Object Oriented Programming

7 Abstract Classes

-

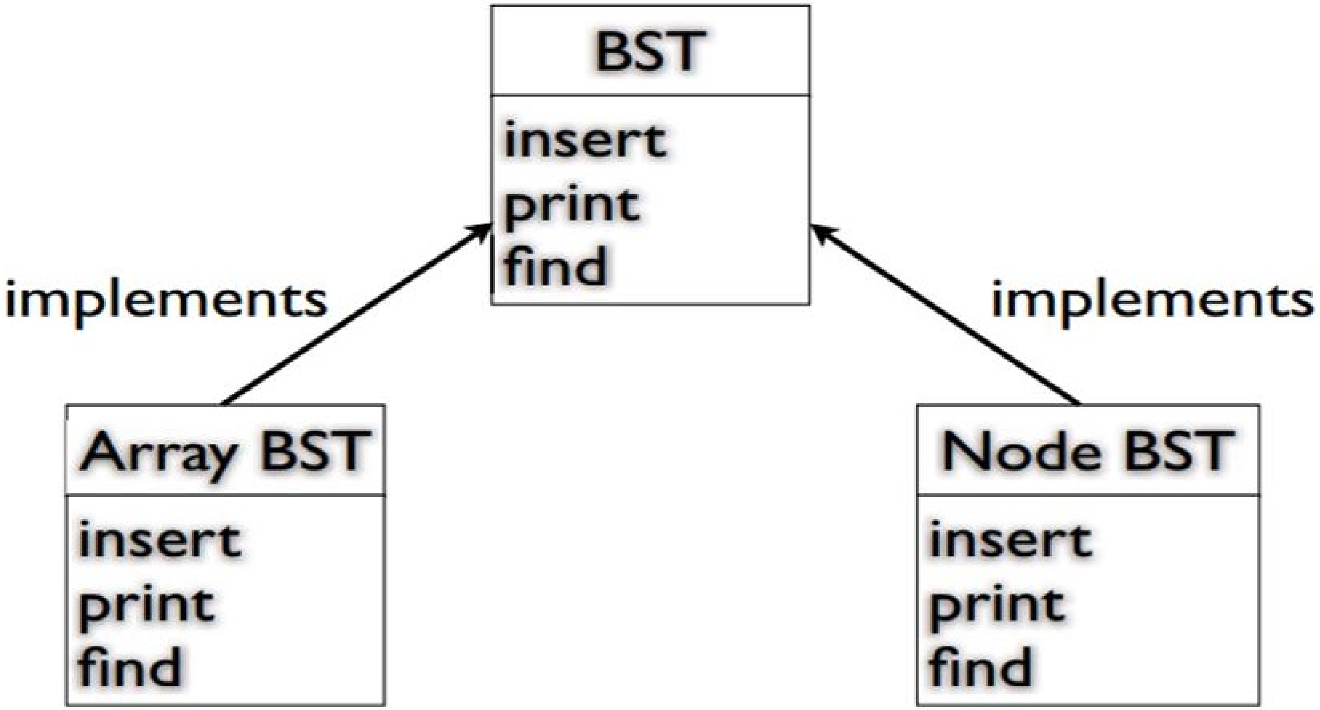
Abstract Methods

- Sometimes, we want to inherit only declarations, not definitions.

- A method without an implementation is called an **abstract method.**

- **Abstract methods** are often used to create an **interface**

Abstract Methods (Example)



- We Can provide multiple implementations to BST (Binary Search Tree).
- We decouple the client from the implementations

Defining Abstract Methods in C++ (1)

Abstract Classes 7

- To define an abstract method, we use pure virtual functions.

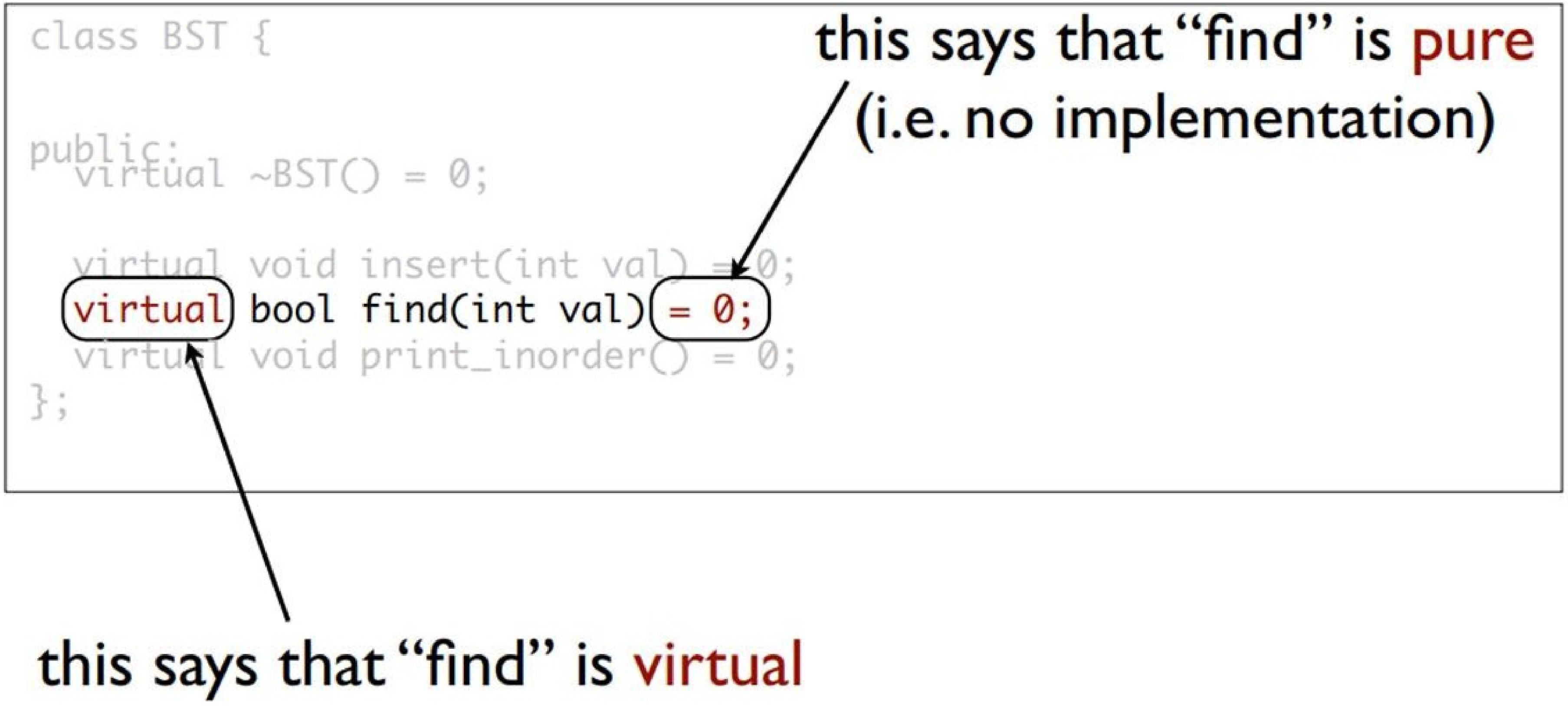
```
class BST {
public:

virtual void insert(int val) = 0;
virtual bool find(int val) {…..};
virtual void print_inorder()  {…..};
 };
```

Syntax:

```
virtual Type Funtion_name= 0;
```

Defining Abstract Methods in C++ (2)



```
class BST {

public:
  virtual ~BST() = 0;

  virtual void insert(int val) = 0;
  virtual bool find(int val) = 0;
  virtual void print_inorder() = 0;
};
```

this says that “find” is pure (i.e. no implementation)

this says that “find” is virtual

Question: Can we have **non-virtual pure** functions?

Abstract Classes in C++

An **abstratc base class**:

- ✓ is a class with one or more pure virtual functions

- ✓ it cannot be instantiated

- ✓its subclass must implement the all of the pure virtual functions (or itself become an abstract class)

Abstract Classes in C++ (Example)

```
class Account {
public: Account( double d );
virtual double GetBalance();
virtual void PrintBalance() = 0;
private:
double _balance;
};
class Derived:public Account
{
public: void PrintBalance() {
cout << "Implementation of Virtual Function in Derived class"; }
};
```

Constructors in Abstract Classes

- Does it make sense to define a constructor? Since, the class will never be instantiated!

- Yes! we should still create **a constructor** to initialize its members, since they will be inherited by its subclass.

Destructors in Abstract Classes

- Does it make sense to define a destructor? Since, the class will never be created in the first place!

- Yes! Always define a **virtual destructor** in the base class, to make sure that the destructor of its subclass is called!

Pure Virtual Destructor

- We can also define a destructor as **pure.** But must also provide a function body. Why?

```
class BST {
public:
virtual ~BST() = 0;
virtual void insert(int val) {….};
virtual bool find(int val) = 0;
virtual void print_inorder() {…..};
 };
BST::~BST() {}
```

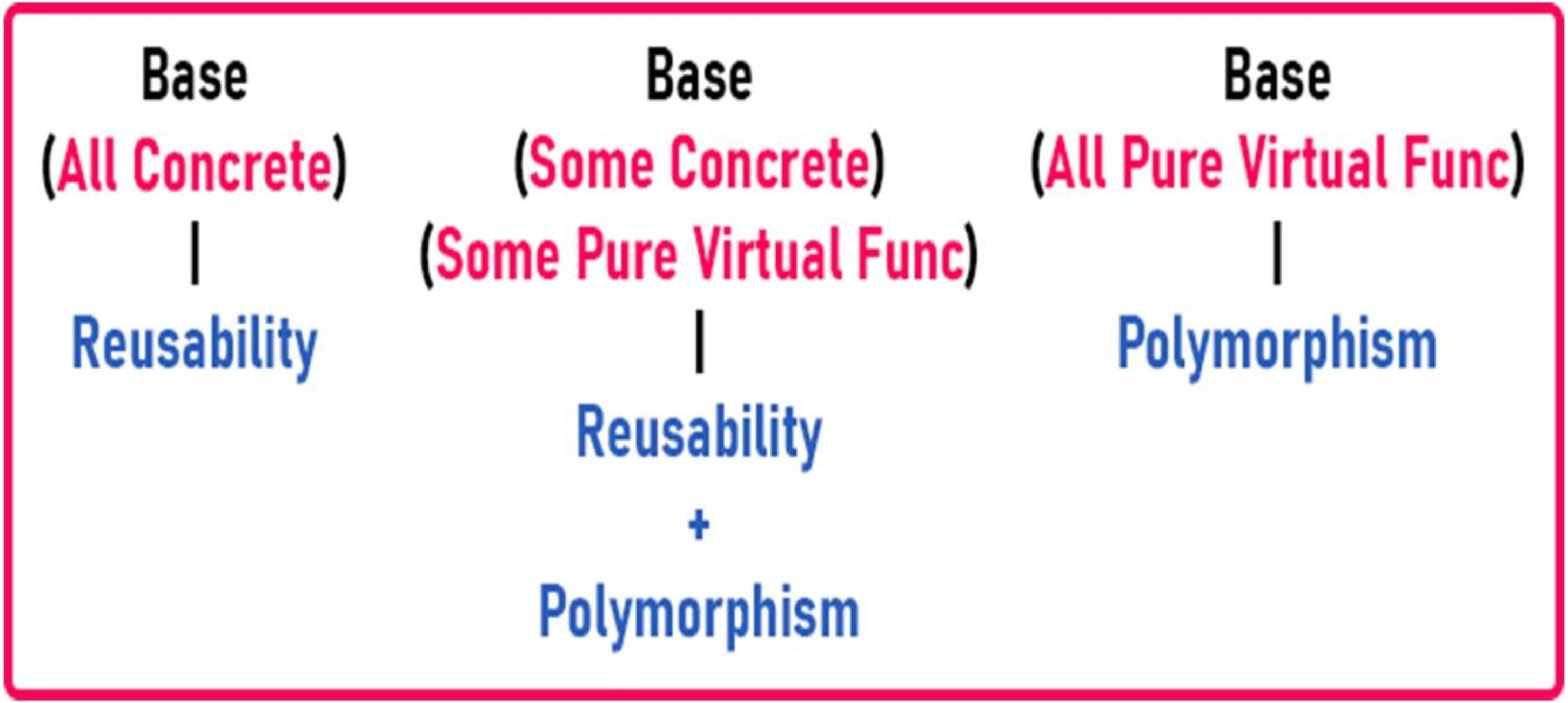
Type of Classes in C++ (1/2)

There are three types of classes that we can write:

1. A base class **with all concrete functions**. What is the purpose of this? **Reusability**.
2. A base class having **some concrete functions** and **some pure virtual functions** then the purpose is **reusability** as well as **polymorphism**.
3. A base class having **all pure virtual functions**, then the only purpose is **polymorphism**. This class is also known as an abstract class or an **interface**.

Type of Classes in C++ (2/2)

7 Abstract Classes



Base
(All Concrete)
|
Reusability

Base
(Some Concrete)
(Some Pure Virtual Func)
|
Reusability
+
Polymorphism

Base
(All Pure Virtual Func)
|
Polymorphism

What is an Interface ?

- An **interface** is an **abstract class** used only to denote a set of functionalities without implementing them; that is, pure virtual functions, which will be defined only in derived classes.

```
class BST {
public:
virtual ~BST() = 0;
virtual void insert(int val) = 0;
virtual bool find(int val) = 0;
virtual void print_inorder() = 0;
 };
BST::~BST() {}
```