDate("2024-05-12");
        date.showDate();
    }
}
```

**Output:**
```
Date: 12/5/2024
```

**Some more common packages in Java:**
➢ `import java.lang.*;`
➢ `import java.io.*;`
➢ `import java.util.*;`

`import java.lang.*;` is **Automatic.**

- ✓ Hierarchical Structure of Packages
- ✓ A package contains **classes** and other **subordinate packages**
- ✓ **Leave** of a hierarchy is a **class name**.

```java
import java.util.*;
class MyDate extends Date{
    ..............................................
}

is equivalent to

class MyDate extends java.util.Date{
    .........................................
}
```

➢ There are **no core Java classes** in the **unnamed default package**; all of the standard classes are stored in some named package.

➢ Java is useless without much of the functionality in **java.lang** and so, it is implicitly imported by the compiler for all programs.

➢ If a class with the same name exists in **two different imported packages**, then **compiler will remain silent** and generate a **compile-time error** if the name is not used explicitly with the class specifying its package.

➢ when a package is imported, only those items within the package declared as **public** will be available to **non-subclasses** in the importing code.

➢ Packages act as **containers** for classes and other subordinate packages. Classes act as containers for data and code.

➢ To create a package, simply include **package** command as the first statement in java source file.

<pre style="color:red">          package mypack;</pre>

➢ If **package** command is omitted, the class names are put into the default package, which has no name.

➢ Creating hierarchy of package.

<pre style="color:red">          package mypack.prog.myprog;</pre>

which is stored as mypack\prog\myprog in windows environment.

➢ A package cannot be renamed without renaming the directory in which classes are stored.

```java
package mypack;

class Balance{
    String name;
    double bal;

    Balance(String n, double b){
        name = n;
        bal = b;
    }
    void show(){
        System.out.println(name + ": $" + bal);
    }
}


class AccountBalance {
    public static void main(String[] args) {
        Balance[] current = new Balance[3];
        current[0] = new Balance("Jerry", 123.23);
        current[1] = new Balance("Tell", 157.02);
        current[2] = new Balance("Tom", -12.33);
        for(int i=0; i<3; i++) current[i].show();
    }
}
```

Command line:

```
java mypack.AccountBalance
```

Incorrect Command:
```
java AccountBalance
```

`AccountBalanace` must be qualified with its package name.

**Output:**
```
Jerry: $123.23
Tell: $157.02
Tom: $-12.33
```

30

## Setter - Getter

```
class Student{
    private String name;
    private int stdID;

    public void setName(String newname){
        name = newname;
    }

    public String getName(){
        return name;
    }

    public void setStdID(int newID){
        stdID = newID;
    }

    public int getStdID(){
        return stdID;
    }
}
```

## Naming Convention

1. Every name should be meaningful.
```
Box ix;          // Very bad choice
Box b;           // Bad choice
Box mybox;       // Bad choice
Box myBox;       // Good choice
```

2. Class names shall be started with Upper Case; method and variable names shall be started with Lower Case.
```
class myclass;          // Bad choice
class MyClass;          // Good choice
double Val;             // Bad choice
int ID;                 // Bad choice
int stdID;              // Good choice
double Count(){…}       // Bad choice
double countName(){…}   // Bad choice
```

# Using Initializer List in C++

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

class Point{
    int x, y;
public:
    Point(int a = 0, int b = 0){
        x = a;
        y = b;
    };
    void display(){
        cout << "x: " << x;
        cout << ", y: " << y << endl;
    }
};

int main(){
    Point p1(10, 20);
    p1.display();
    return 0;
}
```

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

class Point{
    int x, y;
public:
    Point(int a = 0, int b = 0): x(a), y(b) {}
    void display(){
        cout << "x: " << x;
        cout << ", y: " << y << endl;
    }
};

int main(){
    Point p1(10, 20);
    p1.display();
    return 0;
}
```

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

class Point{
    int x, y;
public:
    Point(int a = 0, int b = 0): x(a), y(b){
        cout << "x: " << x;
        cout << ", y: " << y << endl;
    }
};

int main(){
    Point p1(10, 20);
    return 0;
}
```