

Introduction to C, C++, and Unix/Linux

CS 60

Lecture 16: Classes

Today

- C++ Classes
- Reading [KR] Chapters 1-7
- Read [So] chapters 1, 3, 4 (Boolean), 9, 13, 14 & 18 & 10

From namespace to class

- A C++ class combines data and functions in a single namespace, creating a new data type
- Combines features of a struct with a namespace, along with some extra options
 - Data can be declared public, protected, private
 - Some functions automatically generated (e.g., constructor, destructor, copy constructor, **=**, **new**, **delete**)
 - **this**

Classes and objects

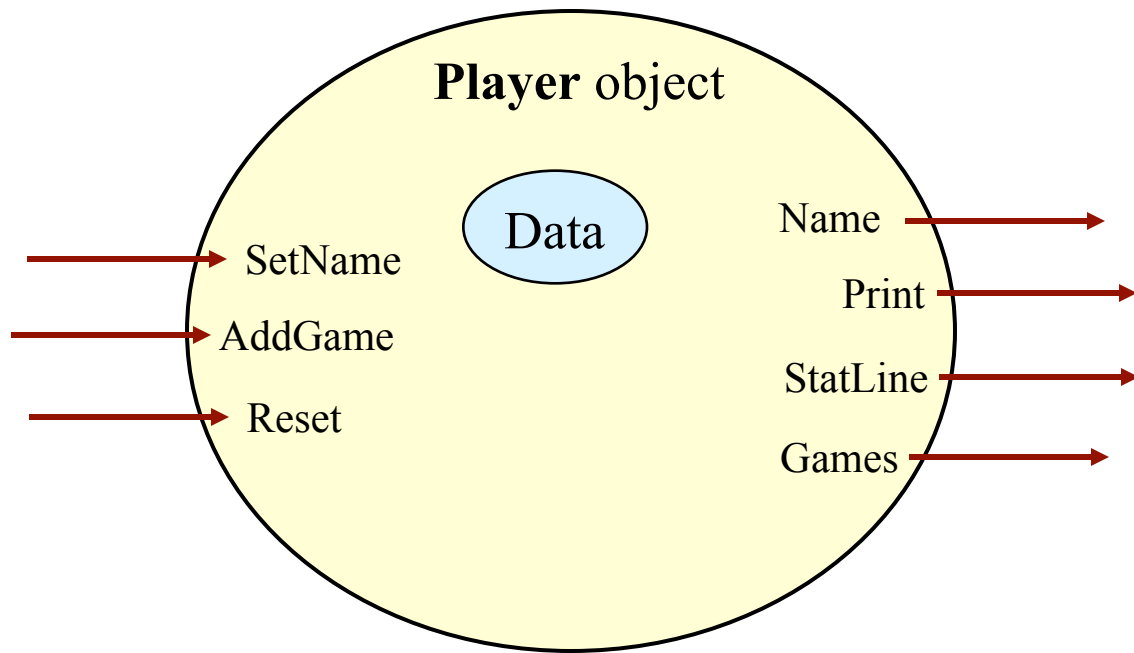
- A C++ class is an object type
 - Defining a class means defining the attributes (data) and behavior (methods/functions) of a new data type
- An object is created by declaring a variable of the class type (instantiation)

```
class Player {                                Player p1, p2, p3;  
  
    ...  
};                                           Class instantiation
```

Class definition/specification
(Often in include file)

Class interface

- The interface defines the behavior of the class to the *outside world* (to other classes and functions that may access variables of your class type).
- The *interface* to a class is the list of public data members and methods
- The implementation of your class doesn't matter outside the class – only the interface
 - The implementation can change dramatically, as long as the interface stays the same



The class user “sees” the interface, not the internal (private) data (directly) and functions

Object oriented programming

- In OOP, the programmer *thinks about* and *defines* the attributes and behavior of objects
 - Often the objects are modeled after real-world entities
- Very different approach than *function-based* programming (like C)
 - Most of the action happens inside classes!
 - Though we still provide **main ()**, and many other things don't change...

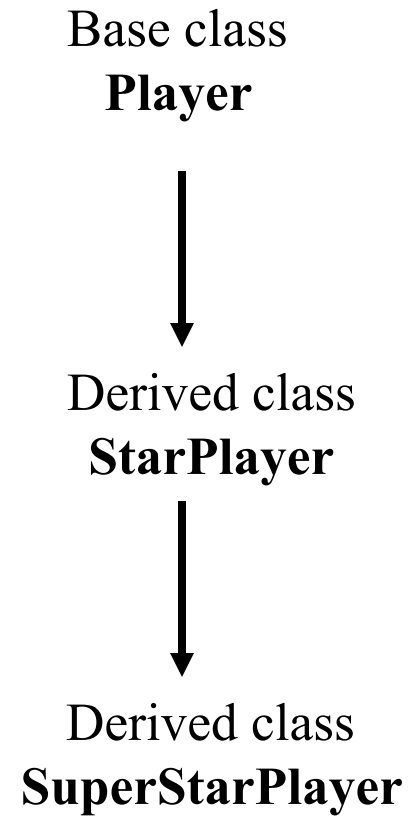
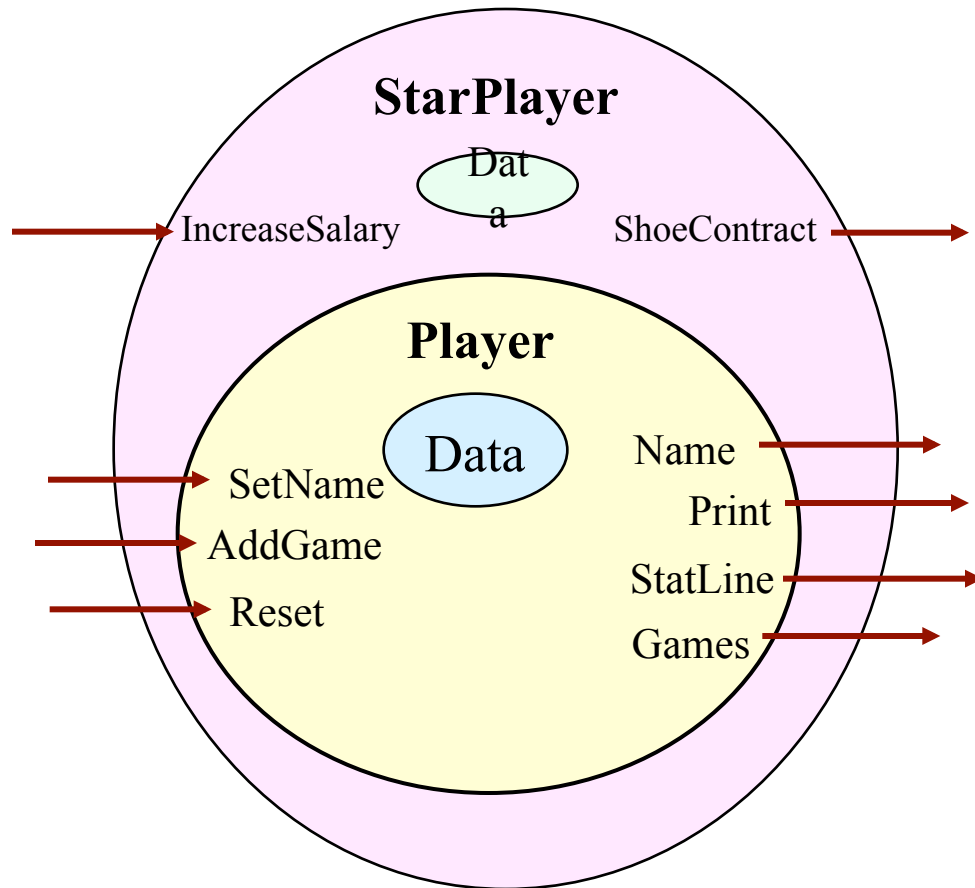
Reasons for object oriented programming

- Modularization
 - Abstraction – representing the essential features of something without including inessential detail
 - Encapsulation – grouping related things together
 - Information hiding – expose only what you want
- Inheritance
- Polymorphism

Inheritance

- It is possible to *extend* existing classes without knowing much about them
 - Add whatever new behavior you want
- Example:
 - You have a class that represents a “player”
 - Create a new class that is a “star player”
 - ◆ Most of the behavior and attributes are the same, but a few are different – specialized

Inheritance (cont.)



Inheritance (cont.)

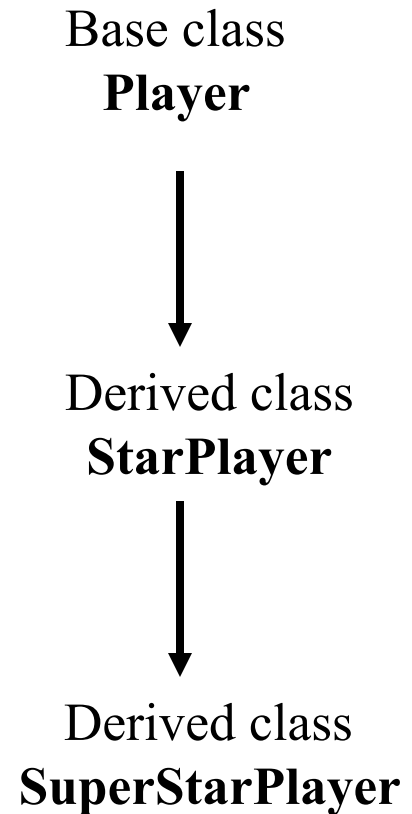
- A **SuperStarPlayer** is a **StarPlayer** is a **Player**
- Any function that takes a **Player** can be passed a **StarPlayer** or a **SuperStarPlayer**

```
float PPG(Player p)
{
    return (p.points/p.games) ;
}
```

If **points** and **games** are ints, should be this:

```
return ((float)p.points/p.games) ;
```

This assumes that the data variables **points** and **games** are public (but they will not be public – compiler error!)



Polymorphism

- The ability of different objects to respond to the same *message* in different ways.
- Tell an **int** to print itself: `i.print()` ;
- Now tell a **double**: `x.print()` ;
- Now tell a **Player**: `p1.print()` ;

Or: `cout << i;`
`cout << x;`
`cout << p1;`

Example: Player.h

```
class Player {  
private:  
    std::string name;  
    int games;  
    int points;  
    int rebounds;  
    int assists;
```

Data (private)

```
public:  
    void Print();  
    bool AddGame(int, int, int);  
    std::string Name();  
    void SetName(std::string);  
    void Reset();  
};
```

Functions (public)

```
void Player::Print()
```

Example: Player.cpp

```
{  
    cout << name << " " << rebounds << " " <<  
    assists << endl;  
}
```

```
void Player::AddGame(int points, int rebs,  
    int assts)
```

```
{  
    game++; this->points += points;  
    rebounds += rebs; assists += assts;  
}
```

When defining class functions, data can be accessed directly
(no “**Player::**” required)

```
inline std::string Player::Name()  
{  
    return name;  
}
```

```
inline void Player::SetName(std::string name)  
{  
    this->name = name;  
}
```

```
inline void Player::Reset()  
{  
    games = points = rebounds = assists = 0;  
}
```

```
Player p1, p2;
```

Example: main.cpp

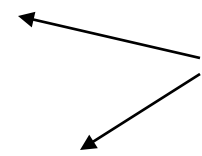
```
p1.SetName("Steve Nash");
```

```
p1.AddGame(48, 5, 5);
```

```
p2.SetName("Dirk Nowitzki");
```

```
p2.AddGame(25, 8, 3);
```

Maybe do this at
initialization time?



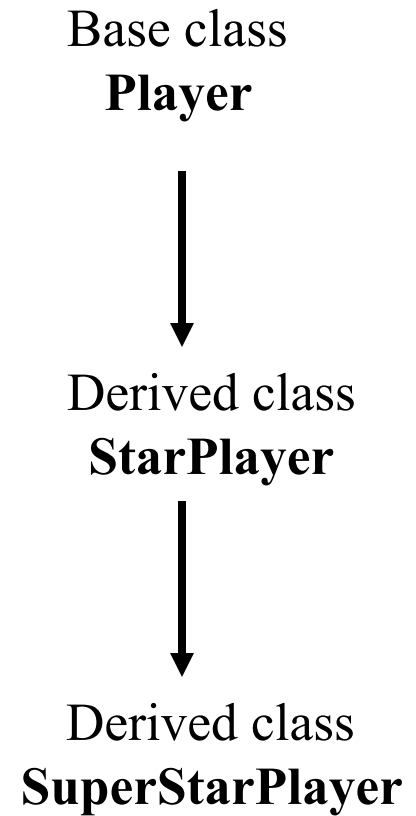
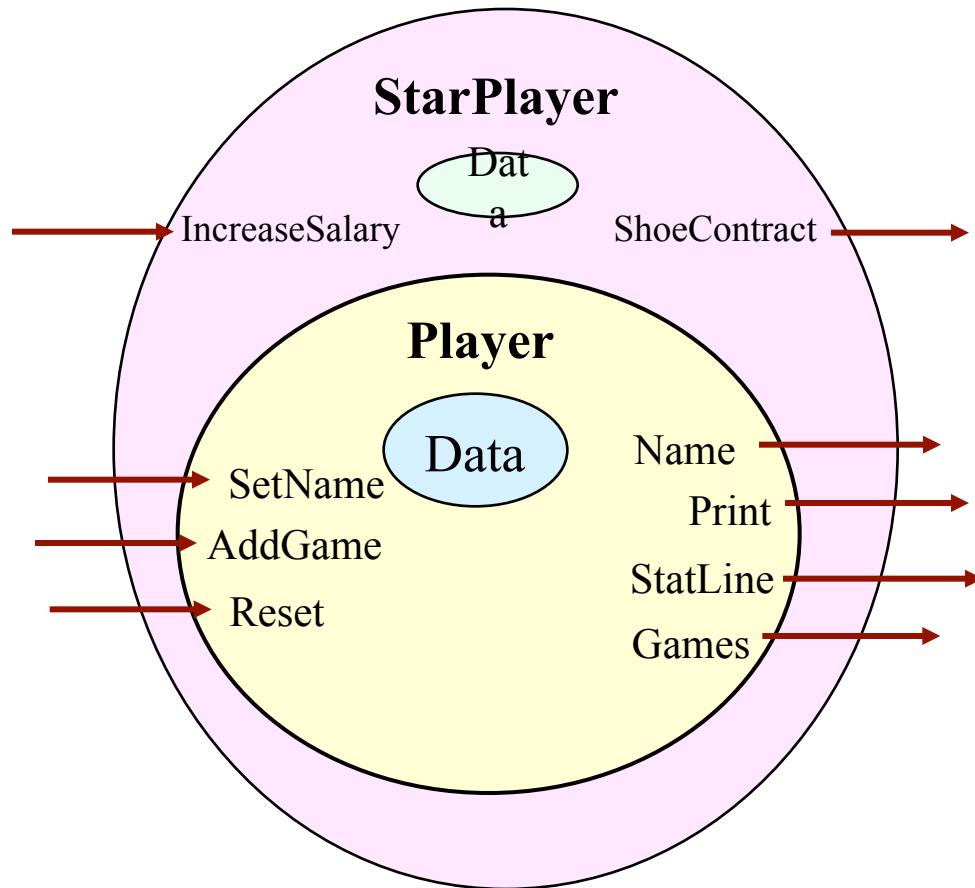
```
std::cout << p1.Name() << "has scored " <<  
    p1.points << endl;
```

No, can't access this member variable – it's private!



What are the initial values of the data variables? Undefined

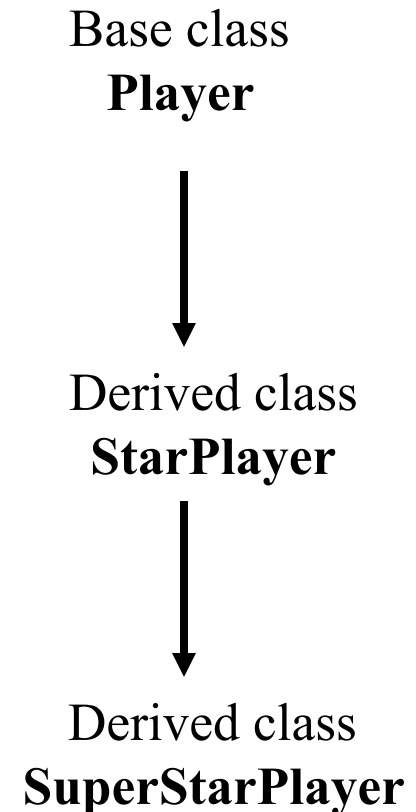
Inheritance (cont.)



Inheritance (cont.)

- A **SuperStarPlayer** is a **StarPlayer** is a **Player**
- Any function that takes a **Player** can be passed a **StarPlayer** or a **SuperStarPlayer**

BUT NOT VICE VERSA!!



Player



StarPlayer



SuperStarPlayer



Player
StarPlayer
SuperStarplayer

Function1 (Player p) 

StarPlayer
SuperStarplayer

Function2 (StarPlayer p)  

SuperStarplayer

Function3 (SuperStarPlayer p)   

```
class Player {  
private:  
    std::string name;  
    int games;  
    int points;  
    int rebounds;  
    int assists;  
public:  
    void Print();  
    bool AddGame(int, int, int);  
    std::string Name();  
    void SetName(std::string);  
    void Reset();  
};
```

Data (private)

The class interface



Functions (public)

```
class StarPlayer : public Player {
private:
    int extraMillions;
    std::string sponsor;
public:
    void IncreaseSalary(int millions);
    std::string& ShoeContract();
};
```

```
class SuperStarPlayer : public StarPlayer {
public:
    void Print();
};
```

```
void StarPlayer::IncreaseSalary(int m)
{
    extraMillions += m;
}

std::string& StarPlayer::ShoeContract()
{
    return(sponsor);
}

void SuperStarPlayer::Print()
{
    std::cout << "The fabulous ";
    Player::Print();
}
```

```
class Player {
```

```
private:
```

```
    // data
```

```
public:
```

```
    // functions
```

```
    int Points() { return points; }
```

```
    Player();
```

```
    Player(std::string name);
```

```
};
```

Functions can be defined in the class definition itself

Automatically inlined

Constructor functions

```
Player p1;
```

```
Player p2("Larry Bird");
```

Notice: No return value type for constructor (and destructor) functions!

```
Player::Player()
```

```
{
```

```
    name = "";
```

```
    Reset();
```

```
}
```

games = points = rebounds = assists = 0



```
Player::Player(str::string str)
```

```
{
```

```
    name = str;
```

```
    Reset();
```

```
}
```