

Computer Programming

Dr. Deepak B Phatak
Dr. Supratik Chakraborty
Department of Computer Science and Engineering
IIT Bombay

Session: More on Inheritance

Recap



- Inheritance with public, private and protected members
- Public, private and protected inheritance/derivation
- Access control in derived classes

Overview of This Lecture



- Redefining member functions of the base class
- Access methods of base class using derived classes
- Constructors for derived classes
- Destructors
- Inheritance of assignment operators

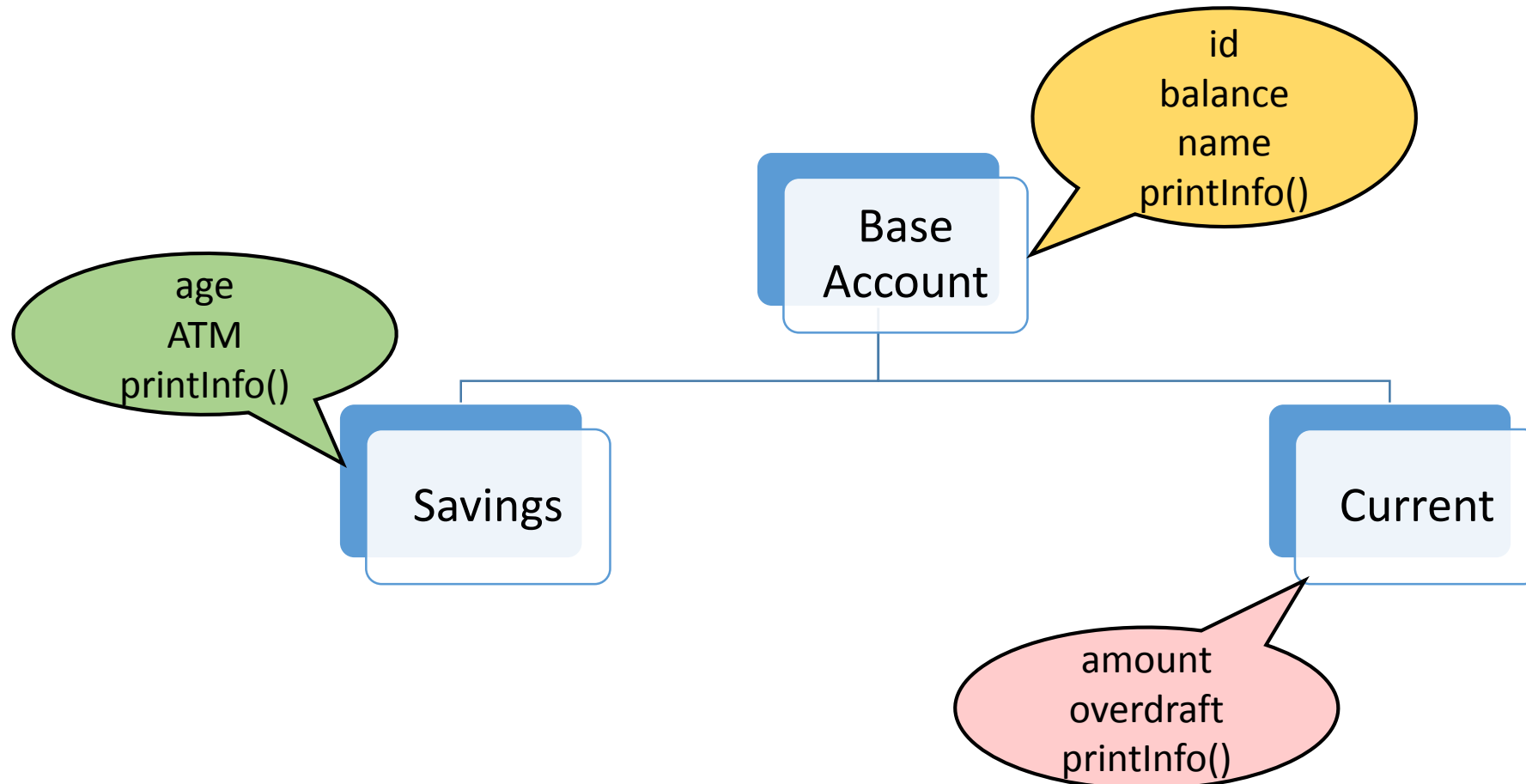
Acknowledgment



- Much of this lecture is motivated by the treatment in **An Introduction to Programming Through C++** by **Abhiram G. Ranade** **McGraw Hill Education 2014**

Some examples used in this lecture are from the above book

Recall: Modified Bank Account hierarchy



Redefining member functions of base class

```
int main() {  
    base ac1; savings ac2; current ac3;  
  
    ac1.id = 1;   ac1.balance = 15000;  
    ac2.id = 2;   ac2.balance = 67890;  
    ac3.id = 3;   ac3.balance = 4500;  
  
    ac2.age = 19; ac2.ATM = 240;  
  
    ac3.amount = 1000; ac3.overdraft = 5300;  
  
    ac1.printInfo();  
    ac2.printInfo();  
    ac3.printInfo();  
  
    return 0;  
}
```

Output of the Program

Printing in base:
1, 15000

Printing in savings:
19, 240

Printing in current:
1000, 5300

Access methods of base class using derived class



```
class base {  
public:  
    int id;  
    float balance;  
    char name[];  
    void printInfo() {  
        cout << "Printing in base: \n" ;  
        cout << id << ", " << balance << endl;  
    }  
};
```

```
class savings : public base {  
public:  
    int age;  
    long int ATM;  
    void printInfo() {  
        base :: printInfo();  
        cout << "\nPrinting in savings: \n";  
        cout << age << ", " << ATM << endl;  
    }  
};
```

Insert

```
class current : public base {  
public:  
    int amount;  
    int overdraft;  
    void printInfo() {  
        base :: printInfo();  
        cout << "\nPrinting in current: \n";  
        cout << amount << ", " << overdraft << endl;  
    }  
};
```

What, if we want to access **printInfo()** of the base class using **derived classes**

Output of the Program	Modified Output
Printing in base: 1, 15000	Printing in base: 1, 15000
Printing in savings: 19, 240	Printing in base: 2, 67890 Printing in savings: 19, 240
Printing in current: 1000, 5300	Printing in base: 3, 4500 Printing in current: 1000, 5300

```
int main() {  
    base ac1; savings ac2; current ac3;  
  
    ac1.id = 1;   ac1.balance = 15000;  
    ac2.id = 2;   ac2.balance = 67890;  
    ac3.id = 3;   ac3.balance = 4500;  
  
    ac2.age = 19; ac2.ATM = 240;  
    ac3.amount = 1000; ac3.overdraft = 5300;  
  
    ac1.printInfo();  
    ac2.printInfo();  
    ac3.printInfo();  
  
    return 0;  
}
```

Constructors for Derived Classes

Case1: (a) With default constructor for base class.
(b) No explicit base constructor invocation in derived class

```
class base {  
    public:  
    int id;  
    float balance;  
    char name[];  
    base(){  
        cout << "Default constructor: base\n" ;  
        id = 0;  
        balance = 0.0;  
    };  
    void printInfo() {  
        cout << "Printing in base: \n" ;  
        cout << id << ", " << balance << "\n";  
    }  
};
```

```
int main() {  
    base ac1;  
    ac1.printInfo();  
    int age = 20;  
    int ATM = 240;  
    savings ac2(age, ATM);  
    ac2.printInfo();  
    return 0;  
}
```

```
class savings : public base {  
    public:  
    int age;  
    long int ATM;  
    savings(int x, int y): age(x), ATM(y){  
        cout << "Derived constructor";  
    }  
    void printInfo() {  
        cout << "\nPrinting in savings: \n" ;  
        cout << age << ", " << ATM << endl ;  
    }  
};
```

Output

```
Default constructor: base  
Printing in base:  
0, 0  
Default constructor: base  
Derived constructor  
Printing in savings:  
20, 240
```


Constructors for Derived Classes



Case2: (a) Without default constructor for base class (parameterised constructor)
(b) No explicit base constructor invocation in derived class

```
class base {  
public:  
    int id;  
    float balance;  
    char name[];  
    base(int a){ // constructor with argument  
        cout << "Constructor: base\n";  
        id = a;  
        balance = 0.0;  
    }  
    void printInfo() {  
        cout << "Printing in base: \n";  
        cout << id << ", " << balance << "\n";  
    }  
};
```

```
int main() {  
    int ATM = 240;  
    int age = 20;  
    savings ac2(age, ATM);  
    return 0;  
}
```

**Compile
error**

**Will this program
compile ?**

```
class savings : public base {  
public:  
    int age;  
    long int ATM;  
    savings(int x, int y): age(x), ATM(y){  
        cout << "Derived constructor ";  
    }  
    void printInfo() {  
        cout << "\nPrinting in savings: \n";  
        cout << age << " < ATM << endl;  
    }  
};
```

**expects base constructor
to be invoked with an
argument**

Constructors for Derived Classes

Case3: (a) No default constructor for base class.

(b) Derived constructor specifying invocation of base constructor

```
class base {  
public:  
    int id;  
    float balance;  
    char name[];  
    base(int a) {  
        cout << "Default constructor: base\n";  
        id = a;  
        balance = 0.0;  
    };  
    void printInfo() {  
        cout << "Printing in base: \n";  
        cout << id << ", " << balance << "\n";  
    }  
};
```

```
int main() {  
    base ac1(1);  
    ac1.printInfo();  
    int id = 10, age = 20;  
    int ATM = 240;  
    savings ac2(id, age, ATM);  
    ac2.printInfo();  
    return 0;  
}
```

```
class savings : public base {  
public:  
    int age;  
    long int ATM;  
    savings(int x, int y, int z):  
        base(x),  
        age(y), ATM(z) {  
        cout << "Derived constructor";  
    }  
    void printInfo() {  
        cout << "\nPrinting in savings: \n";  
        cout << age << ", " << ATM << endl;  
    }  
};
```

Output

```
Default constructor: base  
Printing in base:  
1, 0  
Default constructor: base  
Derived constructor  
Printing in savings:  
20, 240
```

Constructors for Derived Classes

Case4: (a) No default constructor for base class.

(b) Initialize members of derived class: Using body

```
class base {  
public:  
    int id;  
    float balance;  
    char name[];  
    base(int a) {  
        cout << "Default constructor: base\n";  
        id = a;  
        balance = 0.0;  
    };  
    void printInfo() {  
        cout << "Printing in base: \n";  
        cout << id << ", " << balance << "\n";  
    }  
};
```

```
int main() {  
    base ac1(1);  
    ac1.printInfo();  
    int id = 10, age = 20;  
    int ATM = 240;  
    savings ac2(id, age, ATM);  
    ac2.printInfo();  
    return 0;  
}
```

```
class savings : public base {  
public:  
    int age;  
    long int ATM;  
    savings(int x, int y, int z): base(x) {  
        age = y;  
        ATM = z;  
        cout << "Derived constructor";  
    }  
    void printInfo() {  
        cout << "\nPrinting in savings: \n";  
        cout << age << ", " << ATM << endl;  
    }  
};
```

Output

Default constructor: base

Printing in base:

1, 0

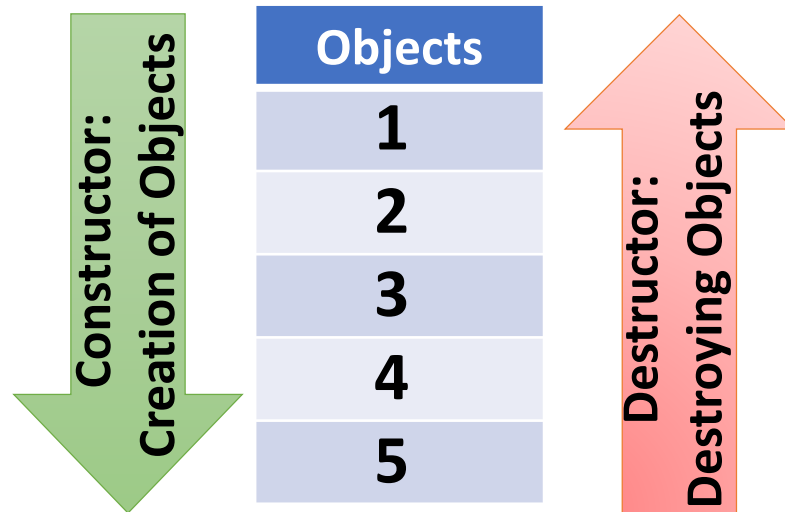
Default constructor: base

Derived constructor

Printing in savings:

20, 240

Destructors



```
class A {  
    public:
```

```
    A () { // Constructor  
        ...  
    }
```

```
    ~A() {  
        ...  
    }
```

```
}
```

Destructors

```
class base {  
public:  
    base(){ 2  
        cout << "Constructor: base\n";  
    }  
  
    ~base(){ 5  
        cout << "Destructor: base\n";  
    }  
};
```

```
int main() {  
    savings s; 1  
    return 0;  
}
```

```
class savings : public base {  
public:  
    savings(){ 3  
        cout << "Constructor: savings\n";  
    }  
  
    ~savings(){ 4  
        cout << "Destructor: savings\n";  
    }  
};
```

Output

```
Constructor: base  
Constructor: savings  
Destructor: savings  
Destructor: base
```

Inheritance of assignment operators

```
class base {  
    public:  
        int id;  
        base(int x):id(x){ } base constructor  
        base & operator=(base &a){  
            id = a.id; assignment operator  
            cout << "base class operator\n" ;  
            return *this;  
        }  
};
```

```
class savings : public base {  
    public:  
        int age;  
        savings(int x, int y):base(x),age(y) { } savings constructor  
};
```

```
int main() {  
    base b1(10);  
    savings s1(11,20) s2(12,30);  
    b1 = s1; ✓  
    s2 = b1; ✗  
    return 0;  
}
```

b1.operator=(s1);

s2.operator=(b1): assignment operator is not inherited

Summary



- Redefining member functions of the base class
- Access methods of base class using derived classes
- Constructors for derived classes
- Destructors
- Inheritance of assignment operators