Inheritance: Designing

CPSC 233: Introduction to Computer Science for Computer Science Majors II
Winter 2022

Jonathan Hudson, Ph.D.
Instructor
Department of Computer Science
University of Calgary

Wednesday, 10 November 2021

UNIVERSITY OF CALGARY

Outline

- Creating Subclasses
- Overriding Methods
- Class Hierarchies
- ➡ Designing for Inheritance

Inheritance Design Issues

- Every derivation should be an is-a relationship
- Think about the potential future of a class hierarchy, and design classes to be reusable and flexible
- Find common characteristics of classes and push them as high in the class hierarchy as appropriate
- Override methods as appropriate to tailor or change the functionality of a child
- Add new variables to children, but don't redefine (shadow) inherited variables

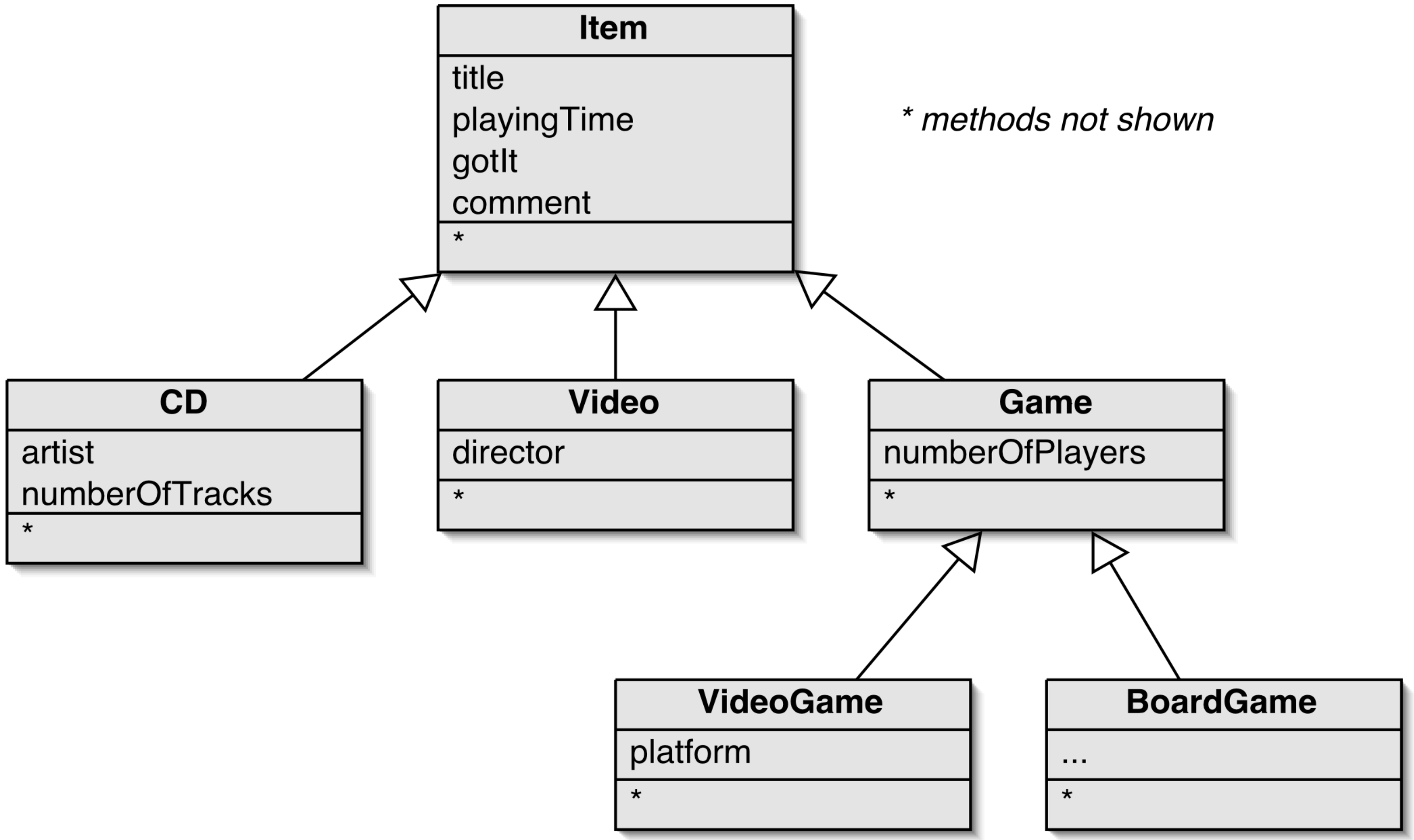
Inheritance Design Issues

- Allow each class to manage its own data; use the `super` reference to invoke the parent's constructor to set up its data
- Even if there are no current uses for them, override general methods such as `toString` and `equals` with appropriate definitions
- Use abstract classes to represent general concepts that lower classes have in common
- Use visibility modifiers carefully to provide needed access without violating encapsulation

Restricting Inheritance

- The `final` modifier can be used to curtail inheritance
- If the `final` modifier is applied to a method, then that method cannot be overridden in any descendent classes
- If the `final` modifier is applied to an entire class, then that class cannot be used to derive any children at all
  - Thus, an abstract class cannot be declared as `final`
- These are key design decisions, establishing that a method or class should be used as is

Deeper Hierarchies



** methods not shown*

Review of Inheritance

- Inheritance (so far) helps with:
    1. Avoiding code duplication
    2. Code reuse
    3. Easier maintenance
    4. Extendibility

Subtypes

We've seen that any object has a type (Date, Currency, String, ...)

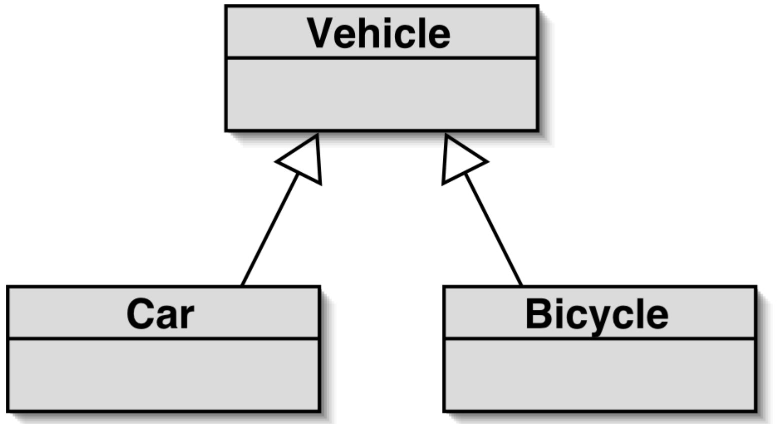
Types of classes have subtypes (subclasses define subtypes)

eg. type **Student** is a subtype of type **Person**

You can think of an object of a derived class as having **two** types – that of the **superclass and the subclass**.

1. So you can use it (e.g. in parameter list or assignment) where either type is legal
2. Objects of subclasses can be used where objects of supertypes are required. (This is called substitution .)

Subtypes (An Example)



subclass objects may be assigned to superclass variables

```
Vehicle v1 = new Vehicle();
Vehicle v2 = new Car();
Vehicle v3 = new Bicycle();
```

Polymorphism

We will learn about polymorphism

- defining polymorphism and its benefits
- Review of overriding methods in subclasses
- Run-time (late) binding of methods

Polymorphism

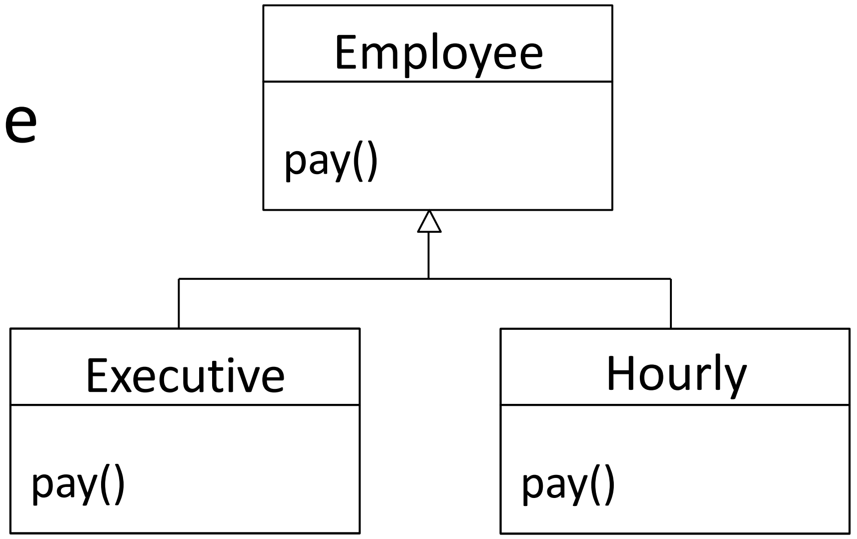
A subclass

- Inherits instance variables from its superclass
- Inherits methods from its superclass
- can redefine, or *override* superclass methods
  - ⇨ So can customize their behaviour

The method actually called depends on the type of the object

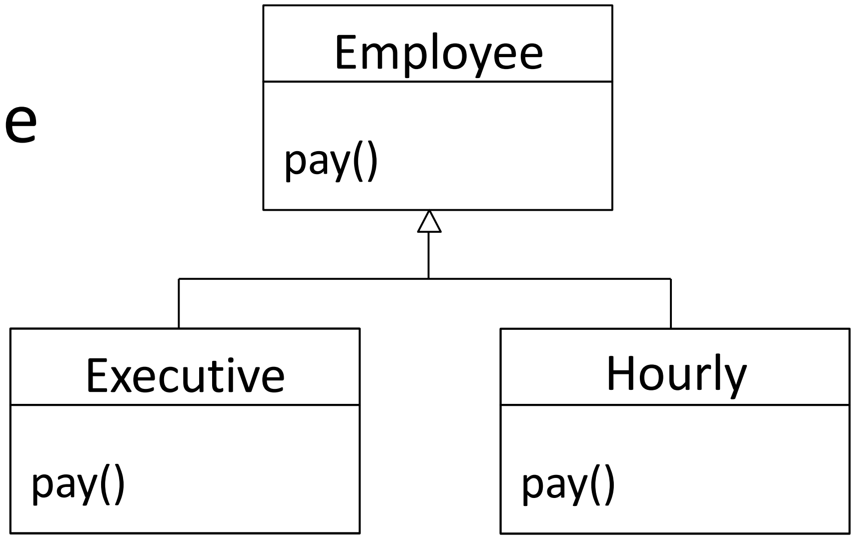
Example - Employees

- Hourly workers paid by the hour, 1½ for overtime
- Execs paid salary plus % profit
  - Have objects named ***emp, exec, hourly***
  - What code is run if have:
    - emp.pay(), exec.pay(), hourly.pay()



Example - Employees

- Hourly workers paid by the hour, 1½ for overtime
- Execs paid salary plus % profit
  - Have objects named ***emp, exec, hourly***



  - Have array of Employees with **all** employees in it
  - What code is run if have employee.get(i).pay()?
    - how does this work? **polymorphism**

```java
ArrayList<Person> people = new ArrayList<>();

Staff staff = new Staff("staff_name", 100, null);
Student student = new Student("student_name", 200, new ArrayList<Session>());
Faculty faculty = new Faculty("faculty_name", 300, new ArrayList<Session>());

Person p1 = (Person) staff;
Person p2 = (Person) student;
Person p3 = (Person) faculty;

people.add(staff);
people.add(student);
people.add(faculty);

for (Person p : people) {
    System.out.println(String.format("Class->%s", p.getClass()));

    System.out.println(String.format("Person->%s", p instanceof Person));
    System.out.println(String.format("Staff->%s", p instanceof Staff));
    System.out.println(String.format("Student->%s", p instanceof Student));
    System.out.println(String.format("Faculty->%s", p instanceof Faculty));

    System.out.println(p.printVersion());

    System.out.println();
}
```

Polymorphic Binding – How does it function?

- Consider obj.doIt();
- At some point, this invocation is bound to the definition of the method that it invokes
- If this binding occurred at compile time, then that line of code would call the same method every time
- Java defers method binding until run time
  - so the type of the object determines the method called
  - late binding or dynamic binding

```java
ArrayList<Person> people = new ArrayList<>();

Staff staff = new Staff("staff_name", 100, null);
Student student = new Student("student_name", 200, new ArrayList<Session>());
Faculty faculty = new Faculty("faculty_name", 300, new ArrayList<Session>());

Person p1 = (Person) staff;
Person p2 = (Person) student;
Person p3 = (Person) faculty;

people.add(staff);
people.add(student);
people.add(faculty);

for (Person p : people) {
    System.out.println(String.format("Class->%s", p.getClass()));

    System.out.println(String.format("Person->%s", p instanceof Person));
    System.out.println(String.format("Staff->%s", p instanceof Staff));
    System.out.println(String.format("Student->%s", p instanceof Student));
    System.out.println(String.format("Faculty->%s", p instanceof Faculty));

    System.out.println(p.printVersion());

    System.out.println();
}
```

```
Class->class Staff
Person->true
Staff->true
Student->false
Faculty->false
staff_name(100) Boss->null

Class->class Student
Person->true
Staff->false
Student->true
Faculty->false
student_name(200) Classes->[]

Class->class Faculty
Person->true
Staff->false
Student->false
Faculty->true
faculty_name(300) Sessions->[]
```

Onward to … Object Tools and Interfaces

Jonathan Hudson
jwhudson@ucalgary.ca
https://pages.cpsc.ucalgary.ca/~jwhudson/

UNIVERSITY OF CALGARY