Inheritance

CS 61B Spring 2016: Discussion 4

Inheritance

Q. What’s the object-oriented way to get wealthy?

A. Inheritance.

Inheritance

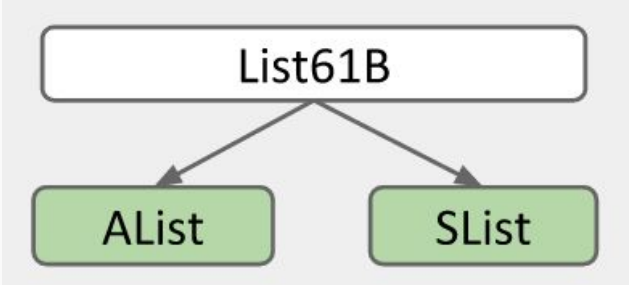
Recall SList and AList from lecture. Both are some kind of “list”.

In English:

- List61B is a **hypernym** of SList and AList.
- SList and AList are **homonym**s of List61B.

In Java (or programming in general):

- List61B is a **superclass** of SList and AList.
- SList and AList are **subclass**es of List61B.



(image borrowed from Lecture 8 Slide 18)

Overriding Methods

A **subclass** can override a method defined in the **superclass** by defining its own method with the same method signature.

We commonly will tag the method with **@Override**.

- This is optional, but usually good practice.

Recall that if X is a superclass of Y, then an X variable can hold a reference to a Y.

Which print method do you think will run when the code below executes?

- List.print()
- SList.print()

```
public static void main(String[] args) {
   List61B<String> someList = new SList<String>();
   someList.insertFront("elk");
   someList.insertFront("are");
   someList.insertFront("watching");
   someList.print();
}
```

Inheritance

All variables in Java have two types:

- **Static type (“compile-time” type)**: specified at declaration
  - e.g. List61B<String> myList;
- **Dynamic type (“run-time” type)**: specified at instantiation
  - e.g. myList = new SList<String>();

The **static type** and **dynamic type** can be the same.

Why are these two types important?

Dynamic Method Selection

If a variable has…

Static type X

Dynamic type Y

and

Y overrides a method defined in X (this also implies Y is a subclass of X)

then

Y’s version of the method is called instead of X’s version

Dynamic Method Selection

Dynamic method selection happens only for **overridden methods**.

It does **not happen** for **overloaded methods**. In this case, the static type is still used to decide which method to use.

```
public static String olpuzzle(List61B<String> list) {
  return list.getBack();
}
public static String olpuzzle(SList<String> list) {
  return list.getFront();
}
```

(image borrowed from Lecture 8 Slide 38)

Now, let’s start with Problem 1!