IIT Bombay

# Computer Programming

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

## Session: Polymorphism and Virtual Functions

# Recap

- Objects of base and derived classes
- Objects of classes with pointers and references
- Inheritance
  - Multiple
  - Diamond

# Overview of This Lecture

- Recapitulating ‘printInfo’ of base and derived classes
- Polymorphism
- Virtual destructor
- Abstract class

# What is Polymorphism?

## Dictionary Meaning

The condition of **occurring** in several **different forms**

or

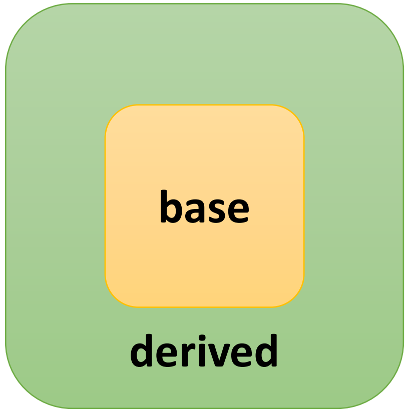
The ability to **assume different forms** or **shapes**.

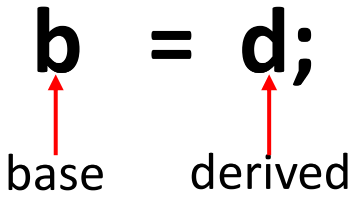
## Computer Science

**Greek:** **polys** → **many, much**

**morphē** → **form, shape**

# Already seen in some forms





```
b = d;
```

**Object ‘d’ being an object of derived class,
can also be viewed as an object of base class
(has all members of the base class)**

**Thus, object ‘d’ can be viewed as having multiple ‘forms’**

# Examining printInfo() from savings and current

IIT Bombay

```
class base {
  public:
    int id; float balance;
    void printInfo() {
      cout << "base\n";
    }
};
```

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

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

```
int main() {
  base b; savings s; current c;

  base *bptr;
  bptr = &s;
  bptr->printInfo();

  bptr = &c;
  bptr->printInfo();
  return 0;
}
```

**address of 's' assigned to base pointer**

**address of 'c' assigned to base pointer**

**Output**

**base**

**base**

**How to print info from 'savings' and 'current' by invoking bptr->printInfo()?**

# How do we solve ?

**We want ‘bptr->printInfo();’ to behave as**
**(1) printInfo() in ‘savings’ after ‘bptr = &s;’**
**(2) printInfo() in ‘current’ after ‘bptr = &c;’**

**Solution: Virtual functions**

**Polymorphism**

# Polymorphism

```
class base {
  public:
    int id; float balance;

    virtual void printInfo() {
      cout << "base\n";
    }
};
```

```
class savings : public base {
  public:
    int age; long int ATM;

    void printInfo() {
      cout << "savings \n";
    }
};
```

```
class current : public base {
  public:
    int amount, overdraft;

    void printInfo() {
      cout << "current \n";
    }
};
```

```
int main() {
  base b; savings s;
  current c;

  base * bptr = &s;

  bptr->printInfo();

  bptr = &c;

  bptr->printInfo();

  return 0;
}
```

**Assigning addr of 'savings' object to 'base' pointer**

**print info from the 'savings' object**

**Assigning addr of ‘current' object to 'base' pointer**

**print info from the ‘current' object**

**Output**

**savings**

**current**

# Polymorphism

```
class base {
  public:
    int id; float balance;
    void call() { cout << "base call\n";  }
    virtual void printInfo() {
      cout << "base\n";
    }
};
```

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

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

```
int main() {
  base b; savings s;
  current c;

  base * bptr = &s;
  bptr->call();
  bptr->printInfo();

  bptr = &c;
  bptr->call();
  bptr->printInfo();

  return 0;
}
```

**Output**

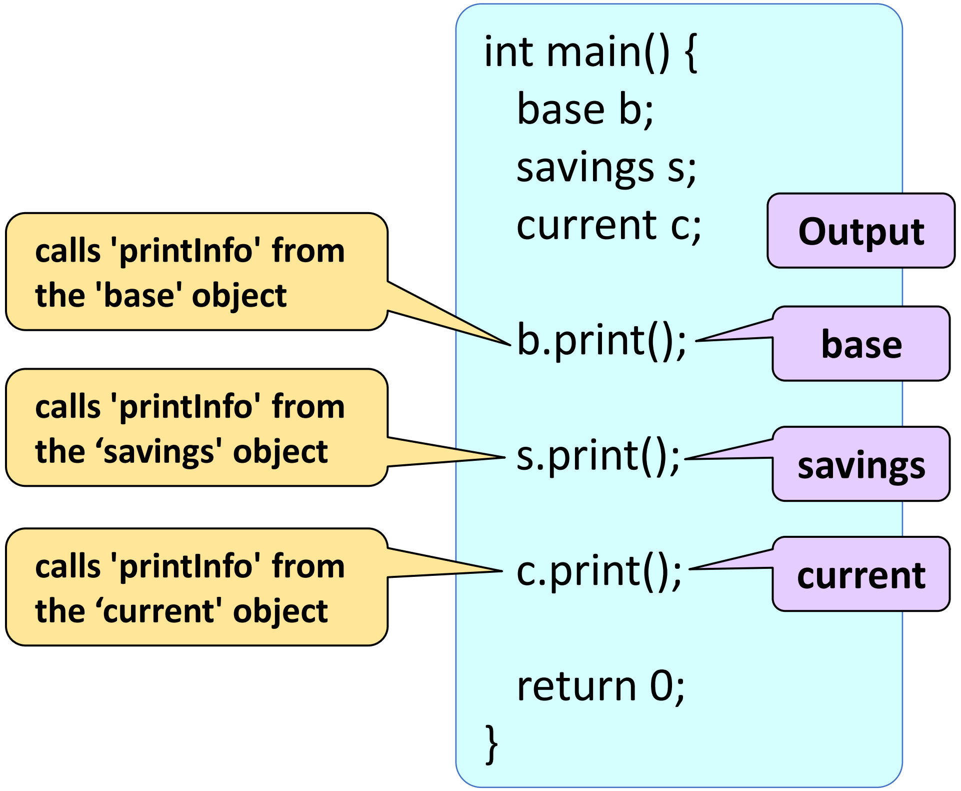
```
base call
savings
base call
current
```

# Polymorphism: A different variant

```
class base {
  public:
    int id;
    float balance;
    void print() { printInfo();}
    virtual void printInfo() {
      cout << "base\n";
    }
};
```

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

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



```
int main() {
  base b;
  savings s;
  current c;

  b.print();
  s.print();
  c.print();

  return 0;
}
```

**calls 'printInfo' from the 'base' object**

**calls 'printInfo' from the ‘savings' object**

**calls 'printInfo' from the ‘current' object**

**Output**

**base**

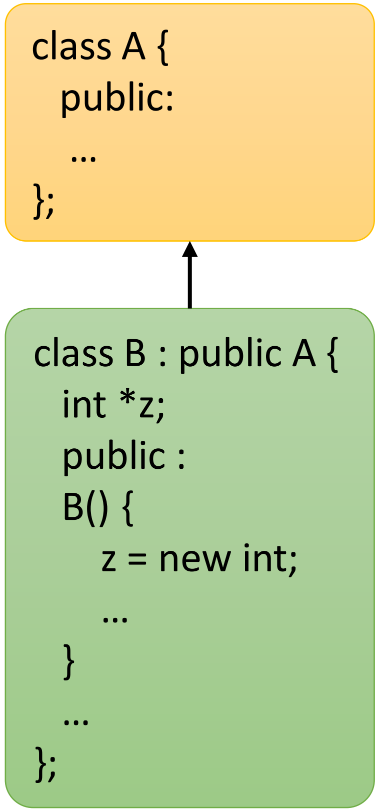
**savings**

**current**

# Virtual Destructor

## Problem Overview:

- 2 classes, ‘class A’ and ‘class B’.
- ‘B’ inherits from ‘A’.
- ‘aptr’ is of type ‘A*’
- Object pointed by ‘aptr’ is of type ‘B’
- Private data member ‘z’ of class ‘B’



```
class A {
  public:
   …
};
```

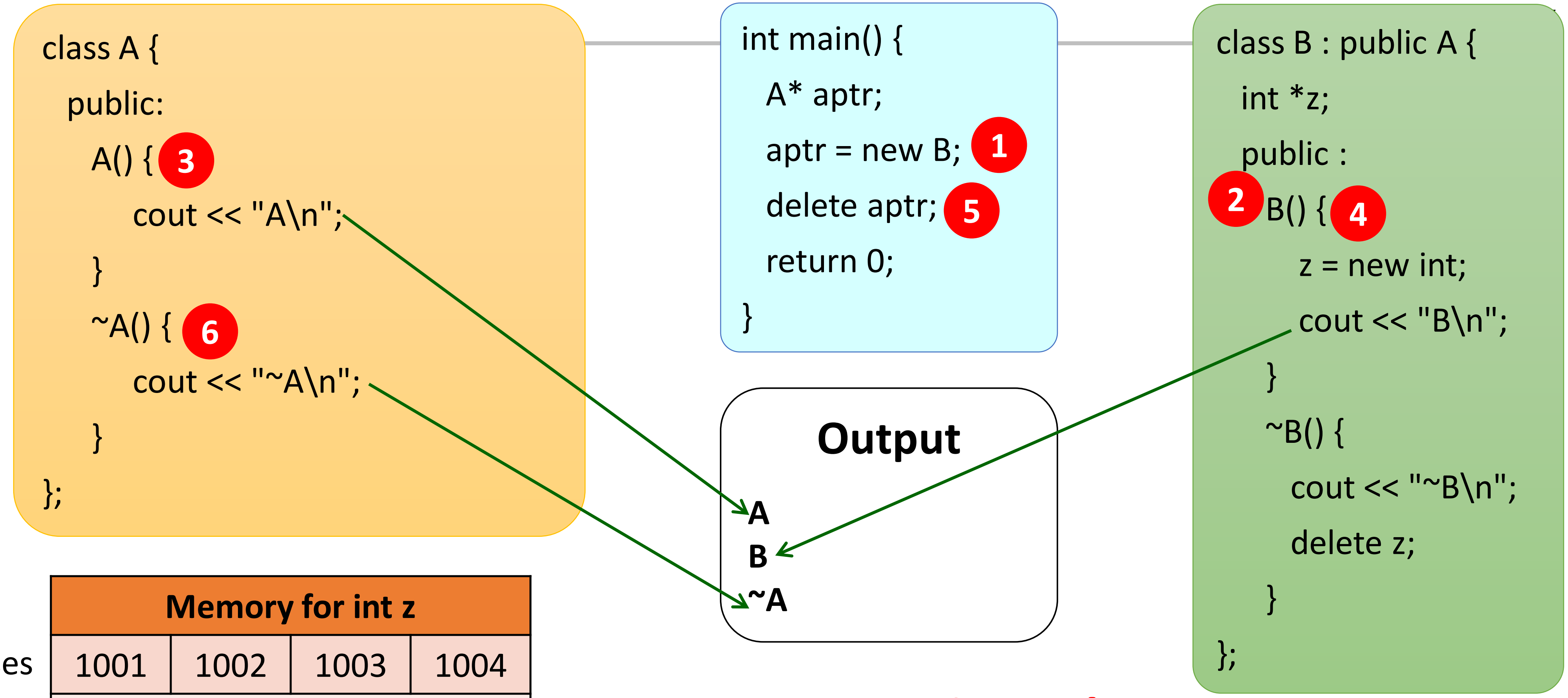
```
class B : public A {
  int *z;
  public :
  B() {
     z = new int;
     …
  }
  …
};
```

```
int main() {
   A* aptr;
   aptr = new B;
   …
}
```

## Problem Definition:

- How to **delete resources/memory** occupied by the **derived class using the ‘base’** class pointer ?

# Motivation: Virtual Destructor



```
class A {
  public:
    A() {
      cout << "A\n";
    }
    ~A() {
      cout << "~A\n";
    }
};
```

```
int main() {
  A* aptr;
  aptr = new B;
  delete aptr;
  return 0;
}
```

```
class B : public A {
  int *z;
  public :
   B() {
     z = new int;
     cout << "B\n";
   }
   ~B() {
     cout << "~B\n";
     delete z;
   }
};
```

**Output**

```
A
B
~A
```

|  | Memory for int z | | | |
|---|---|---|---|---|
| Addresses | 1001 | 1002 | 1003 | 1004 |
|  | Value of int *z | | | |

**Program terminated**

**Base destructor not called**

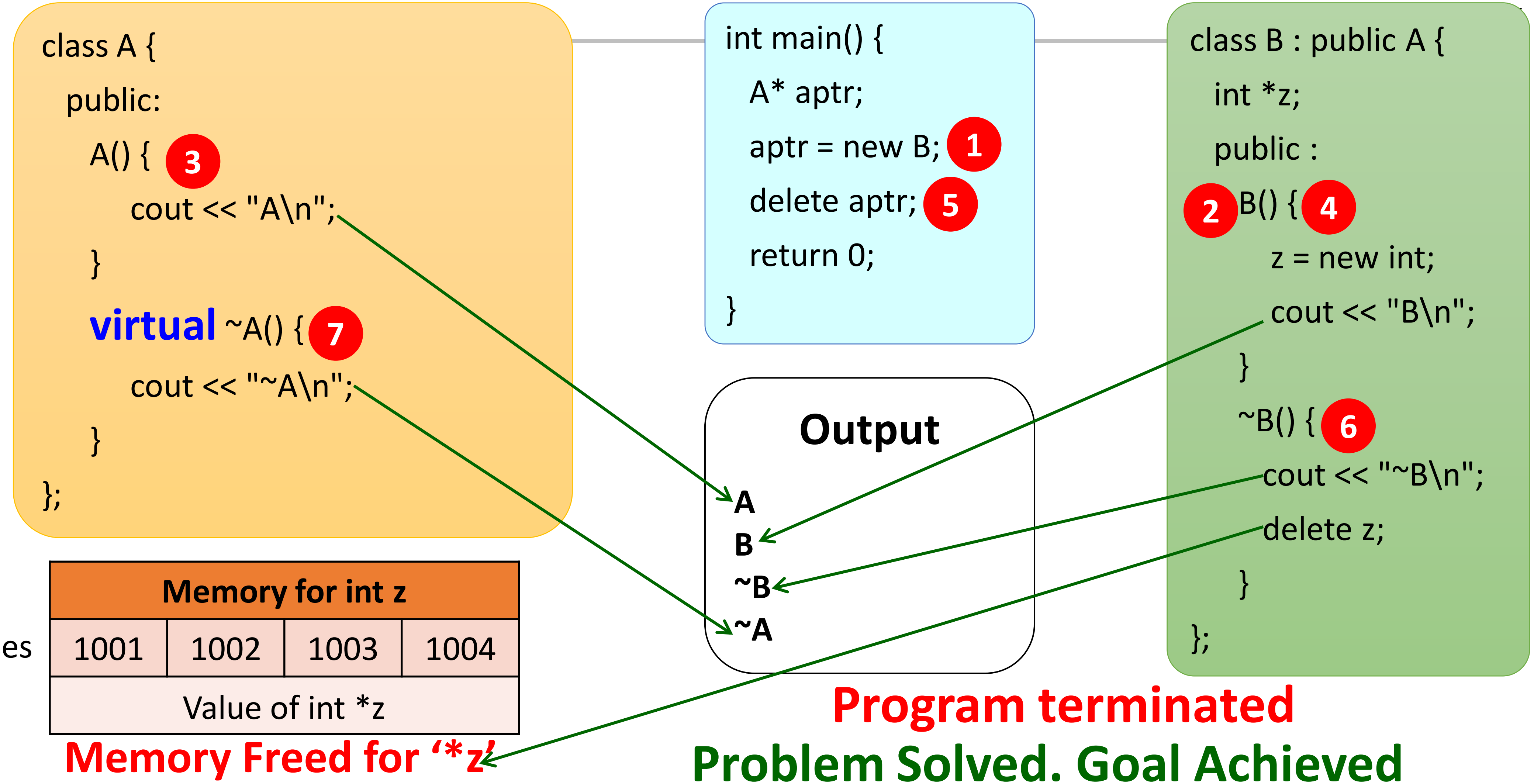
**Memory for ‘z’ not freed. Hence, problem NOT solved**

# Proposed solution: Virtual destructor

**To enforce that destructor ‘B’ is called:**

**Sol: Declare destructor of ‘A’ as virtual**

# Virtual Destructor



```
class A {
  public:
    A() { 3
      cout << "A\n";
    }
    virtual ~A() { 7
      cout << "~A\n";
    }
};
```

```
int main() {
  A* aptr;
  aptr = new B; 1
  delete aptr; 5
  return 0;
}
```

```
class B : public A {
  int *z;
  public :
2   B() { 4
      z = new int;
      cout << "B\n";
    }
    ~B() { 6
      cout << "~B\n";
      delete z;
    }
};
```

**Output**

```
A
B
~B
~A
```

| | Memory for int z | | | |
|---|---|---|---|---|
| Addresses | 1001 | 1002 | 1003 | 1004 |
| | Value of int *z | | | |

**Memory Freed for '*z'**

**Program terminated**

**Problem Solved. Goal Achieved**

# Abstract class

## Abstract class is:

- A class that cannot be instantiated directly
- Implemented as a class that has one or more **pure virtual functions**
    - Which should be overridden by member function definitions of derived class

## When should we use it

- When using the base class directly has no meaningful purpose
- i.e. It makes sense to use it only as a derived class

## Example (Bank account – already examined)

- A person does not have **just a bank account**.
- It is either a **savings bank account** or a **current bank account**
- Instantiating class ‘base’ by itself has no meaningful purpose

# Abstract class: Example 1

```
class base {
   public:
      int id; float balance;
      virtual void call() = 0;
      virtual void printInfo() = 0;
};
```

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

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

```
int main() {
   //base B;
   base *b;
   savings s;

   b = &s;
   b->call();
   b->printInfo();

   current c;
   b = &c;
   b->call();
   b->printInfo();

   return 0;
}
```

Compile Error

Cannot declare variable ‘B’ to be of abstract class type ‘base’

assigning 'savings' object to 'base' pointer

print info from the 'savings' object

assigning ‘current' object to 'base' pointer

print info from the ‘current' object

Output

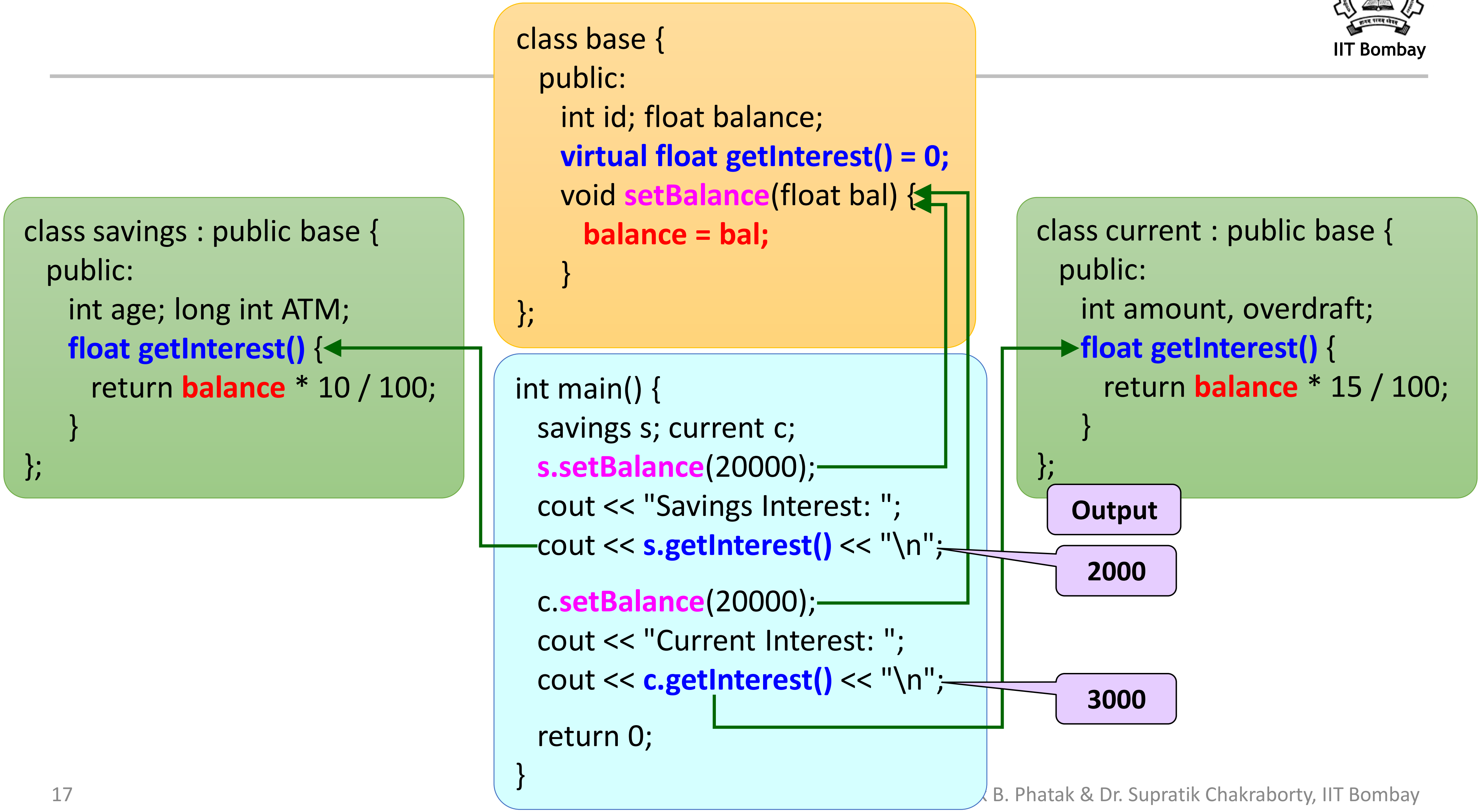
savings call

savings

current call

current

# Abstract class: Example 2



```
class base {
  public:
    int id; float balance;
    virtual float getInterest() = 0;
    void setBalance(float bal) {
      balance = bal;
    }
};
```

```
class savings : public base {
  public:
    int age; long int ATM;
    float getInterest() {
      return balance * 10 / 100;
    }
};
```

```
class current : public base {
  public:
    int amount, overdraft;
    float getInterest() {
      return balance * 15 / 100;
    }
};
```

```
int main() {
  savings s; current c;
  s.setBalance(20000);
  cout << "Savings Interest: ";
  cout << s.getInterest() << "\n";

  c.setBalance(20000);
  cout << "Current Interest: ";
  cout << c.getInterest() << "\n";

  return 0;
}
```

**Output**

**2000**

**3000**

# Abstract class

- Used when base class is only meant for derivation
- Helps in readability and understanding
- Prevents accidental instantiation of abstract class

**Caveat**: You cannot instantiate objects of this class

# Summary

- Polymorphism in C++ programming
- Virtual destructor
- Abstract class