MA122 - Computer Programming and Applications

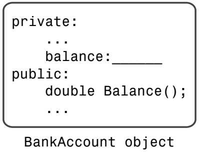
Indian Institute of Space Science and Technology

April 12, 2017

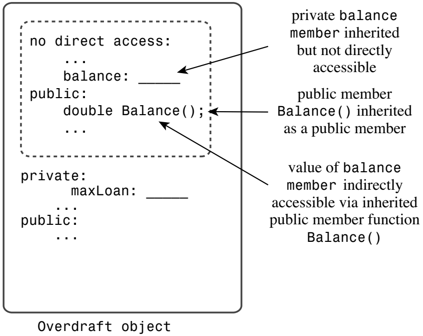
Lecture 29

Class Inheritance

1. When one class inherits from another, the original class is called a **base** class, and the inheriting class is called a **derived** class.



class Overdraft : public BankAccount {...};


no direct access:
...
balance: _____
public:
double Balance();
...
private:
maxLoan: _____
...
public:
...
Overdraft object
private balance member inherited but not directly accessible
public member Balance() inherited as a public member
value of balance member indirectly accessible via inherited public member function Balance()

Base Class

```
#include<iostream>
#include <string>
using std::string;
class TableTennisPlayer
  {
  private:
    string firstname;
    string lastname;
    bool hasTable;
  public:
    TableTennisPlayer (const string & fn = "none",
                     const string & ln = "none", bool ht =
                            false);
    void Name() const;
    bool HasTable() const { return hasTable; };
    void ResetTable(bool v) { hasTable = v; };
  };
```

Base Class

```
TableTennisPlayer::TableTennisPlayer (const string &
    fn, const string & ln, bool ht) : firstname(fn),
    lastname(ln),hasTable(ht) {}

void TableTennisPlayer::Name() const
{
  std::cout << lastname << ", " << firstname;
}
int main ( void )
{
  using std::cout;
```

Base Class

```
  TableTennisPlayer player1("Chuck", "Blizzard", true)
      ;
  TableTennisPlayer player2("Tara", "Boomdea", false);
  player1.Name();
  if (player1.HasTable())
    cout << ": has a table.\n";
  else
    cout << ": hasn't a table.\n";
  player2.Name();
  if (player2.HasTable())
    cout << ": has a table";
  else
    cout << ": hasn't a table.\n";
  return 0;
}
```

Lecture 29

Derived Class

```
#include<iostream>
#include <string>
using std::string;
class TableTennisPlayer
  {
  private:
    string firstname;
    string lastname;
    bool hasTable;
  public:
    TableTennisPlayer (const string & fn = "none",
                  const string & ln = "none", bool ht =
                      false);
    void Name() const;
    bool HasTable() const { return hasTable; };
    void ResetTable(bool v) { hasTable = v; };
  };
```

Derived Class

```
class RatedPlayer : public TableTennisPlayer
  {
  private:
    unsigned int rating;
  public:
    RatedPlayer (unsigned int r = 0, const string & fn
         = "none",
           const string & ln = "none", bool ht = false)
               ;

    RatedPlayer(unsigned int r, const
        TableTennisPlayer & tp);

    unsigned int Rating() const { return rating; }

    void ResetRating (unsigned int r) {rating = r;}
  };
```

Derived Class

```
TableTennisPlayer::TableTennisPlayer (const string &
    fn, const string & ln, bool ht) : firstname(fn),
    lastname(ln),hasTable(ht) {}


void TableTennisPlayer::Name() const
{
  std::cout << lastname << ", " << firstname;
}
```

Derived Class

```cpp
// RatedPlayer methods
RatedPlayer::RatedPlayer(unsigned int r, const string
    & fn,
            const string & ln, bool ht) :
                TableTennisPlayer(fn, ln, ht)
{
  rating = r;
}


RatedPlayer::RatedPlayer(unsigned int r, const
    TableTennisPlayer & tp)
: TableTennisPlayer(tp), rating(r)
{
}
//default copy constructor
//TavleTennisPlayer(const TableTennisPlayer &)
```

Derived Class

```cpp
int main ( void )
{
  using std::cout;
  using std::endl;
  TableTennisPlayer player1("Tara", "Boomdea", false);
  RatedPlayer rplayer1(1140, "Mallory", "Duck", true);
  rplayer1.Name();        // derived object uses base
      method
  if (rplayer1.HasTable())
    cout << ": has a table.\n";
  else
    cout << ": hasn't a table.\n";
```

Derived Class

```
  player1.Name();         // base object uses base
      method
  if (player1.HasTable())
    cout << ": has a table";
  else
    cout << ": hasn't a table.\n";
  cout << "Name: ";
  rplayer1.Name();
  cout << "; Rating: " << rplayer1.Rating() << endl;
  // initialize RatedPlayer using TableTennisPlayer
      object
  RatedPlayer rplayer2(1212, player1);
  cout << "Name: ";
  rplayer2.Name();
  cout<< "; Rating: " << rplayer2.Rating() << endl;
  return 0;

}
```