CS-152: Inheritance, interfaces, and abstract classes

Neal Holtschulte

July 16, 2013
Many objects in a program will need similar instance variables and methods.
Let’s talk about a video game example.
It is common to create a general object with such shared instance variables from which many other objects can **inherit**.
General Object

The general object may have instance variables such as:

- x
- y
- direction
- speed
- hitPoints
General Object

This general object may have methods such as:

- move - move in the current direction at the current speed
- turn - change direction
- isAlive - return true or false based on whether or not object is alive
- takeDamage - takes an int argument and subtracts that amount from hit points
General Object

Lots of objects inherit from this generic object in the video game including:

- the player
- enemies
- vehicles
- the camera
Strange bug

A strange bug was discovered.

Explosions near the player cause the camera to stop moving.
Strange bug

What is going on?
Strange bug

What is going on?

The move command and turn command check to see if the object isAlive before executing.

Camera inherited from the generic object and only had the default hit points.
Strange bug

Occasionally an explosion near the player killed the camera without killing the player.

The camera stopped responding to move and turn commands.

The solution was to make the camera unkillable.
To inherit or not to inherit

Sometimes it makes sense to inherit methods and variables, but these particular game designers might have gone too far.
Asteroids

What sort of inheritance might you use in a game like Asteroids?
Most of the following is from

Java An Introduction to Programming and Problem Solving
by Savitch

Starting on page 557
Inheritance lets you define a general class, then define more specialized classes that add details to the general class.

These specialized classes “inherit” the methods and instance variables of the general class.
public class Person{
    private String name;
    public Person(String name){
        this.name = name;
    }
    public String getName(){
        return name;
    }
}
public class Student extends Person{
    private int grade;
    public Student(String name, int grade){
        super(name);
        this.grade = grade;
    }

    public int getGrade(){
        return grade;
    }

    public void setGrade(int grade){
        this.grade = grade;
    }
}
//Create a new student.
Student charlie = new Student("Charlie", 0);

//We can call methods of Student’s parent class
//on objects of type Student.
System.out.println(charlie.getName());

//We can also call methods of the Object as we
//normally would.
charlie.setGrade(100);
Is-a versus Has-a

Use inheritance only to model "is-a" relationships.

"has-a" relationships imply instance variables.
Is-a versus Has-a

A student is-a person.

A student has-a grade.
Is-a relationships

Quadrilateral

Parallelogram

Rhombus

Rectangle

Square
New Java syntax

```
public class Student extends Person{
    private int grade;
    ....
```
New Java syntax

The constructor should call super() which calls the super class (aka parent class) constructor with any arguments that the parent class’s constructor takes.

```java
public Student(String name){
    super(name);
}
```
Overriding

You can **override** a parent class method in a child class by creating a method of the same name with the same number and type of parameters.
Overriding

If the following was in the Student class, it would override the Person class’s method of the same name.

```java
public String getName() {
    return "Fred";
}
```
public class Person{
    private String name;
    public Person(String name){
        this.name = name;
    }

    public String getName(){
        return name;
    }
}

Reminder of Child class

```java
public class Student extends Person{
    private int grade;
    public Student(String name, int grade){
        super(name);
        this.grade = grade;
    }

    public int getGrade(){
        return grade;
    }

    public void setGrade(int grade){
        this.grade = grade;
    }
}
```
Private parent variables

A child class cannot directly access its parent class’s private variables or methods.

The Student class cannot have the following method:

```java
public void printName(){
    System.out.println(name);
}
```
Private parent variables

The Student class could have the following method:

```java
public void printName(){
    System.out.println(getName());
}
```
Question

What is the difference between overriding a method and overloading a method?
Suppose class SportsCar is derived from class Automobile. Automobile has private instance variables speed, manufacturer, and number of cylinders. Will an object of type SportsCar have instance variables speed, manufacturer, and number of cylinders?
Inheritance and constructors

Child classes do not inherit constructors. Parent or super class constructors must be called with `super();`. If you omit `super();` this will not break the code! The default parent class constructor will be called automatically.
Polymorphism is the ability to create a variable, a function, or an object that has more than one form.

Source: http://en.wikipedia.org/wiki/Polymorphism_in_object-oriented_programming
Polymorphism in biology is when two or more clearly different phenotypes exist in the same population of a species.

Source: http://simple.wikipedia.org/wiki/Polymorphism
Polymorphism

Polymorphism is the ability of one type, A, to appear as and be used like another type, B. Polymorphism usually means that type A somehow derives from type B, or type C implements an interface that represents type B.

Source:

http://www.princeton.edu/~achaney/tmve/wiki100k/docs/Polymorphism_in_object-oriented_programming.html
Polymorphism

The primary usage of polymorphism in industry (object-oriented programming theory) is the ability of objects belonging to different types to respond to method, field, or property calls of the same name, each one according to an appropriate type-specific behavior.

Source:

http://www.princeton.edu/~achaney/tmve/wiki100k/docs/Polymorphism_in_object-oriented_programming.html
Polymorphism

The different objects involved only need to present a compatible interface to the clients’ (calling routines).

Source:

http://www.princeton.edu/~achaney/tmve/wiki100k/docs/Polymorphism_in_object-oriented_programming.html
Polymorphism allows us to treat RandomCritters and NealsCritters as simply Critters.

See Critter Source Code in World.java, specifically this line:

```
Critter animal = grid[r][c].getOccupant();
```
Polymorphism
Polymorphism

Quadrilateral methods can be called on parallelograms, rhombuses, rectangles, and squares.

Parallelogram methods can be called on rhombuses, rectangles, and squares.

etc
Interfaces and Abstract Classes

When you want to write a contract without writing any code.

Abstract classes are partially implemented interfaces.

An interface is pure contract. No code.

Source: http://stackoverflow.com/questions/1913098/what-is-the-difference-between-an-interface-and-abstract-class
public interface Automobile{
    private int speed;

    public void drive();

    public int getSpeed();

    public void accelerate(double acceleration);
}
Abstract class

```
public abstract class Automobile{
    private int speed = 0;

    abstract void drive();

    public int getSpeed(){
        return speed;
    }

    abstract void accelerate(double acceleration);
}
```