Mucking with polymorphism

1 Inheritance and hiding

In class, we discussed polymorphism, in which a the object of an sub/child-class could behave as though it were an object of the parent/super-class. In particular, we wondered what happens when a subclass overrides (i.e., redefines) an inherited method, and then is polymorphically used as though it were an object of the superclass. Which version of them method gets used?

To be more concrete: Consider a superclass Foo and a subclass Bar Assume that Foo defines an instance method named printMsg() that prints some distinct, short message. Further assume that Bar overrides this method, redefining it to print a different, distinct, short message.

If the program creates a Bar object, but accesses the object through a Foo pointer, then what happens when the printMsg() method is called on that object through that pointer?

2 Questions to answer

Create a new lab-4 directory for yourself and get started by creating the two classes described above, each in its own file named Foo.java and Bar.java respectively. Some details:

- Each class need not have any data members nor any explicitly defined constructors. (The compiler, javac, will automatically generate a default constructor for each.)
- Bar should extend Foo.
- Both of the classes should define their own printMsg() method. The only goal of this method is, when used, to make it clear to the user of the program which method was called.
- Bar can contain a main() method that creates a Bar object, but stores the pointer to it in a Foo pointer space.
- That main() method should then call the object’s printMsg() method.

Now that you have this code to work with, there are a few questions to answer:

1. What happens? Which class’s printMsg() method is used when called from main() through the Foo pointer?

---

1Henceforth, for consistency and clarity, I will try to use only the terms superclass and subclass, although the terms parent, child, ancestor, and descendent appear often in discussions of class inheritance hierarchies.

2The use of silly, placeholder names like foo, bar, baaz, quux, etc., have a long history in programming. It began with the military acronym, FUBAR, which I will leave it to you to look up.

3That is, not a static method.
2. **What if Foo defines a method that uses printMsg()?** Modify Foo to define another method, indirectPrint(), that calls on printMsg() from there. Bar should not override this method—it should just inherit it, as is. What happens is indirectPrint() is called from the main() method through the Foo pointer? Which printMsg() is used now?

3. **What if printMsg() is static?** Change printMsg() in both Foo and Bar to be static methods. What happens when printMsg() is called now, either directly from main() or indirectly through indirectPrint()?

Open a text file named answers.txt and write your answers to these questions. Then conclude, after the answers, what generalized rules you’ve inferred from that behavior.

### 3 How to submit your work

Use the CS submission systems to submit your Foo.java, Bar.java, and answers.txt files. You may use either of the following two methods, while connected to remus or romulus, to use the submission system:

- **Web-based:** Visit the submission system web page.
- **Command-line based:** Use the `~/lamcgeoch/submit` command at your shell prompt.

This assignment is due on Tuesday, Oct-18, 11:59 pm.