CSE115 / CSE503
Introduction to Computer Science I

Dr. Carl Alphonce

343 Davis Hall

alphonce@buffalo.edu

Office hours:

Tuesday 10:00 AM – 12:00 PM*

Wednesday 4:00 PM – 5:00 PM

Friday 11:00 AM – 12:00 PM

OR request appointment via e-mail

**Tuesday adjustments: 11:00 AM – 1:00 PM on 10/11, 11/1 and 12/6*

ROADMAP

Last time

Iterators

Inheritance

Coding Exercise

Today

Inheritance

Coding Exercise

Coming up

Inheritance

int representation in detail

ANNOUNCEMENTS

EXAM 2 NOTICE

DATE: Tuesday November 15

TIME: 8:45 PM – 9:45 PM

LOCATION: as for exam 1

COVERAGE:

lecture material from 9/26 up to and including 11/04
lab material labs 4 – lab 9

readings: all assigned up to and including 13.4

HAVE A CONFLICT?

Send e-mail with PDF of documentation.

Subject line: [CSE115] Exam 2 conflict documentation

BRING: your UB card

NO ELECTRONICS: cell phone, calculator, etc.

UTA OFFICE HOURS

We are working on the office hour change.

Stay tuned!

Exercise solutions

Solutions to the exercises are in the repo:

Exercise-01

Exercise-02

Exercise-03

Exercise-04

Exercise-05

Ignore these – they're unrelated – we never used them:

Exercise-01-A

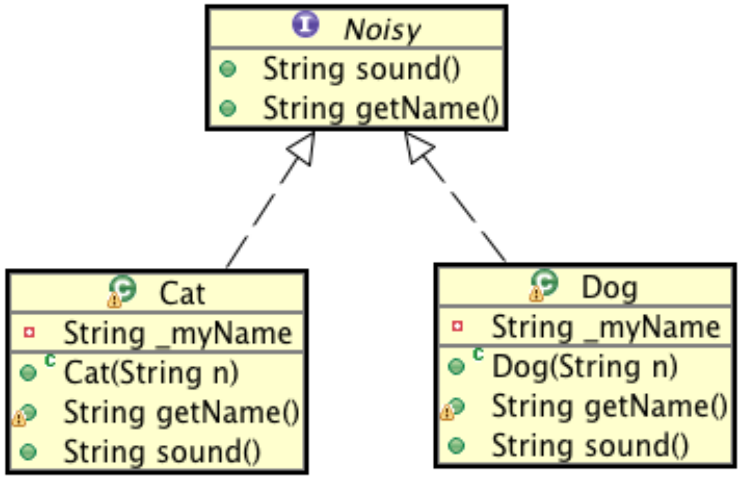
Exercise-01-B

Tomorrow
(Tuesday, November 8)
is
ELECTION DAY

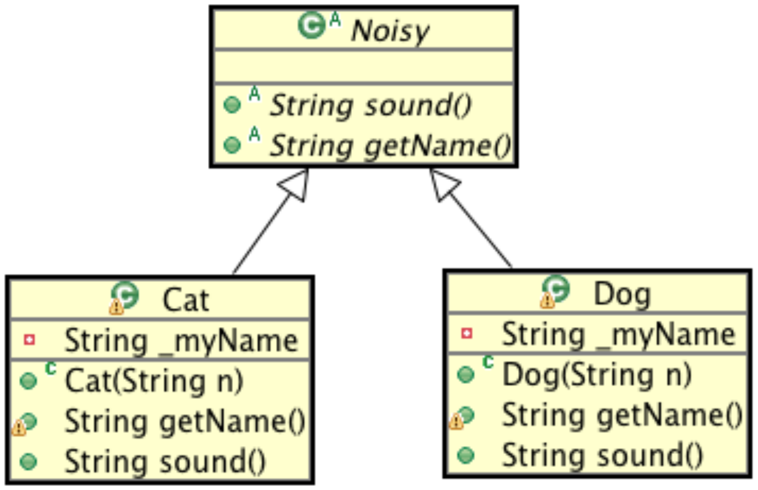
Get out and VOTE!
Make YOUR voice heard!

REVIEW

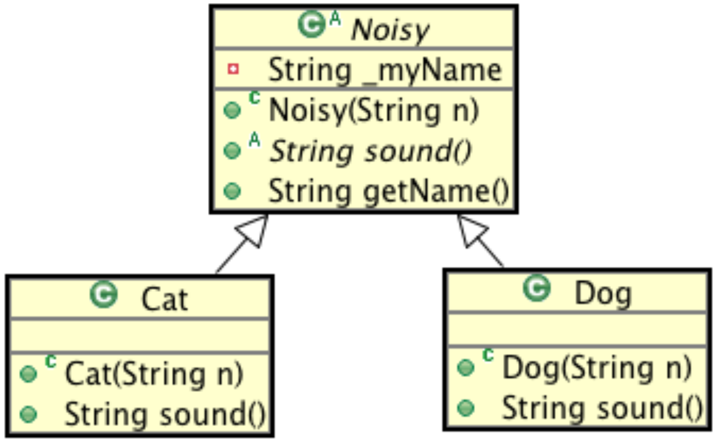
original design



class to class inheritance I



class to class inheritance II



Code duplication

a “code smell”

```
package noninheritance;

public class Cat implements Noisy {

    private String _myName;

    public Cat(String n) {
        _myName = n;
    }

    @Override
    public String getName() {
        return _myName;
    }

    @Override
    public String sound() {
        return "meow";
    }
}
```

```
package noninheritance;

public class Dog implements Noisy {

    private String _myName;

    public Dog(String n) {
        _myName = n;
    }

    @Override
    public String getName() {
        return _myName;
    }

    @Override
    public String sound() {
        return ″ruff";
    }
}
```

Refactored code

(-:   a breath of fresh air   :-)

```
public abstract class Noisy {
    private String _myName;
    public Noisy(String name) {
        _myName = name;
    }
    public abstract String sound();
    public String getName() {
        return _myName;
}   }
```

```
public class Cat extends Noisy{
    public Cat(String n) {
        super(n);
    }
    @Override
    public String sound() {
        return "meow";
}   }
```

```
public class Dog extends Noisy{
    public Dog(String n) {
        super(n);
    }
    @Override
    public String sound() {
        return ”ruff";
}   }
```

EXERCISE 06

PAIR CODING EXERCISE

Define a class quiz.Question. In this class define a method named answer.

Define this method so that it returns a count of all the elements of its argument, an ArrayList<String>, that have length 3. If the argument is null, return 0.

Submit to Exercise-06 in Web-CAT – enter the usernames of everyone in your group!

Submit no later than 6:00 PM tomorrow (11/8).

INHERITANCE

In Java inheritance can take place between

two interfaces

or

two classes

interface inheritance

In Java, an interface can extend zero or more interfaces:

extending zero interfaces

```
public interface A { public void foo(); }
```

extending one interface

```
public interface B extends A { public void bar(A a); }
```

extending many interfaces

```
public interface C extends B, ActionListener {
    public ActionEvent getEvent();
}
```

implements clause

A class can implement an arbitrary number of interfaces.

implements clause examples

```
public final class String
        implements java.io.Serializable, Comparable<String>, CharSequence
```

```
public abstract class Expression
        implements java.io.Serializable, ExpressionNode, XPathVisitable
```

Object

The class Object is pre-defined in the language.

Object does not extend any class, and is the only class that does not have a parent class.

Any class (other than Object) which does not have an explicit ‘extends’ clause in its header extends Object by default.

Note the distinction between Object (‘capital-O Object’), which is a class, and object (‘little-o object’), which refers to an instance of a class.

type vs class hierarchy

type hierarchy vs. class hierarchy

- class: single root – Object
- type: many roots

All instantiable types (concrete classes) fall under Object.

This means all objects (class instances) are of type Object.

class hierarchy vs. type hierarchy

Since both classes and interfaces define types, the class hierarchy is a sub-hierarchy of the type hierarchy.

The class hierarchy has a single root (Object).

The type hierarchy does not have a single root (since any interface which does not extend another interface is a root).

interface inheritance, cont'd

When an interface A extends another interface B, A inherits the methods specified by B.

This means that a class which implements A must define all the methods specified in both A and B.

An interface can have at most one specification for any given method: even if an interface inherits the very same method specification (same name, same parameter list) from two or more parent interfaces, the interface has the method specified just once.

Implications of “extends”

Same ***type*** implications as for interfaces:

- instance of subclass belongs to subclass type and superclass type

inheritance: non-private members of superclass can be accessed via subclass object.

- e.g. it’s as if methods of superclass were defined in subclass

class (implementation) inheritance

A (user-defined) class always extends exactly one (other) class:

```
public class Circle extends Shape {…}
```

If class header has no explicit extends clause, parent class is implicitly Object:

```
public class Shape {…}
public class Shape extends Object {…}
```

Object is the root of Java’s class hierarchy.

What is inherited?

Given what we know, a correct answer is that anything that is not private is inherited.

All our properties (instance variables) are private, so they are not inherited.

All our methods are public (not private), so they are inherited.

What is effect of inheritance?

A method inherited from a superclass to a subclass can be invoked on a subclass instance, even though not defined there:

```
public class Foo {
    private Bar _bar;
    public void setBar(Bar b) {
        _bar = b;
    }
}
public class FooSub extends Foo {
    …
}
```

This is legal:

```
new FooSub().setBar(new Bar())
```

implications of the type hierarchy

hierarchy of types

- implementation (class to interface)
- inheritance (class to class, interface to interface)

assignment

- variable of type T can be assigned a value of type S
  - where S and T are the same type
  - where S is a subtype of T

members

- where S is a subtype of T, all non-private members of both S and T are accessible on an instance of S

declared vs. actual type

- using a variable of type T, only methods defined for T may be called, even if object referred to has additional methods
- when a method is called using a reference whose type is T, but which refers to an object of type S (S a subtype of T), the definition for S is used. Essence of POLYMORPHISM

When it comes to methods defined in a superclass, a subclass has options:

inherit

totally override

partially override

When it comes to methods defined in a superclass, a subclass has options:

inherit

- provide no definition in subclass, accept definition from superclass

When it comes to methods defined in a superclass, a subclass has options:

- totally override
  - completely reject definition in superclass
  - provide entirely new definition in subclass

When it comes to methods defined in a superclass, a subclass has options:

partially override

- call superclass method, and
- provide additional code in subclass

sample code

(see examples in LectureCode repository, inheritance pacakge)