The meaning of OO, the conclusion

Comp-304 : The meaning of OO, the conclusion
Lecture 7

Alexandre Denault
Original notes by Hans Vangheluwe
Computer Science
McGill University
Fall 2006

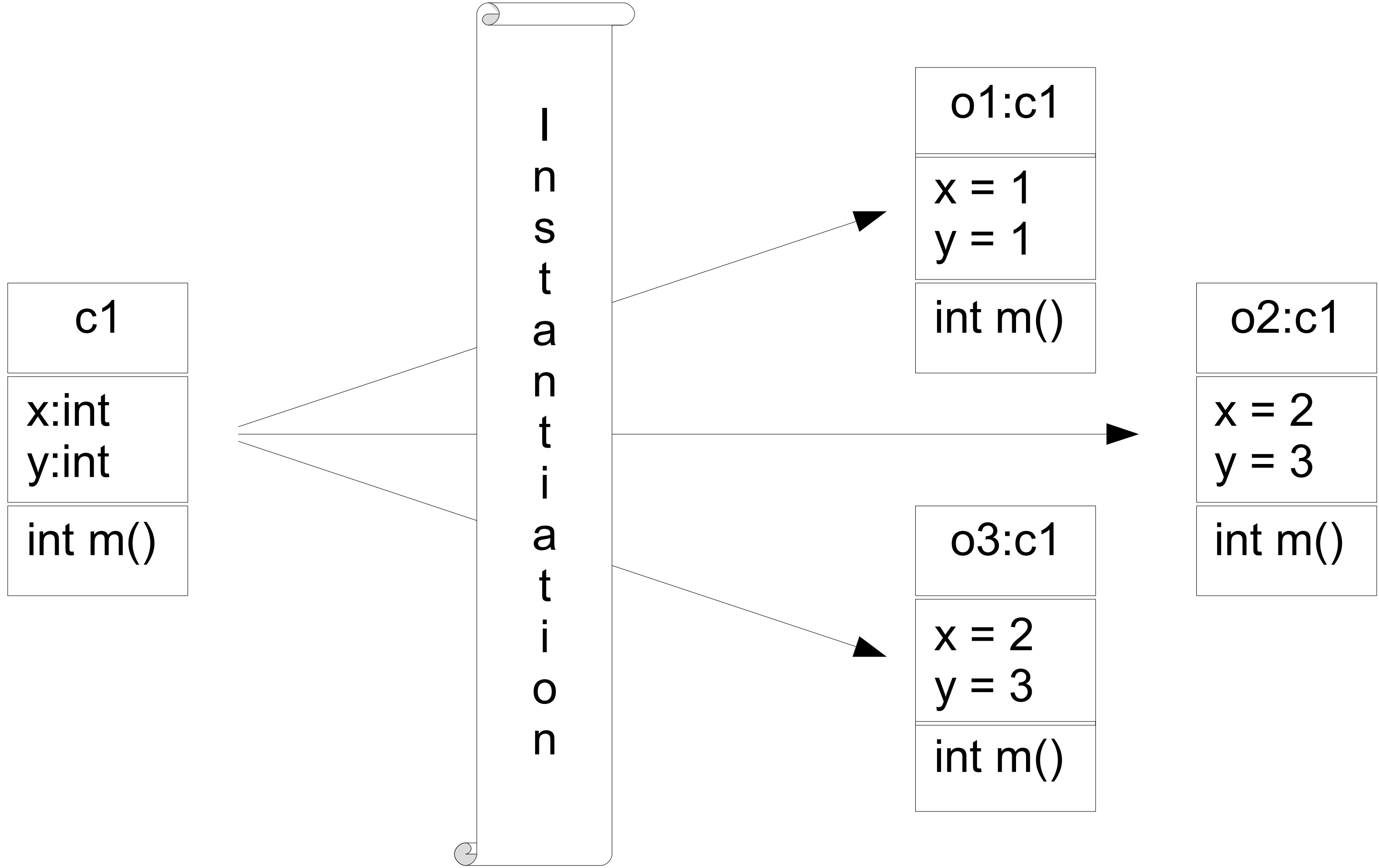
Recap

1) Encapsulated
2) State Retention
3) Implementation / Information Hiding
4) Object Identity
5) Messages
6) Classes
7) Inheritance
8) Polymorphism
9) Generacity

Classes

- A class is the stencil from which objects are created (instantiated).
- Each object has the same structure and behavior as the class from which it is instantiated.
  - same attributes (same name and types)
  - same methods (same name and signature)
- If object **obj** belongs to class C
  - then **obj** is an instance of C.
- So, how do we tell objects apart?
  - Object Identity

Instantiation



Classes vs Objects

- Classes are static and are evaluated at compile time.
  - Only one copy of the class exist.
  - Memory to store methods is only allocated once.
- Objects are dynamic and are created at run time.
  - One copy of the object is created every time the object is instantiated
  - Thus, memory to store the attributes is allocated for every instantiated object.

Instantiating Ghosts

Class
Ghost

Instantiating Pellets

Class
Pellets

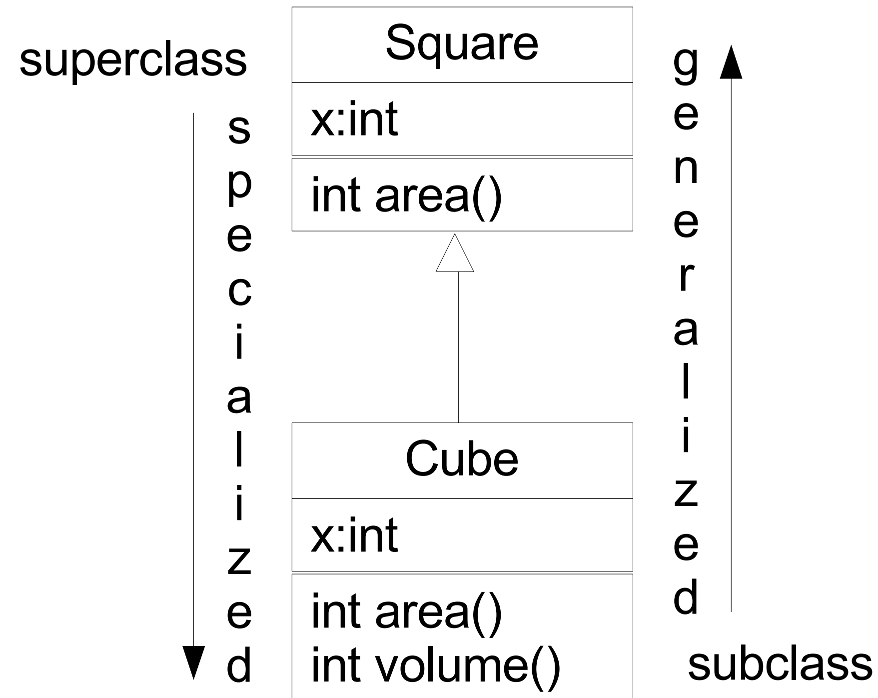
Inheritance

- Suppose you have classes c1 and c2. At design time, you notice that everything in c1 (attributes and methods) should also be in c2, plus some extra stuff.
- Instead of rewriting all of c1's code into c2, we say that c2 inherits from c1.
- Thus, c2 has defined on itself (implicitly) all the attributes and methods of c1, as if the attributes and methods had been defined in c2 itself.

Relationship

- Inheritance is an “is a” relationship
- Suppose we have a class MotorVehicle
  - A Automobile is a MotorVehicle
  - A Motorcycle is a MotorVehicle
- We call MotorVehicle the superclass and Automobile is a subclass
  - MotorVehicle is more generalized
  - Automobile is more specialized

Specialization



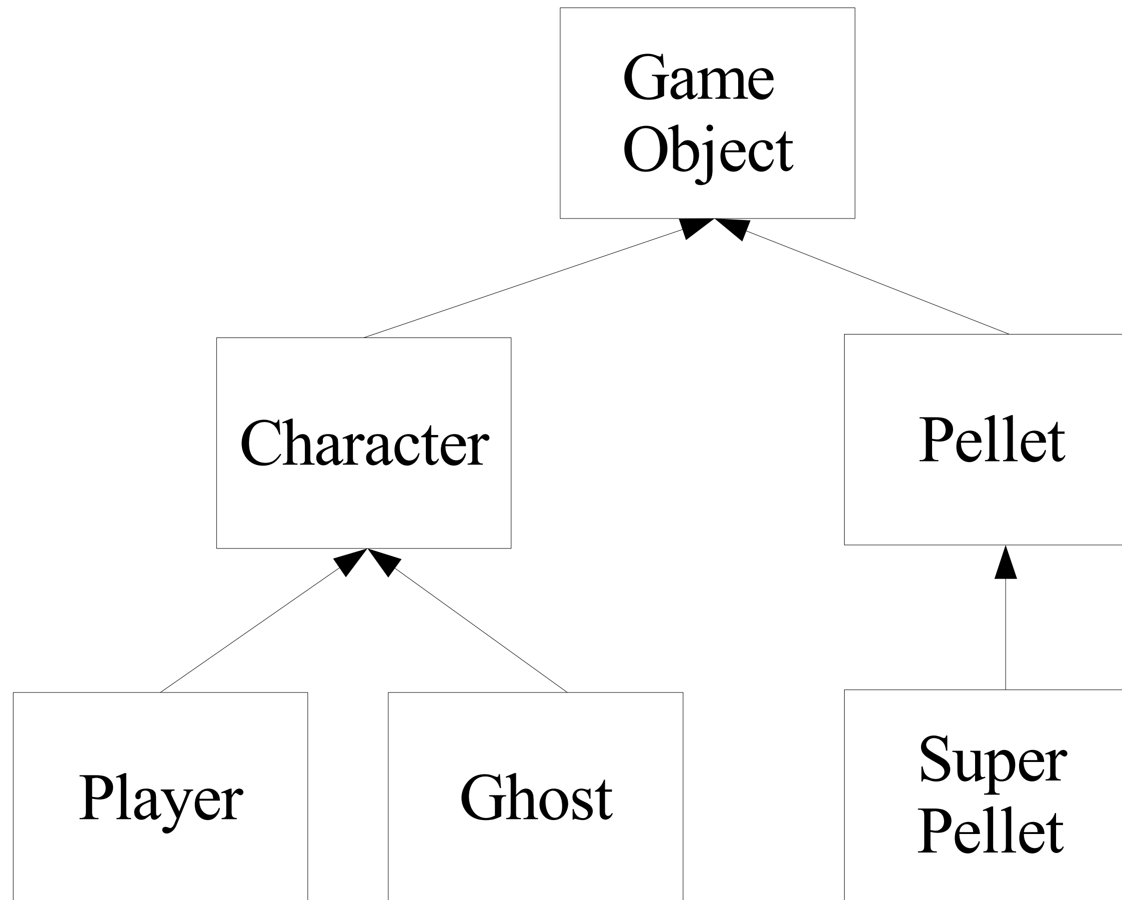
Type Family

- A type family is defined by a type hierarchy.
- At the top of the hierarchy is a supertype that defines behavior common to all family members.
- Other members are subtypes of this supertype.
- A hierarchy can have many levels.
- Type hierarchy can be used
  - to define multiple implementations of a type that are more efficient under particular circumstances.
    - Vector & LinkedList implement Collection
  - to extend the behavior of a simple type by providing extra methods
    - BufferedReader extends Reader

Substitution Principal

- A supertypes behavior must be supported by all subtypes.
- Therefore, in any situation in which a supertype can be used, it can be substituted by a subtype.
- Most compilers enforces this by only allowing extensions to a type
  - you can only redefine and add methods, not remove them.
- The substitution principle provides abstraction by specification for type hierarchies:
  - Subtypes behave in accordance with the specification in their supertype.

Inheritance In Pacman



Multiple Inheritance

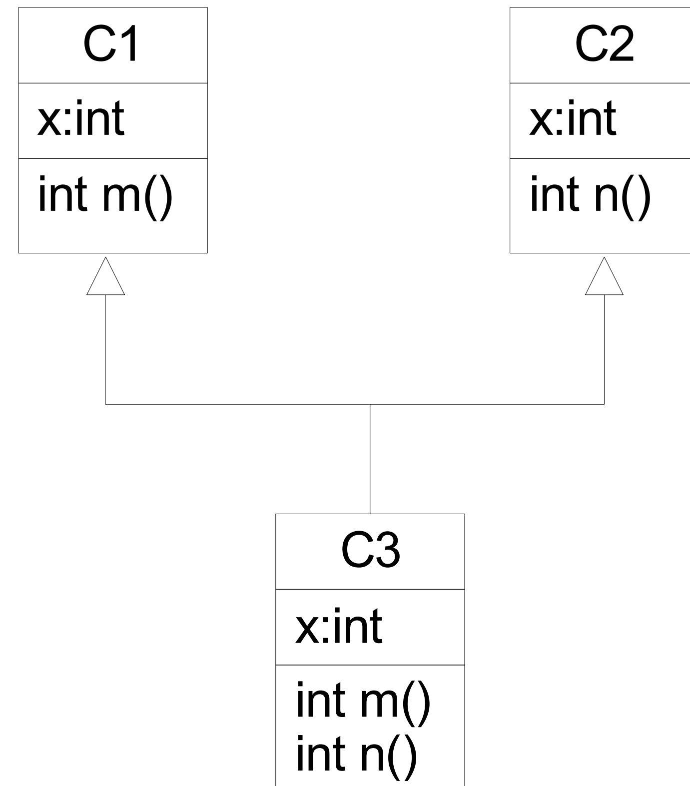
- Many classes can inherit from one class
- One class can inherit from many classes
  - Why is this good ?
  - Why is this bad?

The Good

- Allows code reuse
  - code in superclasses doesn't have to be rewritten in subclasses
- Ease of maintenance
  - if we add an attribute to a superclass, all subclasses will automatically inherit it

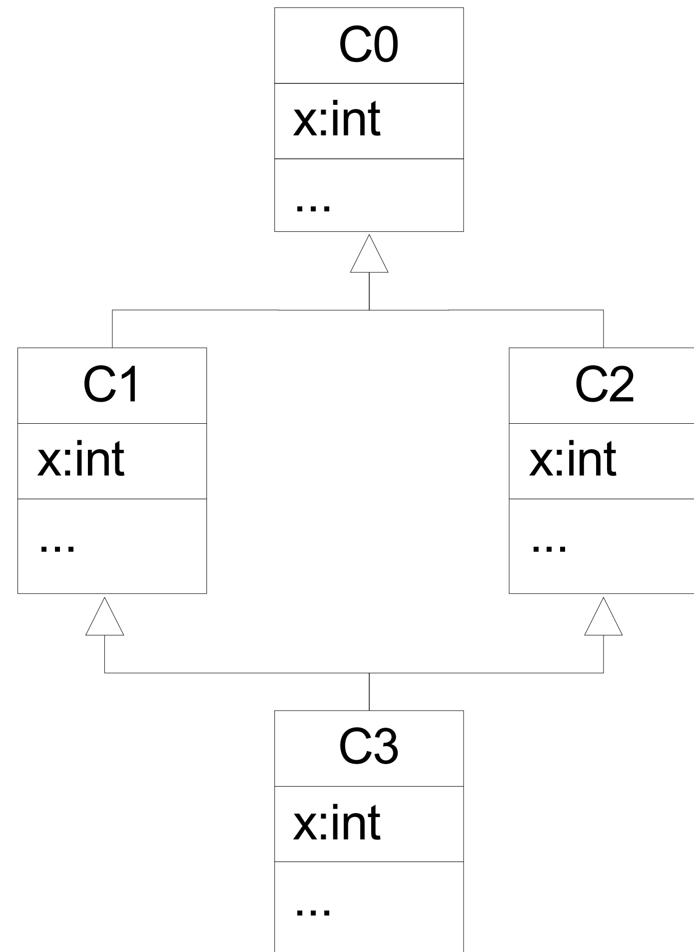
The Bad

- If one class can inherit from many classes, we may get multiple inheritance
- Which x should C3 inherit, the one from C1 or the one from C2?
- How can this be taken care of?



The Worse

- If many classes can inherit from one class, we may get repeated inheritance
- C1 and C2 inherit x from C0. Now, they are all the “same” x, but which x does C3 inherit?



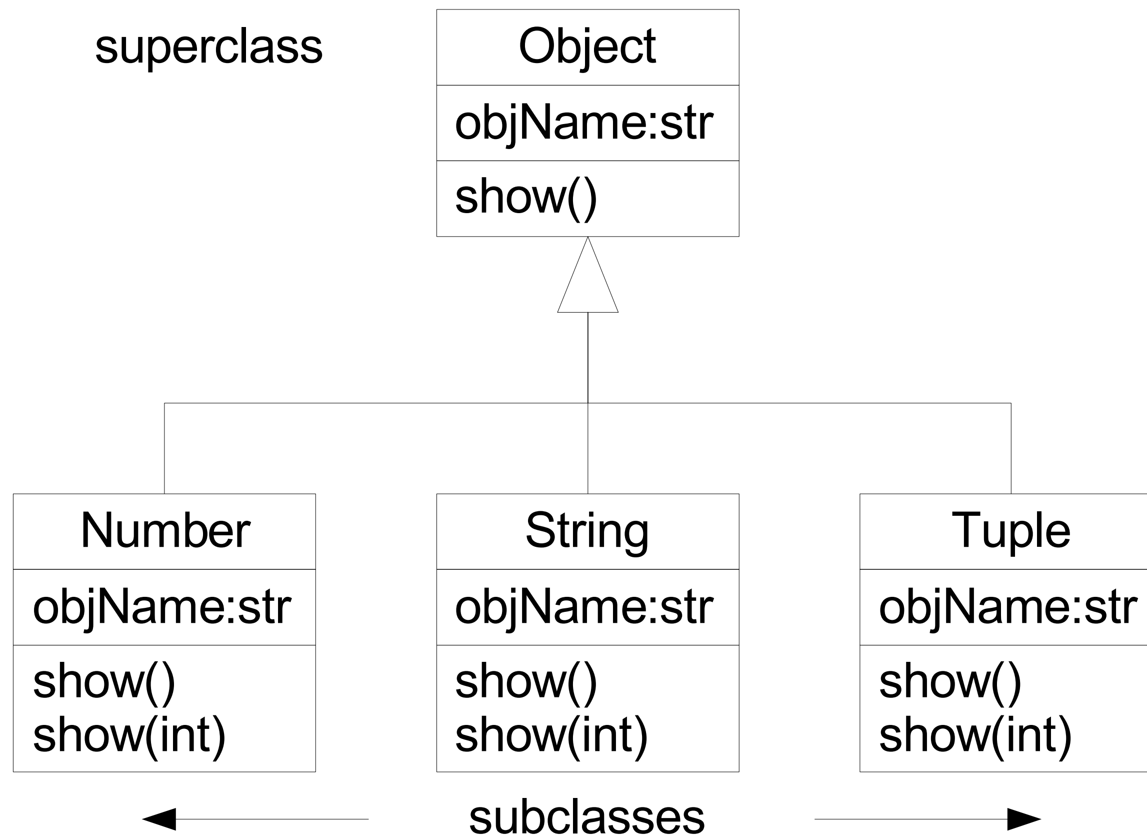
Polymorphism

- A single method (or attribute) defined on more than one class that may take on different implementations in each different class
- An attribute or variable that may refer to objects of different classes at different times during program execution
- Polymorphism literally means many forms in Greek

Real type vs Apparent type

- Collection myVar = new LinkedList()
- The apparent type of myVar is Collection.
  - At compile time, the compiler only keeps track of the apparent type of a variable.
- The real type of myVar is LinkedList.
  - At run time, in most programming language, the application keeps track of the real type of a variable.

Example of Polymorphism



First definition

- Method show() is a form of polymorphism, as per the first definition.
- When we call someObject.show(), the object which is being referenced will know how to show itself
- It must be ensured that show() is properly implemented for each subclass (and possibly the superclass) and that the user need not worry about the implementation

Which show() to call?

- Which show() to execute will be determined at run-time (and NOT at compile-time). This is known as dynamic, run-time or late binding
- Consider this code

```
Object o
o = Object.new()
s = String.new()
t = Tuple.new()
...
if user says string : o = s
else : o = t
...
o.show()
```

Second Definition

- At run-time, the object o may be an object of type String or of type Tuple.
- What o actually is will only be determined at run-time, after the user's input.
- When o.show() is executed, the method show() of the appropriate object will be executed.
- Attribute o is an example of polymorphism, as per the second definition, because it can point to objects of different types.

Overloading vs Overriding

- Overriding is the redefinition of a method defined on a class C in one of C's subclasses.
- Overloading of a name or symbol occurs when several operations (or operators) defined on the same class have that name or symbol.
  - We say that the name or symbol is overloaded.

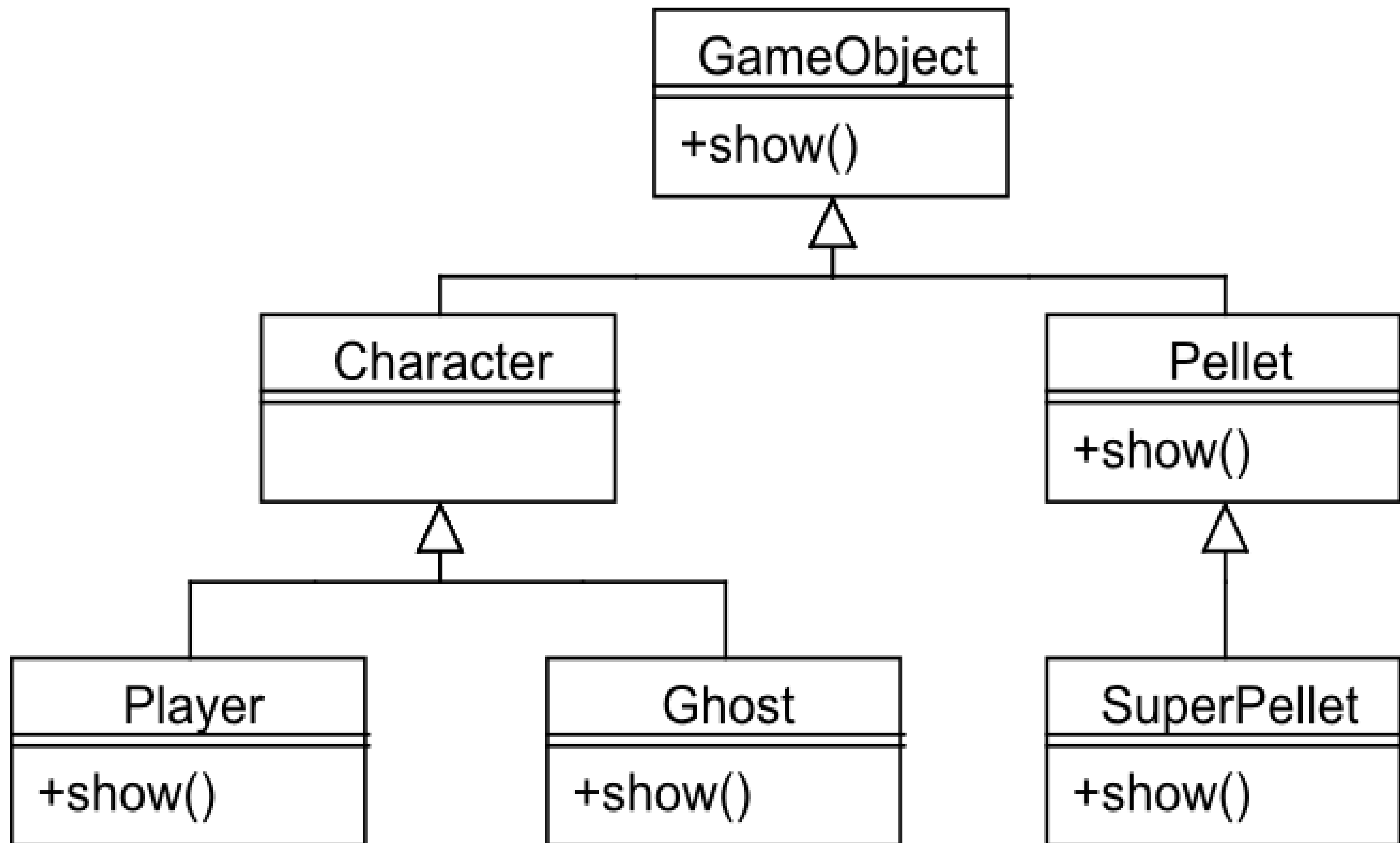
Overridding

- show() is an example of overriding because subclasses Number, String and Tuple redefined show() to suit their needs.
- If we wish to actually execute show() of the superclass (Object), we would execute super.show() in the subclass.
- Overriding can also be used to cancel certain inherited methods.
  - Suppose we have a subclass Hash that cannot show itself, then we can override show() in class Hash to return some error.
  - This is not clean O.O., but it is a practical solution.

Overloading

- show(int) is an example of overloading
  - show() will show the object at some default size
  - show(int) will show the object at some ratio, passed as an argument
- Which method will be executed depends on which method signature is used to call it.

Pacman : show()



More tricky

- If B and C are subclasses of A.
- Class D has the following methods.
  - show(B b)
  - show(C c)
- What happens if?
  - A var = new B();
  - d.show(var)
- Depends on the lookup:
  - Lookup uses apparent type : call is ambiguous
  - Lookup is dynamic : call to show(B b) is made

Genericity

- Imagine I spend thousands of dollars developing an algorithm to sort trees of integers.
- I don't want to rebuild the algorithm if I store floats or strings in the trees.
- I want a generic algorithm for all trees containing items that can be compared.
- Solution : Genericity (also known as templates)

Definition

- Genericity – one or more classes that are used internally by some class and are only supplied at run-time (or upon instantiation)
- Genericity can be emulated using inheritance.

Suppose ...

- Suppose
  - we code a class IntArray which ENTIRELY deals with the ins and outs of arrays and array operations (the array holds ints)
- Suppose
  - we code a class StrArray which ENTIRELY deals with the ins and out of arrays and array operations (the array holds strs)
- We will notice that all of the code in IntArray and StrArray will be identical except for the type of element that the array holds.
- Instead of having two (or more) separate classes, we should have one class called Array and parameterize it.
- We write Array <ElementType> where ElementType will be the class (or type) of the element that the array will store.