University at Buffalo
The State University of New York

cse@buffalo

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

Today

- Iterators
- Inheritance
- Coding Exercise

Coming up

- Inheritance

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?

- I will ask for documentation 11/7 – 11/9

BRING: your UB card

NO ELECTRONICS: cell phone, calculator, etc.

UTA OFFICE HOURS

Possible change to UTA office hours

Group them together into longer blocks with more UTAs present

Goal: same total number of UTA hours, but better able to handle volume of questions

REVIEW

java.util.Iterator<E>

An iterator guarantees to visit all members of a collection exactly once between the time it is created to the time its hasNext() method returns false

java.util.Iterator<E> interface

Three methods; we focus on these two:

public boolean hasNext()

- returns true if there are items of the underlying collection that still need to be visited
- returns false if there are no unvisited items

public E next()

- returns one of the unvisited items, if any
- should not be called if hasNext() returns false

for-each loop – process each element of a collection of elements of type E:

```
for (E item : collection) {
    // do something with item
}
```

Example: to print out each String in coll:

```
for (String s : coll) {
    System.out.println(s);
}
```

recall: for-each loop vs. while loop

```
java.util.Collection<String> coll;
coll = new java.util.LinkedList<String>();
coll.add("Fred");
coll.add("Wilma");
coll.add("Pebbles");
coll.add("Dino");
coll.add("Fred");
System.out.println("The collection: " + coll);
```

```
for (String s : coll) {

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

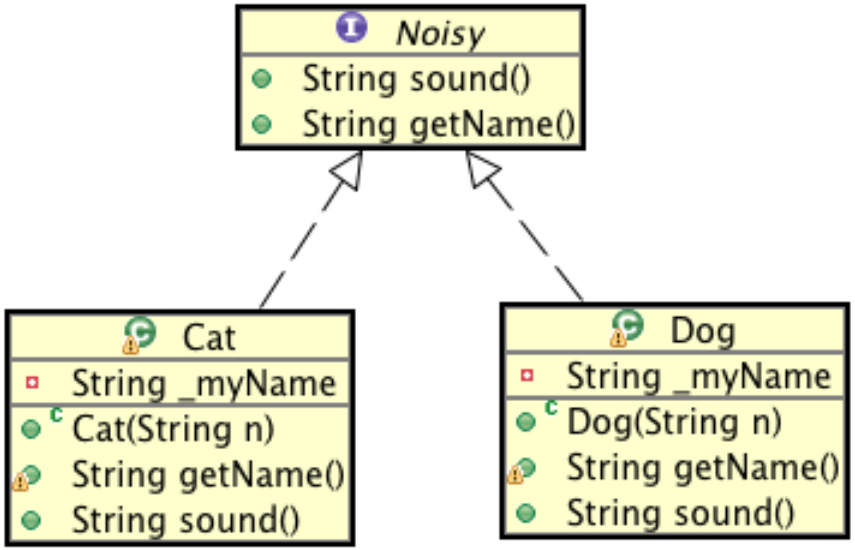
```
Iterator<String> it;
it = coll.iterator();
while (it.hasNext()) {
    String s = it.next();
    System.out.println(s);
}
```

```
java.util.Collection<String> coll;
coll = new java.util.LinkedList<String>();
coll.add("Fred");
coll.add("Wilma");
coll.add("Pebbles");
coll.add("Dino");
coll.add("Fred");
System.out.println("The collection: " + coll);


for (Iterator<String> it = coll.iterator(); it.hasNext();) {
    String s = it.next();
    System.out.println(s);
}
```

INHERITANCE

Example


Noisy
String sound()
String getName()
Cat
String _myName
Cat(String n)
String getName()
String sound()
Dog
String _myName
Dog(String n)
String getName()
String sound()

What's wrong with this code?

```
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";
    }
}
```

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";
    }
}
```

Number one in the stink parade is duplicated code. If you see the same code structure in more than one place, you can be sure that your program will be better if you find a way to unify them.

Refactoring: Improving the Design of Existing Code, Martin Fowler, page 76

Inheritance is the fourth relationship we will study this semester.

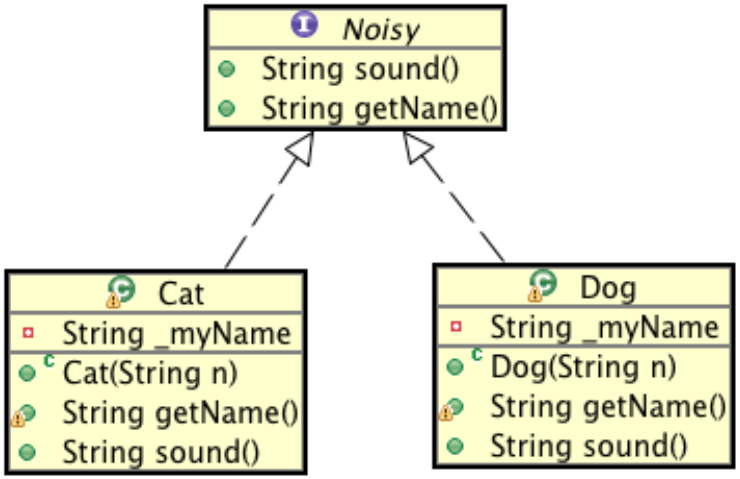
It is a relationship between:

- two classes, OR
- two interfaces.

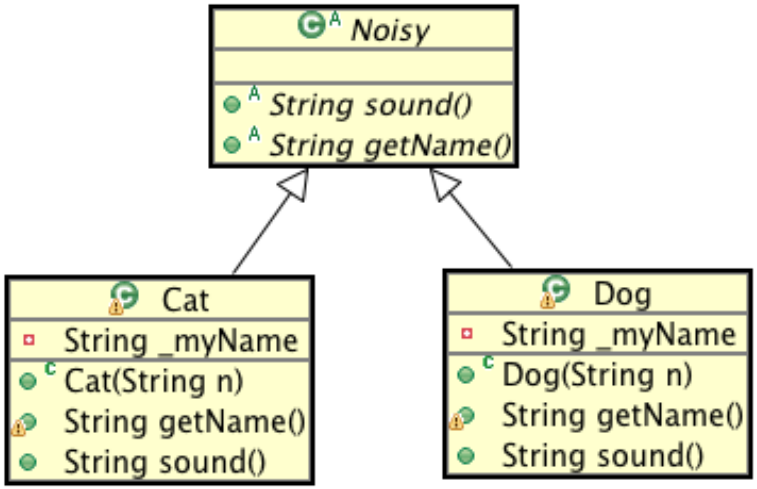
Inheritance is (syntactically) simple

Inheritance is (conceptually) messy

original design



class to class inheritance I



In code

```
public abstract class Noisy {…}

public class Cat extends Noisy {…}

public class Dog extends Noisy {…}
```

Abstract class

A class which mixes method specifications (abstract methods) with fully defined methods (concrete methods) is abstract.

An interface contains only abstract methods (they are labelled ‘abstract’ implicitly).

Abstract class

An abstract class must have the ‘abstract’ keyword in class header.

Like an interface, an abstract cannot be instantiated.

Inheritance

Source class:

- subclass
- child class
- derived class

Target class:

- superclass
- parent class
- base class

Implications of "extends"

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

- instance of subclass belongs to subclass type and superclass type

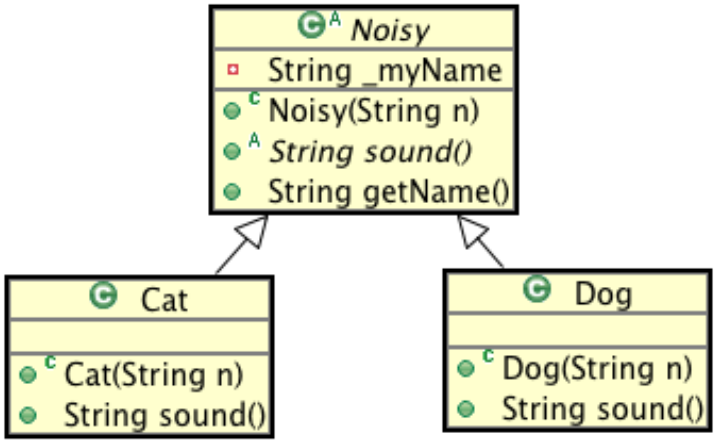
inheritance: non-private members of superclass can be accessed via subclass object.

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

[A] common duplication problem is when you have the same expression in two sibling subclasses. You can eliminate this duplication by using *Extract Method (110)* in both classes then *Pull Up Method (322)*.

Refactoring: Improving the Design of Existing Code, Martin Fowler, page 76

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 05

PAIR CODING EXERCISE

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

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

Submit to Exercise-05 in Web-CAT – enter the usernames of everyone in your group!

Submit no later than 6:00 PM tomorrow (11/5).