CSC127 Inheritance

Inheritance

- “Inheritance is the mechanism to provides the power of **reusability** and **extendibility**.”
- “Inheritance is the process by which one **object can acquire the properties of another object**.”
- “Inheritance is the process by which **new** classes called ***derived*** **classes** are created from **existing** classes called ***base*** **classes**.”
- Allows the creation of **hierarchical** classifications.

Base Class

- **“Base class** is a class which defines those **qualities common to all objects** to be derived from the base.”

- The base class represents the most **general description.**

- A class that is **inherited** is referred to as a base class.

Derived Class

- “The classes derived from the base class are usually referred to as **derived classes**.”

- “A **derived class** includes **all** features of the generic base class and then adds **qualities specific** to the derived class.”

- The class that does the **inheriting** is called the derived class.

Inheritance

Note:

Derived class can be used as a **base class** for another derived class.

- In C++, inheritance is achieved by allowing one class to incorporate another class into its declaration.

Inheritance

- **Syntax:**

  class derived_class**: Acesss_specifier** base_class

  { };

- **Example:**

  class CRectangle**: public** Cpolygon{ };

  class CTriangle**: public** Cpolygon{ };

Inheritance & Access Specifier

Access	public	protected	private
Members of the same class	Yes	Yes	Yes
Members of derived classes	Yes	Yes	No
Non-members	Yes	No	No

Public base class Inheritance

- All **public** members of the base class become **public** members of the derived class.

- All **protected** members of the base class become **protected** members of the derived class.

Private base class Inheritance

- All **public** and **protected** members of the base class become **private** members of the derived class.

- But **private** members of the base class remain **private to base class** only, **not accessible** to the derived class.

Protected Members of Base Class

- Member is not accessible by other **non member** elements of the program.

- The base class' **protected members** become **protected members** of the derived class and are, therefore, accessible by the derived class.

Protected Base-Class Inheritance

- All **public** and **protected** members of the base class become **protected** members of the derived class.

- All **public** members of the base class become **unavailable** to **main()** function.

- All **private** members of the base class become **unavailable** to the derived class.

Inheritance & Access Specifier

Access	public	protected	private
Members of the same class	Yes	Yes	Yes
Members of derived classes	Yes	Yes	No
Non-members	Yes	No	No

Inheriting Multiple Base Classes

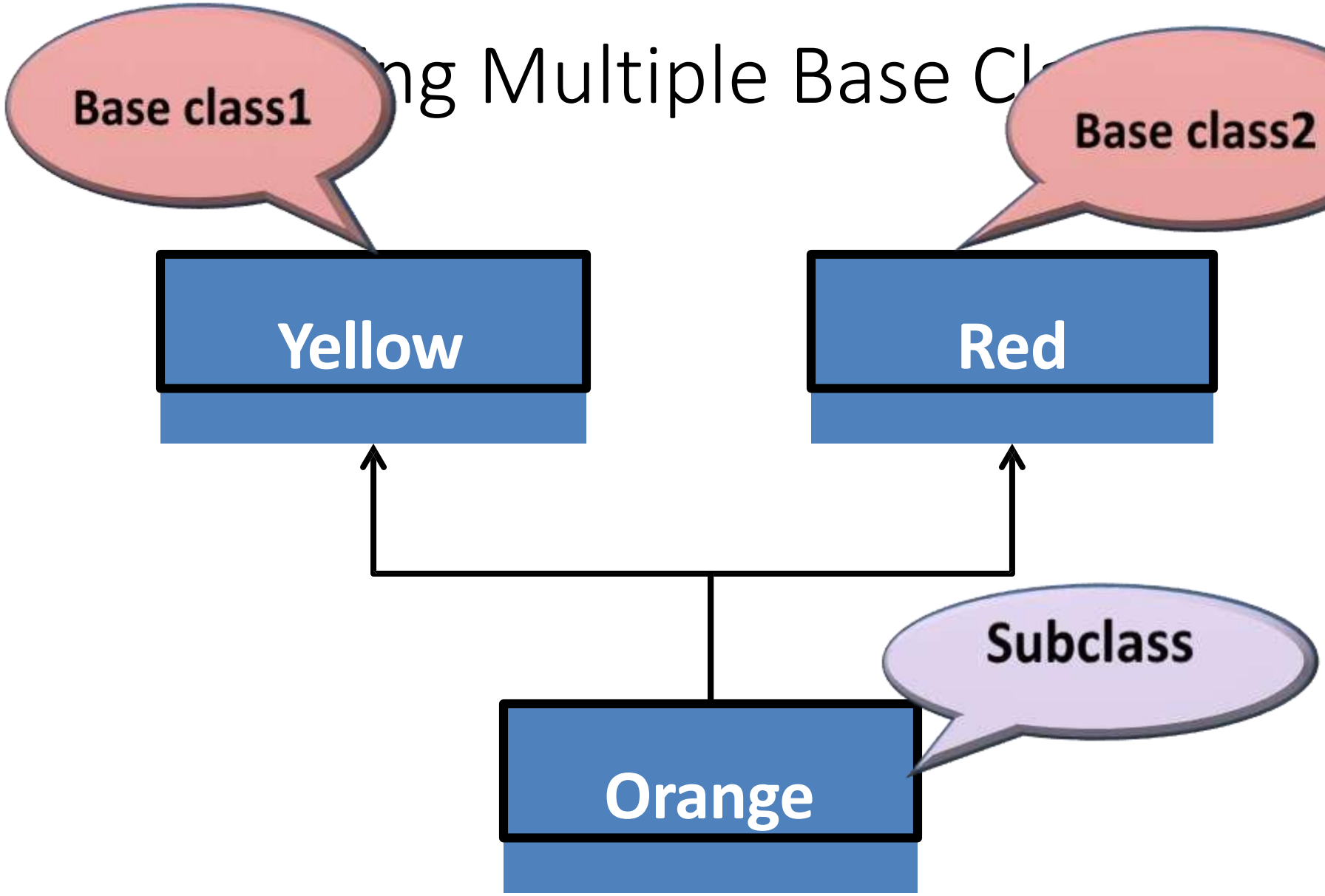
- **Syntax:**

  class derived: Acess_specifier base1,
  Acess_specifier base2
  { };

- **Example:**

  class Orange: Acess_specifier Yellow,
  Acess_specifier Red
  { };

ng Multiple Base Cl

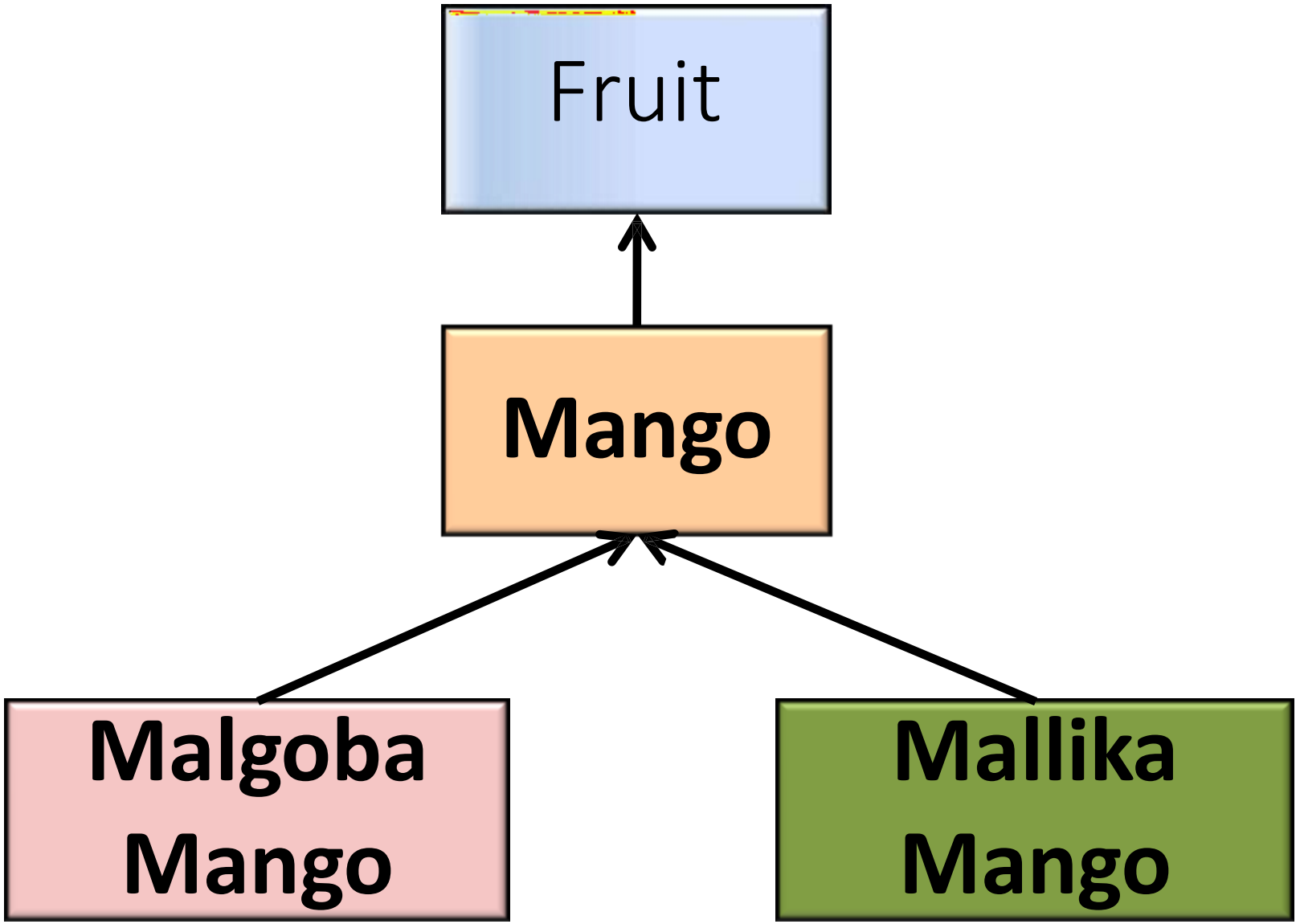


Constructors, Destructors & Inheritance

- Constructor functions are executed in their **order of derivation**.

- Destructor functions are executed in **reverse order of derivation**.

Inheritance



Constructors, Destructors & Inheritance

- When an object of a derived class is created,

  if the base class contains a constructor, it will be called first, followed by the derived class' constructor.

- When a derived object is destroyed, its destructor is called first, followed by the base class' destructor.

Passing Parameters to Base-Class Constructors

- Making use of an expanded form of the **derived class's constructor** declaration, we can pass arguments to one or more base-class constructors.

- **Syntax:**

```
derived-constructor(arg-list) : base1(arg-list),
                                base2(arg-list), … baseN(arg-list)
{   // body of derived constructor    }
```

Passing Parameters to Base-Class Constructors

- As we are **arguments** to a **base-class** constructor are passed via **arguments** of the **derived class**' constructor.

- Even if a **derived class**‘ constructor does **not** use any **arguments**, we need to **declare** a **constructor** as if the base class requires it.

- The **arguments passed** to the **derived class** are simply passed along to the **base class constructor**.

Granting Access

- When a base class is inherited as **private:**

  - all **public** and **protected** members of that class become **private members** of the derived class.

- But in some certain circumstances, we want to **restore** one or more inherited members to their **original access** specification.

Granting Access

- To accomplish this :

using

access declaration

Granting Access

- **using** statement**:**

  is designed primarily to support namespaces.

- **Access declaration:**

  restores an inherited member's access specification

  Syntax:

  base_class_name::member;

Granting Access

- Access declaration is done under the appropriate **access heading** in the **derived class'** declaration.

Note:

No type declaration is required.

Granting Access


```
class base {
public:
int j;
};

class derived: private base {
public:
// here is access declaration
base::j;
};
```