Objects

Discussion 4

Announcements

- Homework 2 due Tuesday, 02/08
- Project 0 due Friday, 02/11
- No Lab - Project OH (See Announcement)
- Test 1 Form due Friday, 02/11
- Exam Review Session on Friday, 02/11 from 2-4 PM in Soda 271
- Make use of gitbugs!

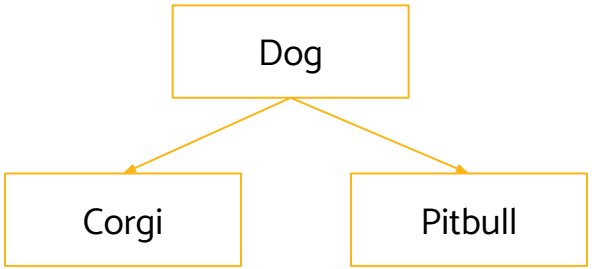
Review

Classes

Subclasses (or child classes) are classes that extend another class. This means that they have access to all of the functions and variables of their parent class in addition to any functions and variables defined in the child class.

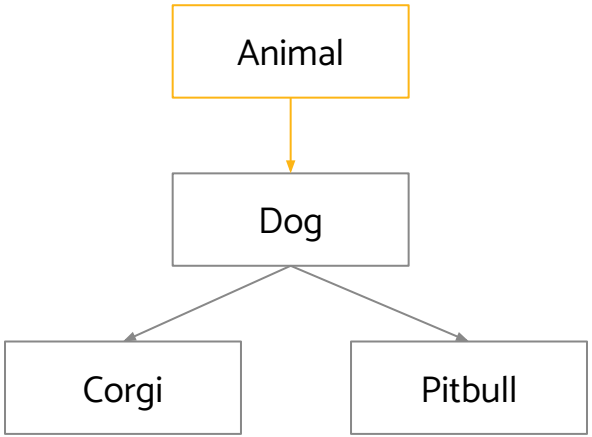
Superclasses or parent classes are classes that are extended by another class. You can use `super(...)` to call the parent constructor of a subclass.

Classes can only extend one class but can be extended by many classes.



Abstract Classes

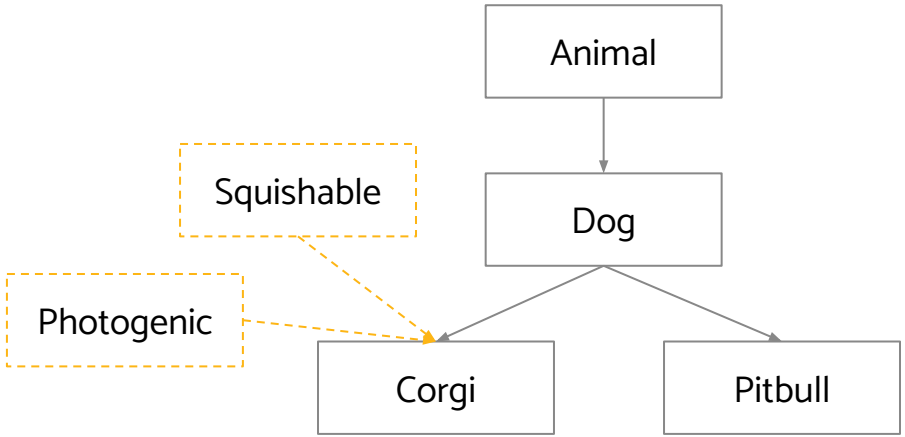
Abstract Classes are classes that cannot be directly referenced. Instead, they must be extended by a **concrete class**. They describe all of the functions that a class of their "type" must be able to do, with or without implementation.



Interfaces

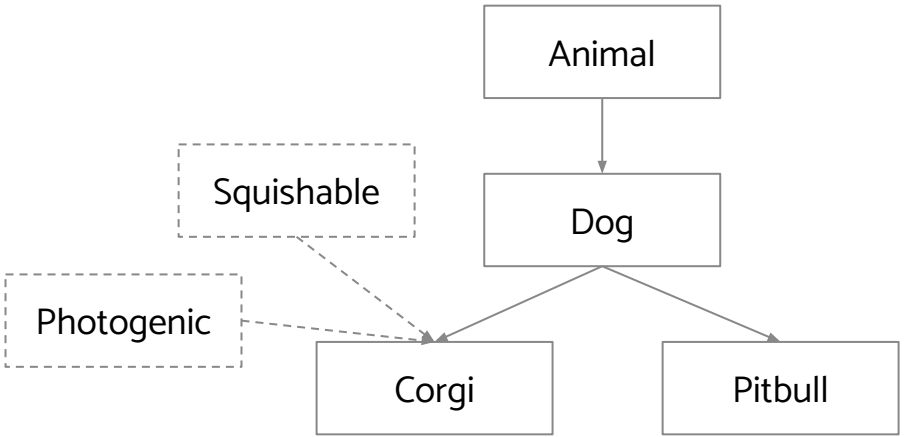
Interfaces are implemented by classes. They describe a narrow ability that can apply to many classes that may or may not be related to one another. They are like abstract classes in that they do not usually implement the methods they specify. One class can implement many interfaces.

Ex. Comparable, List



Implementation

```
abstract class Animal {...}

interface Squishable {...}

interface Photogenic {...}

class Dog extends Animal {...}

class Pitbull extends Dog {...}

class Corgi extends Dog implements Squishable, Photogenic {...}
```



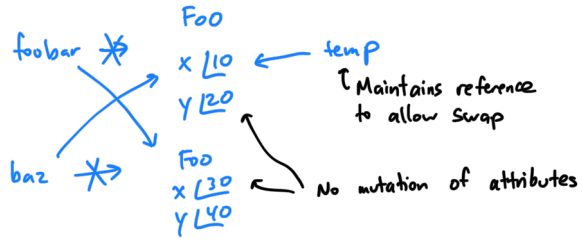
CS 61B Spring 2022 — Objects

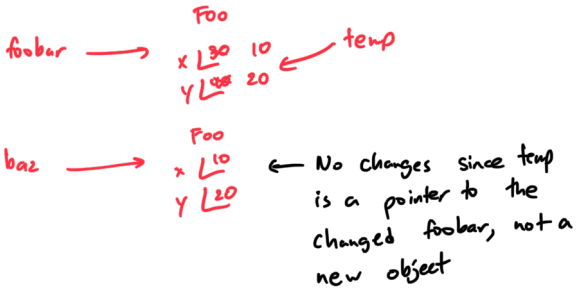
Exam Prep Discussion 4: February 7, 2022

1 Give em the 'Ol Switcheroo

For each function call in the `main` method, write out the x and y values of both foobar and baz after executing that line. (Spring '15, MT1)

```
1  public class Foo {
2      public int x, y;
3
4      public Foo (int x, int y) {
5          this.x = x;
6          this.y = y;
7      }
8      public static void switcheroo (Foo a, Foo b) {
9          Foo temp = a;
10         a = b;
11         b = temp;
12     }
13     public static void fliperoo (Foo a, Foo b) {
14         Foo temp = new Foo(a.x, a.y);
15         a.x = b.x;
16         a.y = b.y;
17         b.x = temp.x;
18         b.y = temp.y;
19     }
20     public static void swaperoo (Foo a, Foo b) {
21         Foo temp = a;
22         a.x = b.x;
23         a.y = b.y;
24         b.x = temp.x;
25         b.y = temp.y;
26     }
27
28     public static void main (String[] args) {
29         Foo foobar = new Foo(10, 20);
30         Foo baz = new Foo(30, 40);
31       1 switcheroo(foobar, baz);    foobar.x: 10 foobar.y: 20 baz.x: 30 baz.y: 40
32       2 fliperoo(foobar, baz);      foobar.x: 30 foobar.y: 40 baz.x: 10 baz.y: 20
33       3 swaperoo(foobar, baz);      foobar.x: 10 foobar.y: 20 baz.x: 10 baz.y: 20
34     }
35 }
```





2 IntList to Array

For this problem we will implement a version of `arraycopy` that copies elements from an IntList into an array of `int`s. As a reminder, here is the `arraycopy` method:

```
System.arraycopy(Object src, int sourcePos, Object dest, int destPos, int len)
```

`System.arraycopy` copies `len` elements from array `src` (starting at index `source`) to array `destArr` (starting from index `dest`).

To simplify things, let's restrict ourselves to using only `int[]`, and assume that `srcList` and `destArr` are not null. Additionally, assume that `sourcePos`, `destPos`, and `len` will not cause an IndexOutOfBoundsException to be thrown.

For example, let IntList `L` be `(1 -> 2 -> 3 -> 4 -> 5)` and `int[] arr` be an empty array of length 3. Calling `arrayCopyFromIntList(L, 1, arr, 0, 3)` will result in `arr={2, 3, 4}`.

```
/** Works just like System.arraycopy, except srcList is of type IntList. */
public static void arrayCopyFromIntList(IntList srcList, int sourcePos,
        int[] destArr, int destPos, int len) {

    for ( int i = 0 ; i < sourcePos; i += 1 ) {

        srcList = srcList.tail;
    }

    for ( int i = destPos ; i < destPos + len; i += 1 ) {

        destArr[i] = srcList.head;

        srcList = srcList.tail;

    }
}
```

Idea:

Cycle through list until desired start index. Then cycle and copy to desired end index.

3 Static Books

Suppose we have the following `Book` and `Library` classes.

```
class Book {
    public String title;
    public Library library;
    public static Book last = null;

    public Book(String name) {
        title = name;
        last = this;
        library = null;
    }

    public static String lastBookTitle() {
        return last.title;
    }
    public String getTitle() {
        return title;
    }
}
```

```
class Library {
    public Book[] books;
    public int index;
    public static int totalBooks = 0;

    public Library(int size) {
        books = new Book[size];
        index = 0;
    }

    public void addBook(Book book) {
        books[index] = book;
        index++;
        totalBooks++;
        book.library = this;
    }
}
```

(a) For each modification below, determine whether the code of the `Library` and `Book` classes will compile or error if we **only** made that modification, i.e. treat each modification independently.

1. Change the `totalBooks` variable to **non static** Compile
2. Change the `lastBookTitle` method to **non static** Compile
3. Change the `addBook` method to **static** Error - can't use "this" in static method, access books/index variables ↑ linked to an instance
4. Change the `last` variable to **non static** Error - lastBookTitle loses access
5. Change the `library` variable to **static** Compile

(b) Using the `Book` and `Library` classes from before, write the output of the `main` method below. If a line errors, put the precise reason it errors and continue execution.

```
1  public class Main {
2      public static void main(String[] args) {
3          System.out.println(Library.totalBooks);                          // 0
4          System.out.println(Book.lastBookTitle());                        // Error - Null Pointer Exception, will compile
5          System.out.println(Book.getTitle());                             // Error - instance method, won't compile
6
7          Book goneGirl = new Book("Gone Girl");
8          Book fightClub = new Book("Fight Club");
9
10         System.out.println(goneGirl.title);                              // Gone Girl
11         System.out.println(Book.lastBookTitle());                        // Fight Club
12         System.out.println(fightClub.lastBookTitle());                   // Fight Club
13         System.out.println(goneGirl.last.title);                         // Fight Club
14                                                                          // ↑ static member
15         Library libraryA = new Library(1);
16         Library libraryB = new Library(2);
17         libraryA.addBook(goneGirl);
18
19         System.out.println(libraryA.index);                              // 1
20         System.out.println(libraryA.totalBooks);                         // 1
21
22         libraryA.totalBooks = 0;
23         libraryB.addBook(fightClub);
24         libraryB.addBook(goneGirl);
25
26         System.out.println(libraryB.index);                              // 2
27         System.out.println(Library.totalBooks);                          // 2
28         System.out.println(goneGirl.library.books[0].title);             // Fight Club
29     }                                                                    // ↳ Accesses Library B
30 }
```