Objects and Arrays

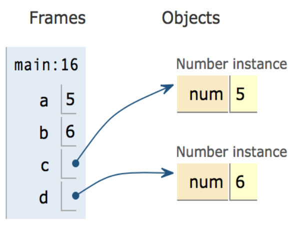
Abhinav Ashar
CS 61B: Data Structures and Algorithms

February 16, 2019

1 Objects

As opposed to the primitive types that you may have learned in class (int, double, float etc.), an object is an instance of a class that often operates as the combination of multiple pieces of data at once. For example, you are not defined by only your name. You are also defined by your age, height, hair color, shoe size etc. The combination of all these attributes come together to describe you. As a result, if we were to describe your attributes in code, we know it would be best to have an object because that combines multiple pieces of data into one entity.

a) Primitive types store values while objects store pointers.
Suppose a, b are primitive integers and c, d are Number objects.

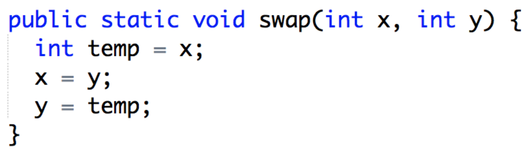


Notice that a,b are storing values while c,d are storing pointers.

b) Pointers are represented in the computer as some hexadecimal number like 0xFFFFE147. Don't get too hung up on that, however. What matters for this class is that if two objects hold the same value (i.e. the same hexadecimal address number), then they are pointing to the exact same object. Thus, if one variable modifies something about that object, the other variable pointing to it will know about the change as well.

c) **Golden Rule of Equals:** When we pass variables into the arguments of methods, we pass by value. This means that if we send a primitive variable, we will send the value of the primitive variable. If we send an object, we will send the value of the object (which is actually some address in memory, not the value of a variable in the object). This can lead to some confusing results.

Let's define the swap function:

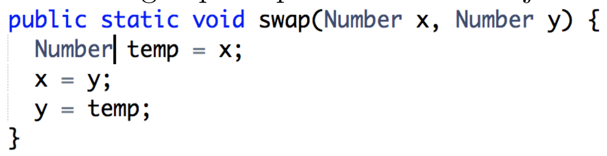


```
public static void swap(int x, int y) {
  int temp = x;
  x = y;
  y = temp;
}
```

This swap function looks like it will take in two integers, and swap their values. However, that is not what it actually does. If we call $swap(a, b)$ using a,b from the image in (a), then a will **pass its value** (5) to x and b will **pass its value** (6) to y. Thus, $x = 5$ and $y = 6$. Then, when we run the code, $x = 6$ and $y = 5$. However, when the function ends, the frame is closed and these variables disappear, and a,b are left unchanged.

What about with objects?

Let's change up swap so it takes in objects instead.

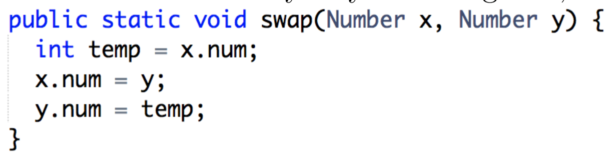


```
public static void swap(Number x, Number y) {
  Number temp = x;
  x = y;
  y = temp;
}
```

Let's try this new swap function instead. If we call $swap(c, d)$ using c,d form the image in (a), then c will **pass its value** (some unknown address number) to x and d will **pass its value** (some unknown address number) to y. Thus, when we run the swap function, x and y will swap their address numbers so x now points what y used to point to (and vice versa). However, when this function ends, the frame is closed and these variables disappear. So once again, c,d are left unchanged.

So how can we get the swap function to swap two numbers?

While there are many ways of doing this, here is one of them:



```
public static void swap(Number x, Number y) {
  int temp = x.num;
  x.num = y;
  y.num = temp;
}
```

Let's try this one last time. If we call $swap(c, d)$, then c will **pass its value** (address) to x and d will **pass its value** (address) to y. Now we have c,x pointing at the first number instance in (a), and we have d,y pointing at the second number instance in (a). Note: *num* is an attribute variable of the Number class that I have made. What this swap function does is that it stores the value of num from x in temp, modifies x's num, and modifies y's num. When the function ends, the frame is closed and x,y disappear. Are we running into the same problem as before? Remember what I said in (b)! Since c,x were pointing to the same object, and d,y were pointing to the same object, c and d knew about the changes that x and y made. Therefore, the values of num in c and d actually swapped!

2 Arrays

Arrays are like lists from Python, but have a few exceptions.

a) Arrays have a defined, fixed length.

b) Arrays cannot change their length.

c) Arrays must be declared with a type. For example, it needs to be an integer array or a String array et. These arrays can only contain that one type of primitive value or object. For example, an array cannot contain a string, integer, and decimal all together.

d) Arrays have a .length property that lets you access the length of the array.

e) Arrays are zero-indexed, meaning they go from array[0] to array[array.length-1].

f) Arrays start with default values based on the type. If the type is an object, the indices start with **null**. If the type if a boolean, the indices start with **false**. If the type is a primitive number, the indices start with **0**.

g) To iterate through an array, you often use a for loop. In addition, you can use a for each loop.
Example:
```
for (int x: numberArray)
```
This means that for each index in numberArray, x will assume the value of the contents in that index and run some code.

3 Inheritance

As discussed in lecture, classes can inherit properties from other classes. This is often used because it is organized, efficient, and makes logical sense for one class to be a subclass of another, more overarching class. For example, it makes sense for Poodle to a subclass of Dog because a poodle is a type of dog. Thus, a Poodle object will get access to all the methods of the Dog class.

a) A subclass can do whatever a parent class can do, in addition to more specific things

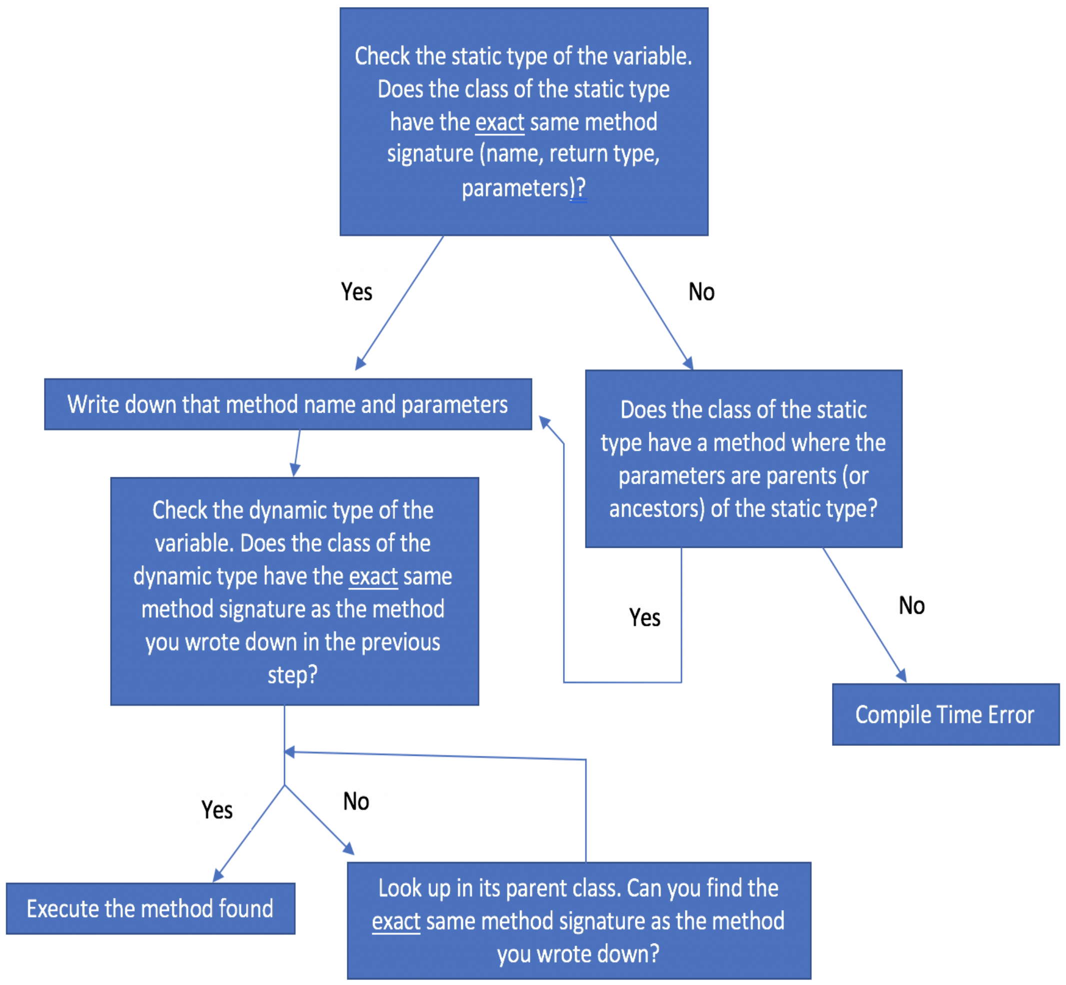
b) **Is-A/Has-A Relationship:** If X is a subclass of Y, then Y is a X, and X has a Y.

c) **Overriding:** A subclass method that has the exact same signature as a parent class method may be executed in runtime instead of the parent class method.

d) **Overloading:** Different methods can have the same number, but they have different signatures (different input parameters or different number of input parameters).

e) If we have the following object creation: X num = new Y() (Note: X is a parent of Y), then X is the static/compile-time type and Y is the dyname/run-time type.

f) One of the most challenging topics in 61B is when we are trying to figure out which method to run relative to static and dynamic types. I recommend that you follow the following chart:


Check the static type of the variable. Does the class of the static type have the exact same method signature (name, return type, parameters)?
Yes
No
Write down that method name and parameters
Does the class of the static type have a method where the parameters are parents (or ancestors) of the static type?
Yes
No
Compile Time Error
Check the dynamic type of the variable. Does the class of the dynamic type have the exact same method signature as the method you wrote down in the previous step?
Yes
No
Execute the method found
Look up in its parent class. Can you find the exact same method signature as the method you wrote down?

If you follow this chart algorithmically, then you should be able to tackle any problem!