cs121 - software development
code design: classes

alexandre r.j. françois

visiting associate professor of computer science

© arjf 2010

outline

- reasons to create a class
- interface
  - abstraction
  - encapsulation
- inheritance
- containment
- member functions and data
- creators, assignment operator
- more design principles

© arjf 2010

reasons to create a class

model real-world objects

model abstract objects

reduce complexity

isolate complexity

- bring all related code into a single place
- simplify interface seen by rest of system

hide implementation details

limit effects of change

© arjf 2010

interface: good abstraction

abstraction: the ability to view a complex operation in a simplified form

provide consistent abstractions

- coherence: move unrelated information to another class

present a consistent level of abstraction in the class interface

- example: game state has a list of platformers
- provide “player” level access or “list” level access, but not both!

© arjf 2010

interface: good encapsulation

keep class data members private

- principle of encapsulation, aka information hiding
- provide setters and getters when appropriate

avoid friend classes

© arjf 2010

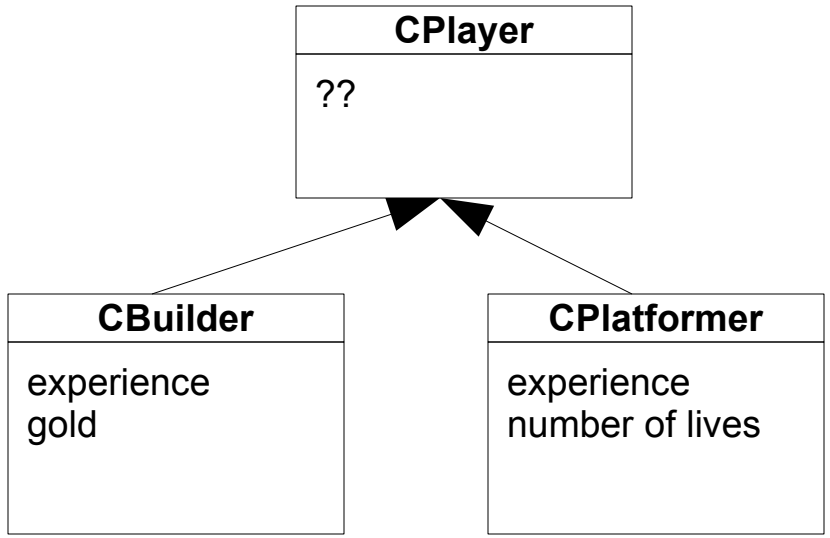
inheritance (is-a)

implement is-a relationship through public inheritance

liskov's substitution principle: an instance of a subclass can be substituted for an instance of the superclass without causing any problem

platformer is-a player

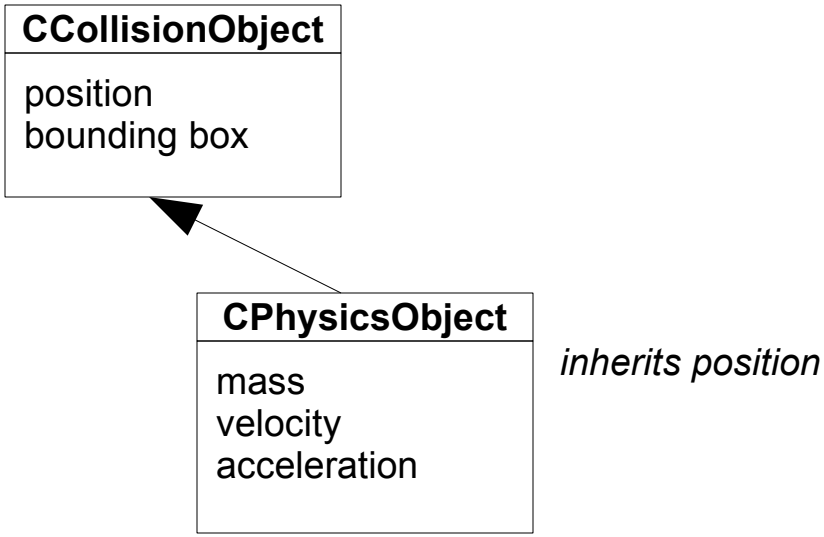
builder is-a player



© arjf 2010

inheritance

move common interface, data, behavior as high as possible in the inheritance tree



make all data private, not protected

© arjf 2010

inheritance

prefer polymorphism to extensive type checking

virtual `OpenGLRender()` method
of `CCollisionObject`
overridden by all game elements

© arjf 2010

inheritance

never redefine an inherited non virtual function

- don't re-use names of non-overridable base class routines in derived classes

never redefine an inherited default parameter value

be suspicious of classes that override a routine and do nothing inside the derived routine

© arjf 2010

inheritance

avoid deep inheritance trees

avoid private inheritance

- unless you know what you are doing

avoid multiple inheritance

- unless you know what you are doing

© arjf 2010

containment (has-a)

implement has-a relationship through containment

- data members

game state:

- has-a level description
- has-a builder
- has-a list/set/array? of platformers
- has-a clock
- etc.

© arjf 2010

member functions and data

minimize indirect routine calls to other classes

law of demeter: an object can call its own routines and the routines of objects it creates, but should avoid calling routines of objects provided by the objects instantiated.

© arjf 2010

member functions and data

know what functions C++ silently writes and calls

- copy constructor, assignment operator, destructor, address-of operators, default constructor (if none provided) – all public!

disallow implicitly generated member functions and operators you don't want

- make private

© arjf 2010

c++ at work!

```
class Empty{};
```

is equivalent to:

```
class Empty{
public:
  Empty();                  // default constructor
  Empty(const Empty &rhs); // copy constructor
  ~Empty();                 // destructor
                            // not virtual unless
                            // by inheritance
  Empty&                        // assignment
  operator=(const Empty &rhs); // operator

  Empty* operator&();              // address-of
  const Empty* operator&() const;  // operators
};
```

© arjf 2010

generated only if needed...

```
const Empty e1;   // default constructor
                  // destructor

Empty e1(e2);     // copy constructor

e2=e1;            // assignment operator

Empty *pE1 = &e2; // address-of
                  // operator (non const)

const Empty *pE1 = &e1; // address-of
                        // operator (const)
```

© arjf 2010

creators

declare a copy constructor and an assignment operator for classes with dynamically allocated memory

prefer initialization to assignment in constructors

initialize all member data in all constructors, if possible

prefer deep copies to shallow copies until proven otherwise

© arjf 2010

assignment operator

have `operator=` return a reference to `*this`

assign to all data members in `operator=`

check for assignment to self in `operator=`

© arjf 2010

more design principles

open-close principle

- open for extension, close for modification

single responsibility principle

no forgery

- keep data in a single place

one rule, one place

- don't duplicate code

© arjf 2010

platitude

classes from the sai architectural design

other classes:

- common data structures (objects, players, etc.)
- message protocols (commands, networking, etc.)

© arjf 2010