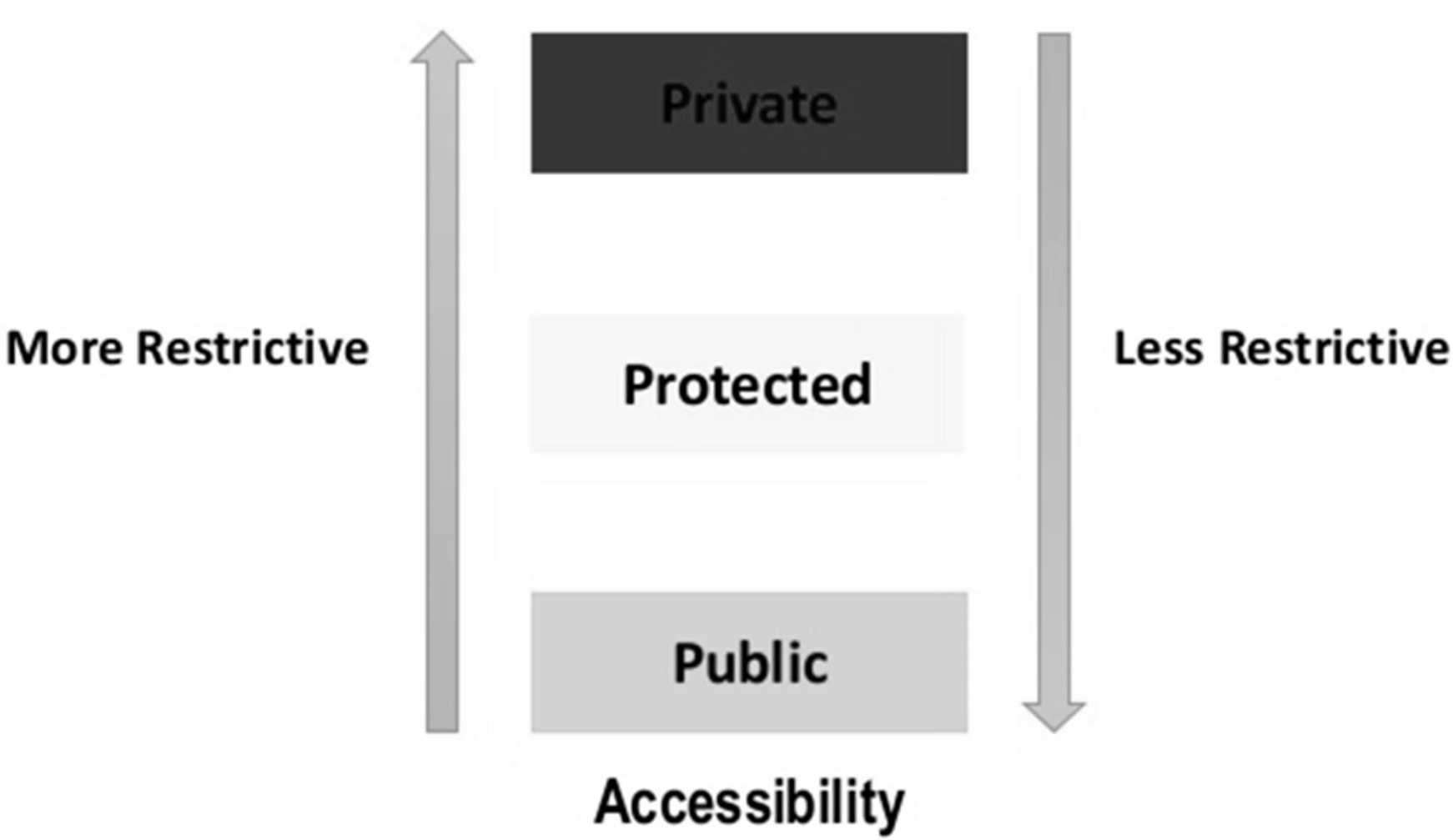
Object Oriented Programming (OOP)

Access Specifiers in C++



Access Specifiers in C++

- **Public**
- Public, means all the class members declared under public will be available to everyone. The data members and member functions declared public can be accessed by other classes too. Hence there are chances that they might change them. So the key members must not be declared public.

```
class PublicAccess
{
 public:                 // public access specifier
 int x;                  // Data Member Declaration
 void display();         // Member Function Declaration
}
```

Access Specifiers in C++

- **Private**
- Private keyword, means that no one can access the class members declared private outside that class. If someone tries to access the private member, they will get a compile time error. By default class variables and member functions are private.

```
class PrivateAccess
{
 private:                  // private access specifier
 int x;                   // Data Member Declaration
 void display();          // Member Function Declaration
}
```

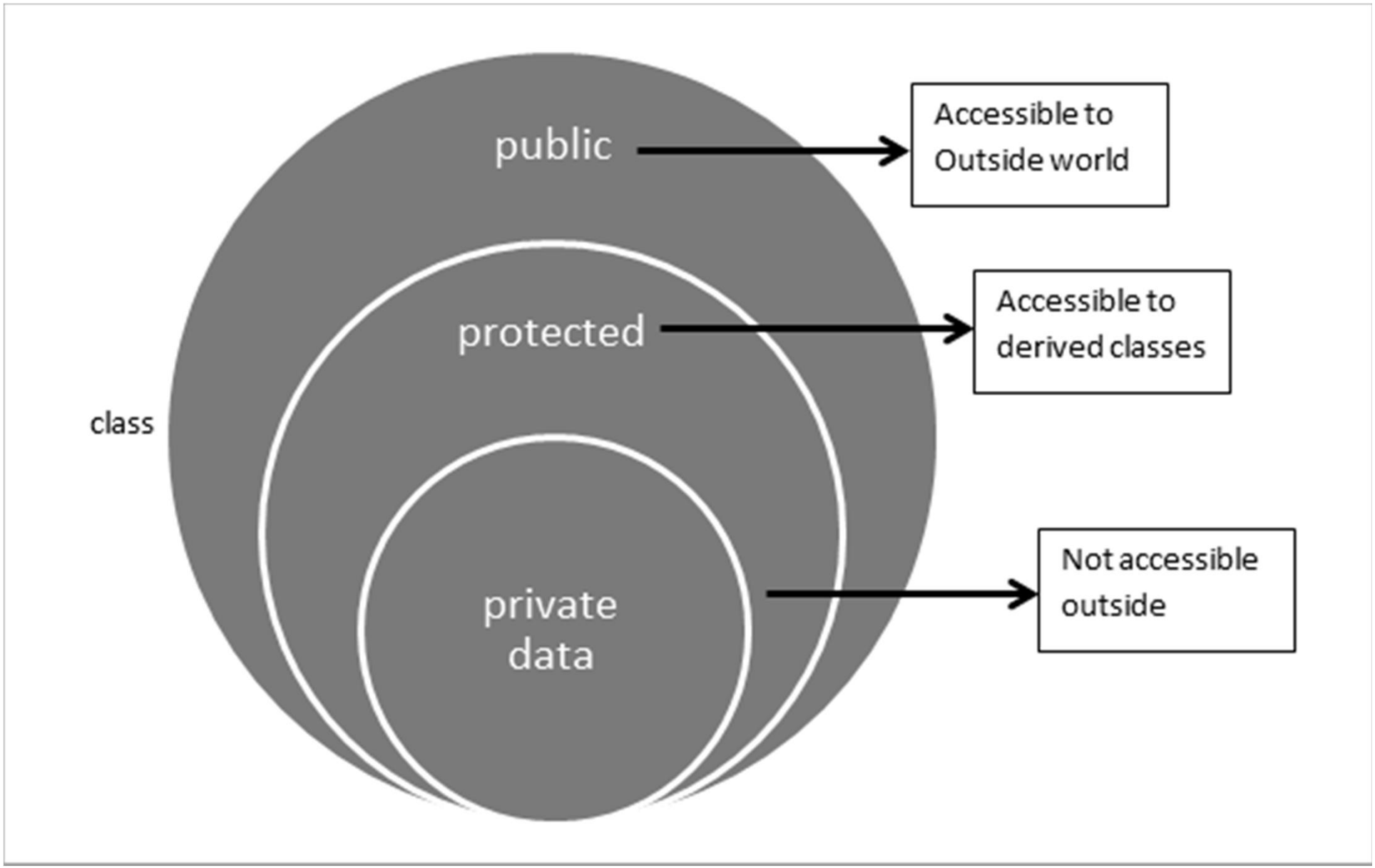
Access Specifiers in C++

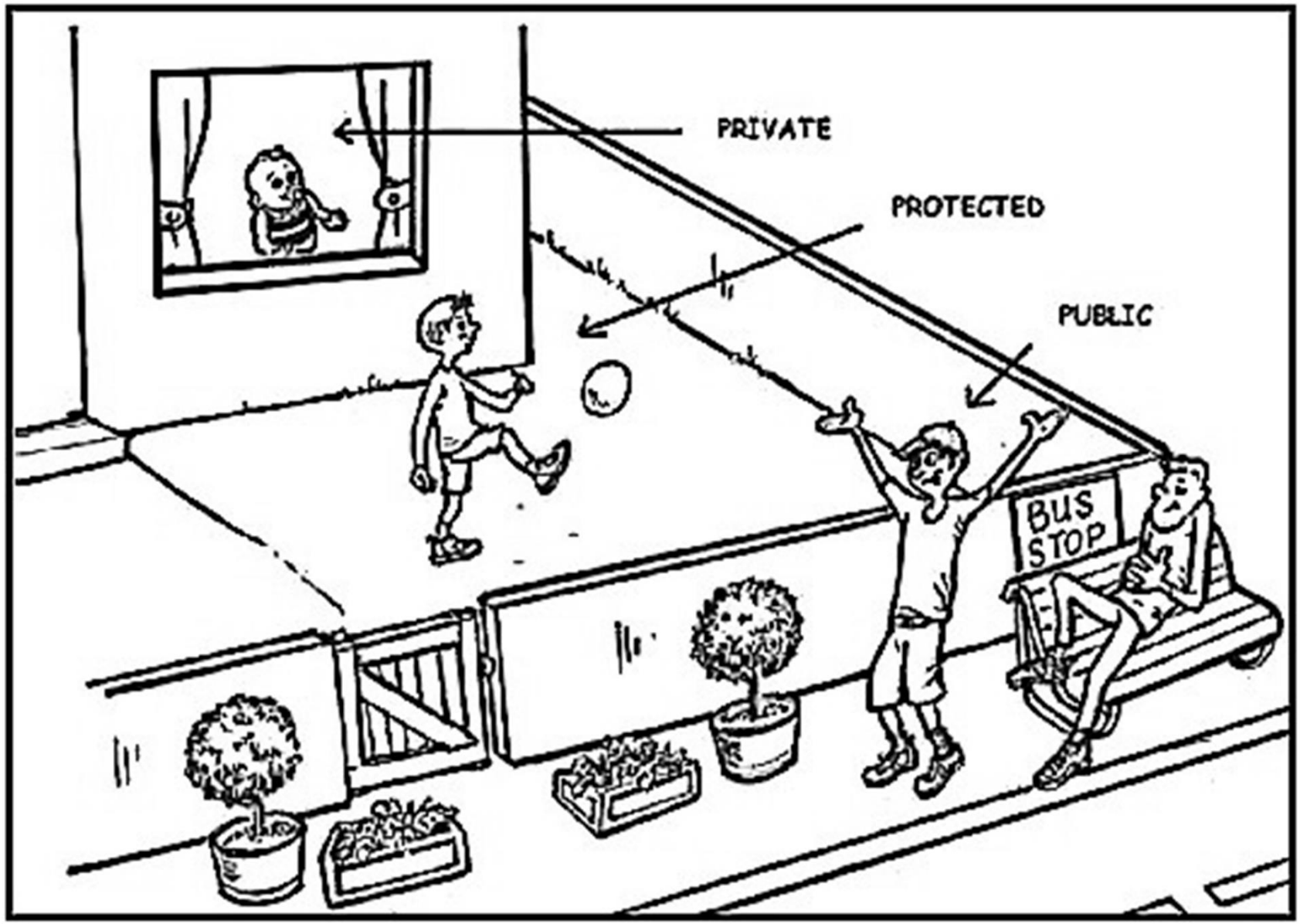
- **Protected**
- Protected, is the last access specifier, and it is similar to private, it makes class member inaccessible outside the class. But they can be accessed by any subclass of that class. (If class A is inherited by class B, then class B is subclass of class A. We will learn this later.)

```
class ProtectedAccess
{
 protected:            // protected access specifier
 int x;                // Data Member Declaration
 void display();       // Member Function Declaration
}
```

Access Specifiers

Access Specifier	Accessible from own class	Accessible from derived class	Accessible from objects from outside the class
public	Yes	Yes	Yes
protected	Yes	Yes	No
private	Yes	No	No


class
public
protected
private
data
Accessible to
Outside world
Accessible to
derived classes
Not accessible
outside


PRIVATE
PROTECTED
PUBLIC
BUS STOP

Example:

```
class MyClass {
  public:    // Public access specifier
    int x;   // Public attribute
  private:   // Private access specifier
    int y;   // Private attribute
};

int main() {
  MyClass myObj;

  myObj.x = 25;  // Allowed (public)

  myObj.y = 50;  // Not allowed (private)
  return 0;
}
```

Default Access Specifier

- When no access specifier mentioned then default access specifier is "**private**"
- **Example**

```
Class Student
{
     int regno;
    string name
}
```

Equivalent

```
Class Student
{
    private:
    int regno;
    string name
}
```

Overloading in C++

- What is overloading
  - Overloading means assigning multiple meanings to a function name or operator symbol
  - It allows multiple definitions of a function with the same name, but different signatures.
- C++ supports
  - Function overloading
  - Operator overloading

Function Overloading in C++

- **What is overloading**
- Doing work in different way is called overloading.

- **What is function overloading**
- Two ore more than two function with same name and with different signature is called function overloading.
- Signature may be vary on the basis of number of argument or data type of argument or order of argument.

Function Overloading

- Is the process of using the same name for two or more functions
- Requires each redefinition of a function to use a different function signature that is:
  - different types of parameters,
  - or sequence of parameters,
  - or number of parameters
- Is used so that a programmer does not have to remember multiple function names

Function Overloading

- Two or more functions can have the same name but different parameters
- Example:

```
int max(int a, int b)
{
    if (a>= b)
        return a;
    else
        return b;
}
```

```
float max(float a, float b)
{
    if (a>= b)
        return a;
    else
        return b;
}
```

Function Overloading (Syntax)

```
class class_Name {
Returntype  method()
{
.....
.....
}
Returntype  method(datatype1 variable1) {
.....
.....
}
Returntype  method(datatype1 variable1, datatype2 variable2) {
.....
.....
}
};
```

Function Overloading (continued)

- Correct function overloading:

```
void functionXYZ()
void functionXYZ(int x, double y)
void functionXYZ(double one, int y)
void functionXYZ(int x, double y, char ch)
```

- Syntax error:

```
void functionABC(int x, double y)
int functionABC(int x, double y)
```

Function Overloading in C++

```
// Error code
int test(int a) { }
double test(int b){ }
```

- The number and type of arguments passed to these two functions are same even though the return type is different. Hence, the compiler will throw error.

By Changing Number of Arguments

```
class Addition {
        public:
        void sum(int a, int b) {
                cout<<a+b;
        }
        void sum(int a, int b, int c) {
                cout<<a+b+c;
        }
};
int main() {
   Addition obj;
   obj.sum(10, 20); //Calling the function having two Parameters
  cout<<endl;
  obj.sum(10, 20, 30); //Calling the function having three Parameters
}
```

By Changing the Data Type

```
class Addition {
        public:
        void sum(int a, int b) {
                cout<<a+b;
        }
        void sum(float a, float b) {
                cout<<a+b;
        }
};
int main() {
        Addition obj;
        obj.sum(10, 20);
        cout<<endl;
        obj.sum(30, 40);
}
```