

CS231 Fall 2013 Lecture 4

Stephanie R Taylor

September 11, 2013

1 Administrative Topics

- Did everyone get my email with Monday's notes? If not, let me know, because maybe you aren't on my email list.
- I am going to start posting notes before lecture. That means there may be more material in the notes than we cover in class. I will still post the code afterwards, because we make that up together.

2 Java's Memory Model

- All variables are allocated space in memory to store something.
- Primitives vs objects
 - All primitive variables store their value in that space.
 - All reference variables store a pointer in that space to the object they refer to.
- Local variables vs fields
 - There are two kinds of variables: fields and local variables/parameters
 - Every method has a stack frame with space for all its local vars (while it is being executed).

- Fields reside in the objects on the heap.
- There are two kinds of variables: instance variables/fields and local variables/parameters
- Every method has a stack frame with space for all its local vars.
- Fields reside in the objects on the heap.
- Example: `main()` method for the `Die` class

```

public static void main( String [] args ) {
    Die die1 = new Die ();
    Die die2 = new Die ();
    die1.roll ();
    die2.roll ();
    System.out.println( "The dice are " + die1.getSideUp() + " and " + die2.getSideUp() );
}

```

- The stack frame for `main` contains entries for `die1` and `die2`, each of which are pointers.
- What would happen if we added the line `die1 = die2;`? Both variables will point to the same `Die` object and the original `Die` object that `die2` pointed to is garbage collected.
- Local variables' lifetimes consist of the time the method that contains them starts until the method returns.
- Parameters are identical to local variables in Java except that they are initialized by the caller instead of in the method body.
- Field's lifetimes consist of the time from the creation of the object until the garbage collection of the object.
- Local variable's scope is from the point of declaration to the end of the enclosing scope.
- Field's scope depends on whether they are public or private. The scope of private fields is the whole class body in which the fields are declared. The scope of public fields is everywhere.

- Here is a table indicating the location, lifetime, and scope of variables, depending on where they are declared:

	local var/params	instance var/field
Location	stack frame	heap
Lifetime	during method execution	object creation to garbage
Scope	remaining body of method	class if private, all if public

- Recall that the new command allocates memory for a new object, including space for all the fields of the object, clears out that memory, calls the constructor to initialize the fields, and then returns a pointer to the memory.
- You can have more than one constructor, as long as they differ on the number or types of the parameters.
- Method calls' parameters are passed by value
 - Primitive values are copied into the parameters
 - Objects are not copied. Instead the pointers to the objects are copied into the parameters.
 - Demo with the add() method in the Money class (using our main method)
- Methods that return a value return the actual value if primitive and return a pointer to the object if it is not primitive.
- Fields are initialized to default values when an object is constructed. What if one field was of type String? Can you guess the default value?
- You can initialize any non-primitive (e.g., Die or Money) variable to null.