

## Introduction

[POGIL and application to CS loosely based on documents found at [cspogil.org](http://cspogil.org). Inspiration for some questions courtesy of ChatGPT v4.0]

This *group* assignment is focused on reading/writing extended classes in Java and using them *polymorphically*

## Assignment Goals

After completing this group assignment, each student is expected to be able to Learning Outcomes

- Define *apparent* and *actual* type.
- Differentiate between the *apparent* and *actual* type of the object a variable refers to.
- Trace function calls across a class hierarchy.

## Procedure

The group begins by *assigning roles*. The goal of the group work is for everyone to participate and the roles (part of the POGIL methodology) assign multiple group members to keep an eye on individual's engagement with the material. The four standard roles are

**Manager** Move discussion forward.

**Reflector** Monitor that everyone gets heard and is caught up. (This is a **group** obligation, really.)

**Speaker** Asks the facilitator questions and communicates what the team has done.

**Recorder** Writes the report.

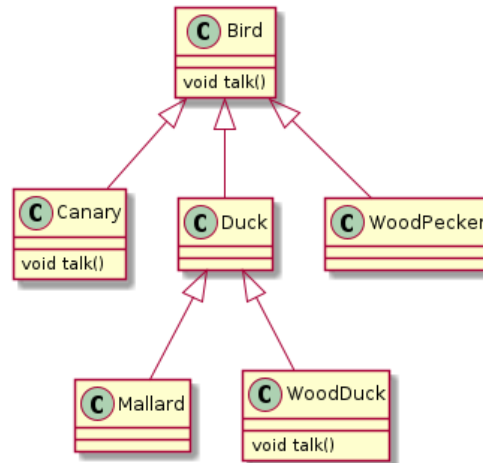
Take a role you have not recently held.

At the end of class, the group will turn in *one* record of their exploration of the questions below. The **Recorder** should make sure everyone present puts their name on top of the paper.

Before each answer, *summarize the question*. The question need not be copied verbatim but it needs to be included for the answer to make sense to a future reader.

The answers written down should be *discussed* by the group and represent the *consensus* of the group. Sometimes a *picture*, a *graph*, or something similar is the most appropriate way to answer a question.

1. *Run-time polymorphism* is accomplished with method **overriding** in subclasses. What methods are polymorphic in the given Shape hierarchy.
2. Add a `toString` method to `Rectangle` that prints its name and area. Do not call any non-`Rectangle` functions.
3. Consider the following class relationship diagram:



- (a) In the following declaration/assignment statement, identify the *apparent* type of the variable *decoy* and the *actual* type of its value at the end of the *assignment*.

```
Bird decoy = new WoodDuck();
```

- (b) After `Bird mmm = new Mallard();`, which version of `talk()` will be called with `mmm.talk();`?

- (c) With the following declaration/assignments

```

Bird b_x = new Canary();           Bird b_z = new WoodPecker();
Bird b_y = new Mallard();          Duck d_x;
Canary c_x;                        Mallard m_x new Mallard();
  
```

Explain how you know whether or not each of the following assignments will compile. (These are **independent** questions: each takes place right after the declarations above.)

- i. `c_x = b_y;`
- ii. `c_x = b_x;`
- iii. `m_x = b_y;`
- iv. `d_x = m_x;`
- v. `b_z = new Canary();`

4. Declare an *array* variable that can hold up to 73 of the any species of bird. Also declare an **int** variable to hold the actual number of elements in use in the array.
5. Assuming that there is at least one bird in your array, write the *simplest* code you can to take the bird with the **highest** index out of the array.
6. Assuming that there is at least one bird in your array, write code to take the bird with the **lowest** index out of the array.
7. Write one line of code that would appear *inside* of the Bird class, declaring a String **field** called *name* such that any class in the diagram above could read or write *name* but no other class in your project can. You only get one (1) line.