OntarioTech
Science

Kourosh Davoudi
kourosh@uoit.ca

Week 1: Virtual Functions
Abstract Classes
Advanced Topics

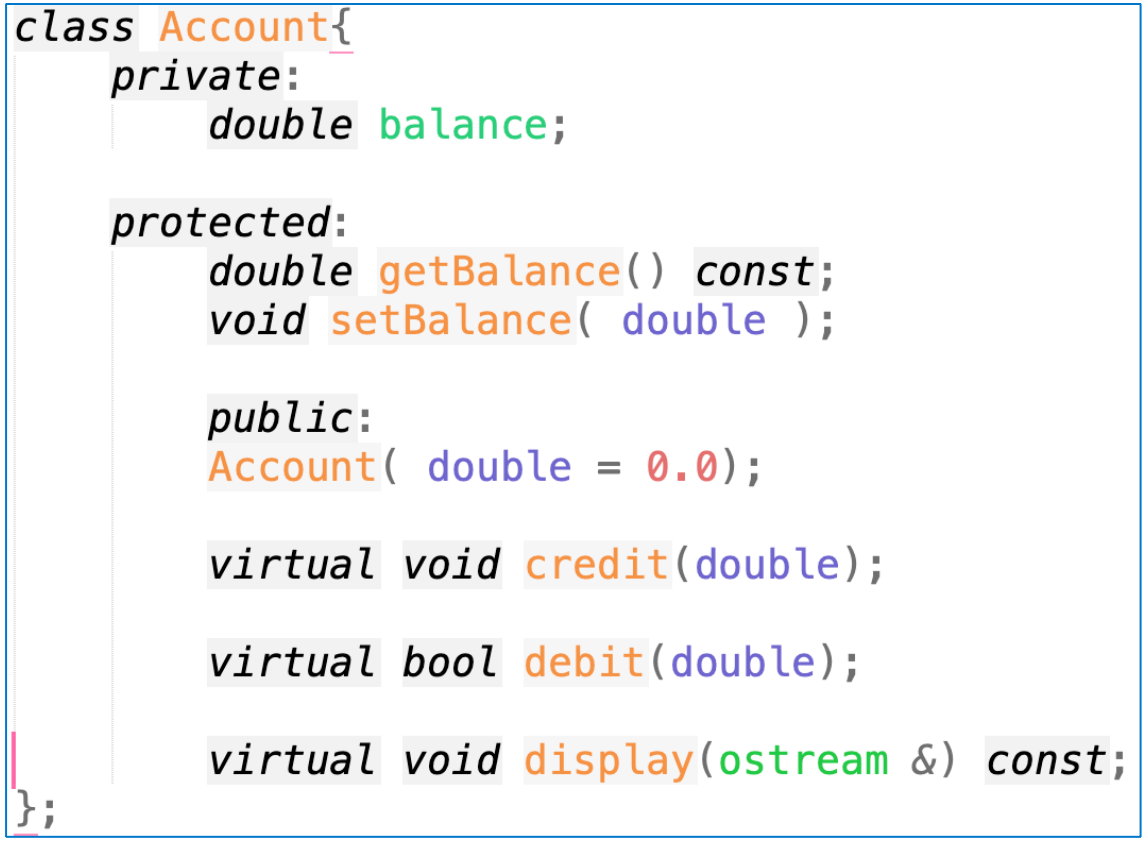
CSCI 1061: Programming Workshop II

Learning Outcomes

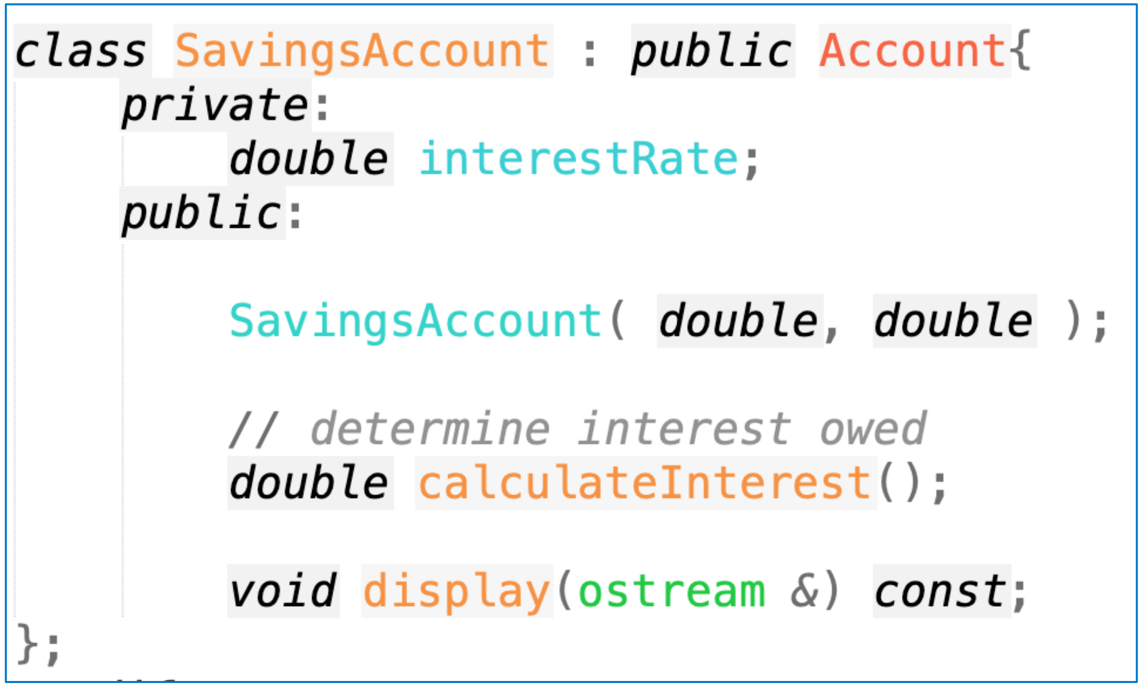
In this week, we learn:

- How to use Inheritance using a case-study:
  - `Account` class
- Abstract class and pure virtual function
- More advanced topics:
  - Overload assignment in the derived class
  - Overloading vs overriding vs shadowing

Account/SavingsAccount

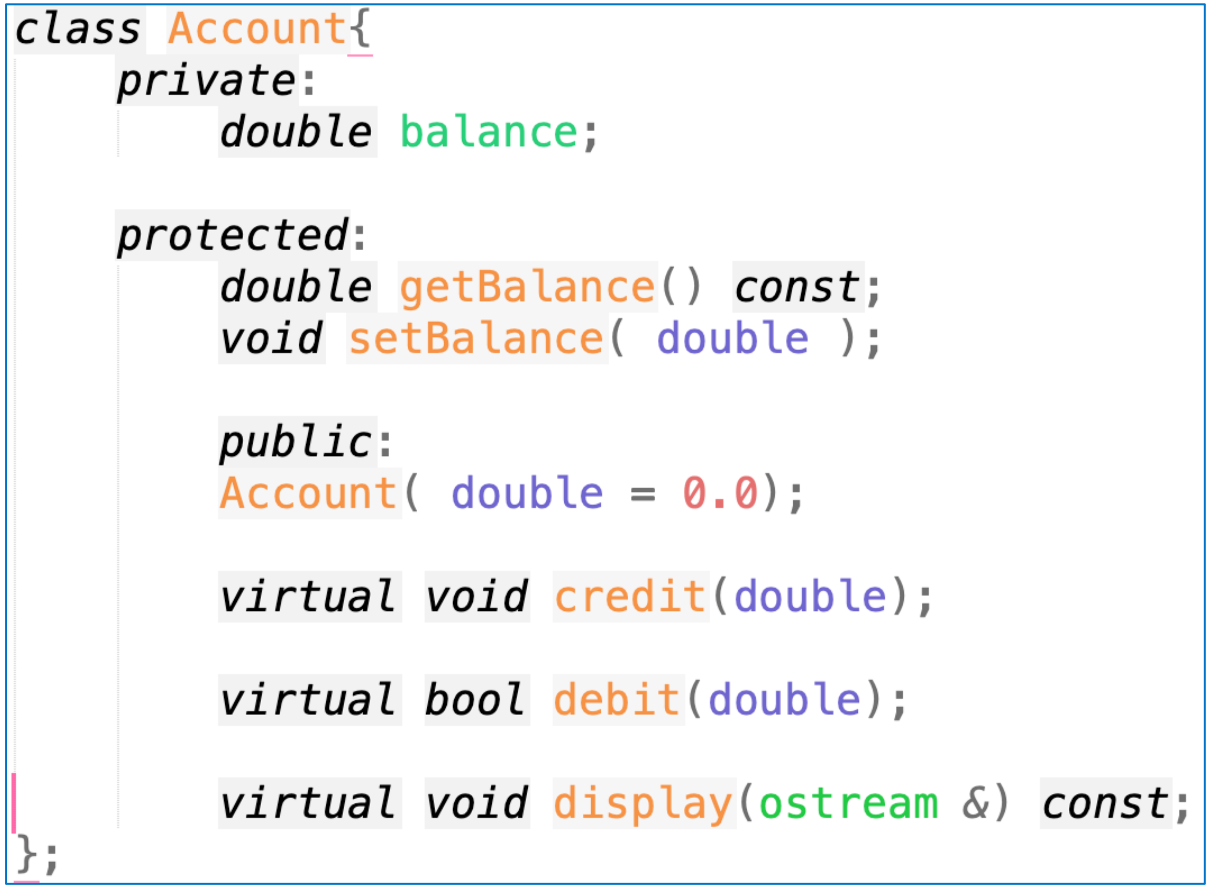


```
class Account{
    private:
        double balance;

    protected:
        double getBalance() const;
        void setBalance( double );

        public:
        Account( double = 0.0);

        virtual void credit(double);

        virtual bool debit(double);

        virtual void display(ostream &) const;
};
```

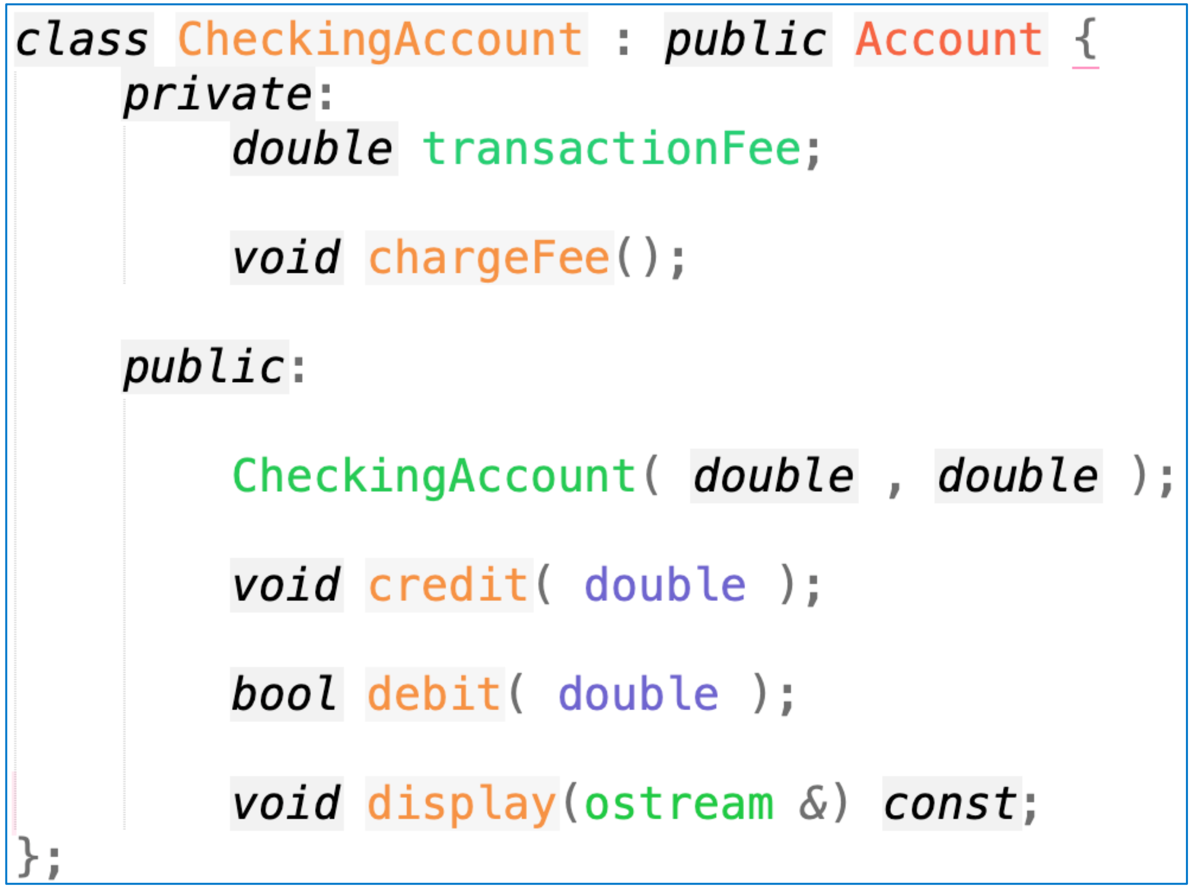


```
class SavingsAccount : public Account{
    private:
        double interestRate;
    public:

        SavingsAccount( double, double );

        // determine interest owed
        double calculateInterest();

        void display(ostream &) const;
};
```

Account/CheckingAccount



```
class Account{
    private:
        double balance;

    protected:
        double getBalance() const;
        void setBalance( double );

        public:
        Account( double = 0.0);

        virtual void credit(double);

        virtual bool debit(double);

        virtual void display(ostream &) const;
};
```



```
class CheckingAccount : public Account {
    private:
        double transactionFee;

        void chargeFee();

    public:

        CheckingAccount( double , double );

        void credit( double );

        bool debit( double );

        void display(ostream &) const;
};
```

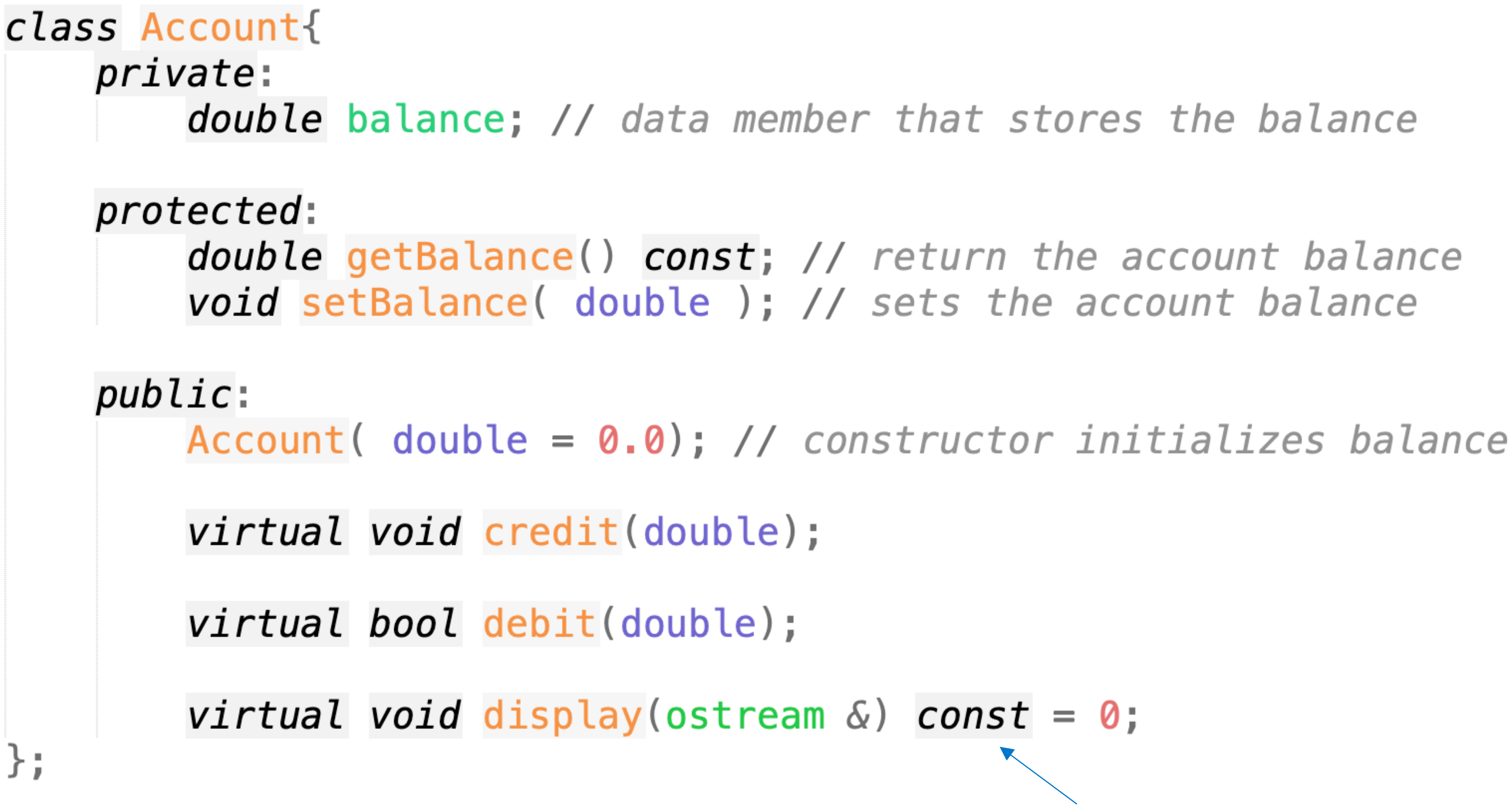
Abstract Class

- In the Account class we, don’t have a proper definition for the display function.
  - The reason is that this function is general to be implemented
- Solution
  - We can define this function as pure virtual
  - By doing this, we don’t need to implement display function in Account

A class which has a pure virtual function is an abstract class

Note: we cannot create an object of the abstract class

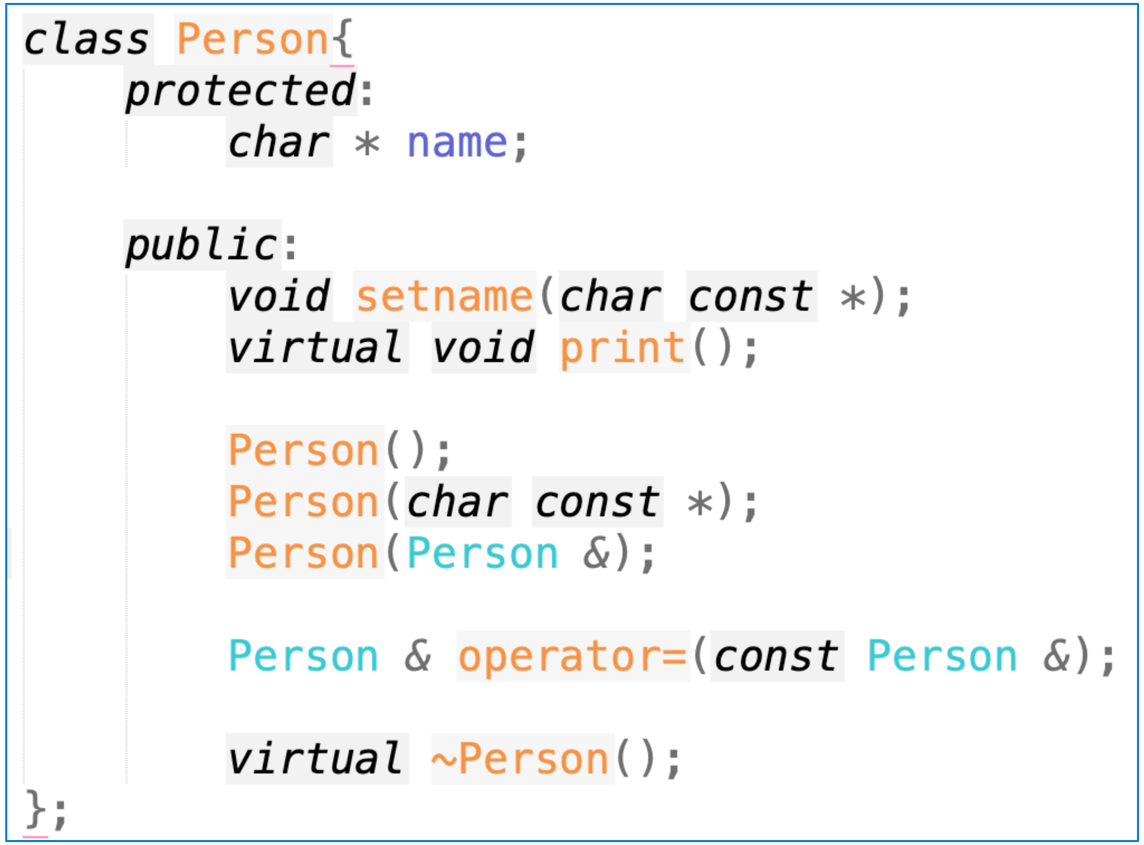
Abstract Class



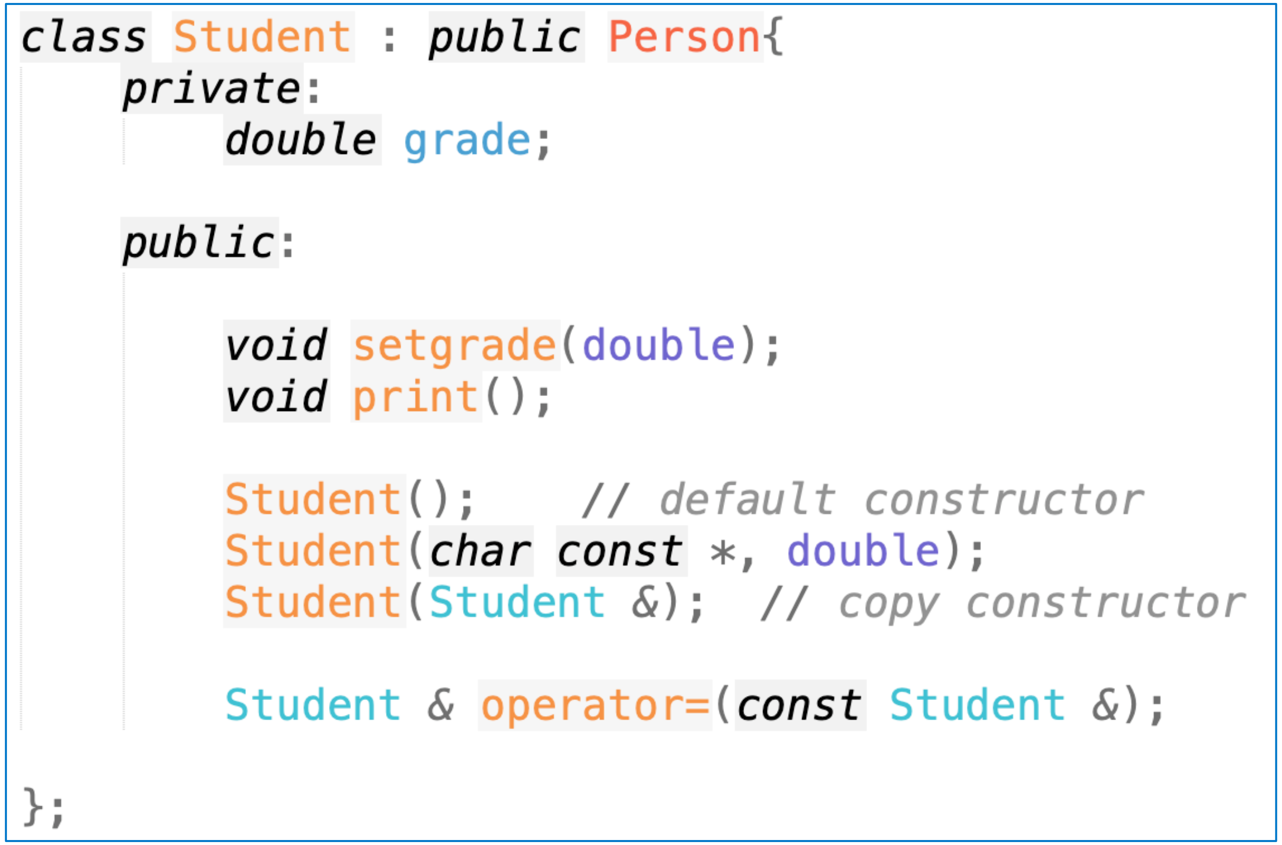
```
class Account{
    private:
        double balance; // data member that stores the balance

    protected:
        double getBalance() const; // return the account balance
        void setBalance( double ); // sets the account balance

    public:
        Account( double = 0.0); // constructor initializes balance

        virtual void credit(double);

        virtual bool debit(double);

        virtual void display(ostream &) const = 0;
};
```

Pure virtual
Function

Overloading operator= in Derived class



```
class Person{
    protected:
        char * name;

    public:
        void setname(char const *);
        virtual void print();

        Person();
        Person(char const *);
        Person(Person &);

        Person & operator=(const Person &);

        virtual ~Person();
};
```



```
class Student : public Person{
    private:
        double grade;

    public:

        void setgrade(double);
        void print();

        Student();    // default constructor
        Student(char const *, double);
        Student(Student &);  // copy constructor

        Student & operator=(const Student &);

};
```

Special Topics

- Polymorphism
  - Overriding: defining a virtual function of the base class in the derived class

```
class Person{

    …
  public:
    virtual void print();

}
```

```
class Student: public Person{

    …
  public:
    void print();

}
```

Same signature (i.e., name and parameter)

Special Topics

- Polymorphism
  - overloading: defining a function in the base class in the derived class with the same name and different parameter nnumber/types

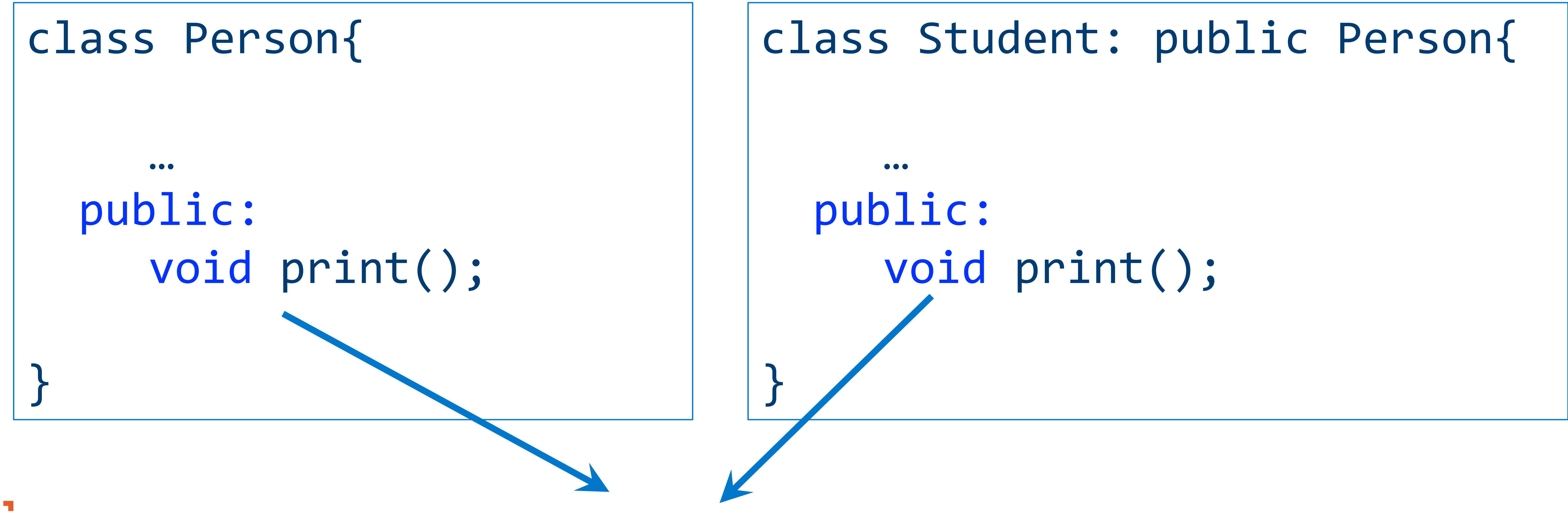
```
class Person{

     …
  public:
     void print();

}
```

```
class Student: public Person{

     …
  public:
     void print(int);

}
```

Different signature (i.e., name and parameter). Student class has both prints

Special Topics

- Shadowing/ redefining: defining a non virtual function of the base class in the derived class



```
class Person{

     …
  public:
     void print();

}
```

```
class Student: public Person{

     …
  public:
     void print();

}
```

Same signature (i.e., name and parameter)

Special Topics

- const keyword:

```
class Person{

      …
  public:
      void print() const;

}
```

It means that print cannot change any member of Person class

Special Topics

- const keyword:

It means that this function cannot change the parameter

```
class Person{

        …
   public:
        Person & operator=(const Person &);

}
```

Special Topics

- A function which returns a reference
  - This kinds of function can be use in the left hand side of assignment !
  - The should return a variable that are alive (lifetime)

```
int & doubleValue(int x)
{
    int value = x * 2 ;
    return value; // return a reference to value here
}                 // value is destroyed here !
```

🤨

Special Topics

```
// Returns a reference to the index element of array
int & getElement(int array[], int index)
{
    // we know that array[index] will not be destroyed when we return to the caller
    return array[index];
}

int main()
{
     int array[100];

     getElement(array, 10) = 5;     // Set the element of array with index 10

     cout << array[10] << endl;

     return 0;
}
```

Special Topics

- String in C (c-style string): in C, we don’t have a built-in string type !
  - We use array of char and \0 (null) to store our string

```
char a[] = {‘H’, ‘e’, ‘l’, ‘l’, ‘o’, ‘\0’};
char b[] = “Hello”;
```

- Useful function are available in #include<cstring>
  - Examples: strcpy, strcat

Special Topics

Initialization list:

```
class Point {
   private:
      int x;
      int y;
   public:
      Point(int  = 0, int  = 0);

      int getX() const {return x;}
      int getY() const {return y;}
};
Point::Point(int i , int j ):x(i), y(j)
{
 }
```

OK

Initialization List

```
class Point {
   private:
      int x;
      int y;
   public:
      Point(int  = 0, int  = 0);

      int getX() const {return x;}
      int getY() const {return y;}
};
Point::Point(int i , int j )
{
   x = i;
   y = j;
}
```

OK

Special Topics

Initialization list:

```
class Point {
   private:
      const int x;
      const int y;
   public:
      Point(int  = 0, int  = 0);

      int getX() const {return x;}
      int getY() const {return y;}
};
Point::Point(int i , int j ):x(i), y(j)
{
 }
```

OK

Initialization List

```
class Point {
   private:
      const int x;
      const int y;
   public:
      Point(int  = 0, int  = 0);

      int getX() const {return x;}
      int getY() const {return y;}
};
Point::Point(int i , int j ):x(i), y(j)
{
   x = i;
   y = j;
}
```

NOT OK

Special Topics

- In C++, this problem is solved by defining a `string` class

- Useful function are available in #include<string>

Read:

https://web.stanford.edu/class/archive/cs/cs106b/cs106b.1132/handouts/08-C++-Strings.pdf

Write the Student class with `name` as a `string`