CSCI 136
Data Structures &
Advanced Programming

## Memory, Objects, and Primitive Types

This Video

- Clear up some important points about Java:
  - Where are objects stored?
  - What distinguishes objects and primitive types?
  - When do values change?
  - How to move data around in Java

- Some stuff we've talked about
- Some stuff you've probably seen while coding
- Some new stuff

Basics of Variables

- Variables store information

- Behind the scenes: all of the local variables in each method are stored next to each other in memory

```
int x;
int y;
int z;
```

x   y   z
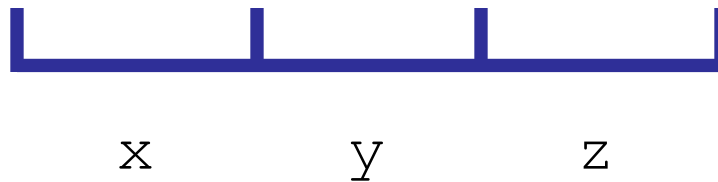
What is stored?

- For primitive types: exactly what you'd expect
- `int` stores a binary number equal to the integer
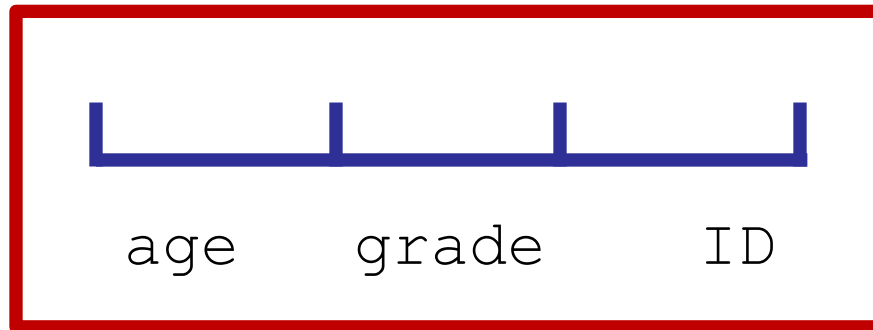- `char` stores the Unicode value for the string in binary

```
int x=10;
```

00000000000000000000000000001010

x

How to store objects

- When you use new, Java finds some unused memory (anywhere---not necessarily near any local variables) to store the object
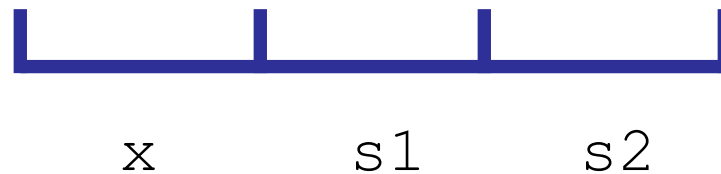- Needs to have room for all instance variables, etc.

new Student()

age   grade   ID
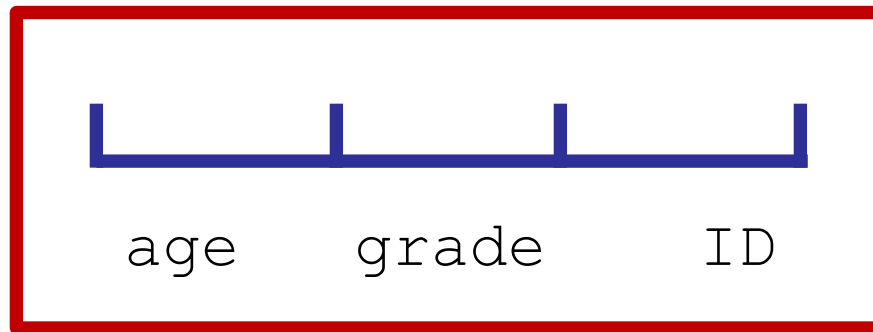
What happens when you store an object

- You really just store the "address" of where the actual object is

```
int x;
Student s1;
Student s2;
```

```
|___|___|___|___|
  x    s1    s2
```

What happens when you store an object

- You really just store the "address" of where the actual object is

age   grade   ID

```
int x;
Student s1;
Student s2;
s1 = new Student();
```
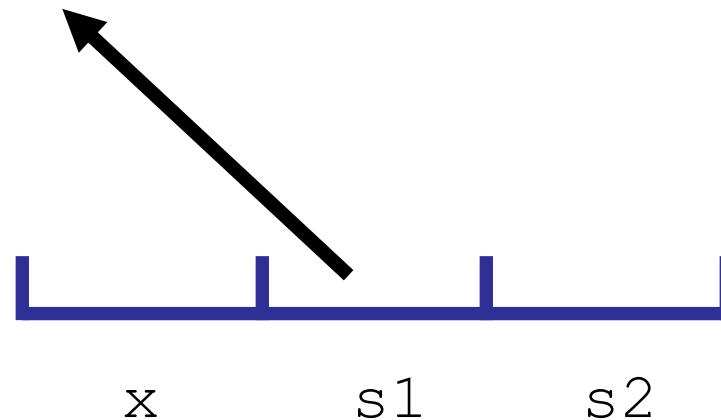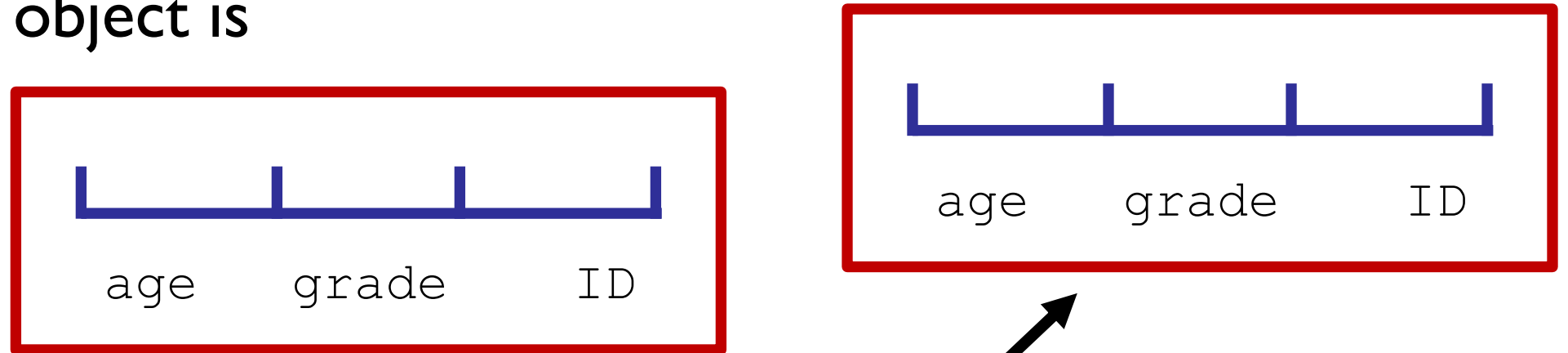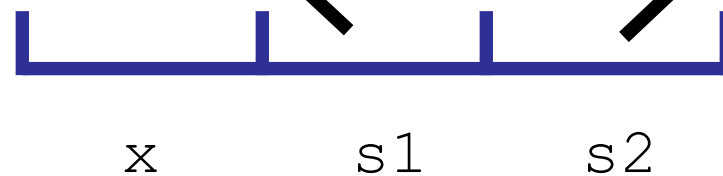
x   s1   s2

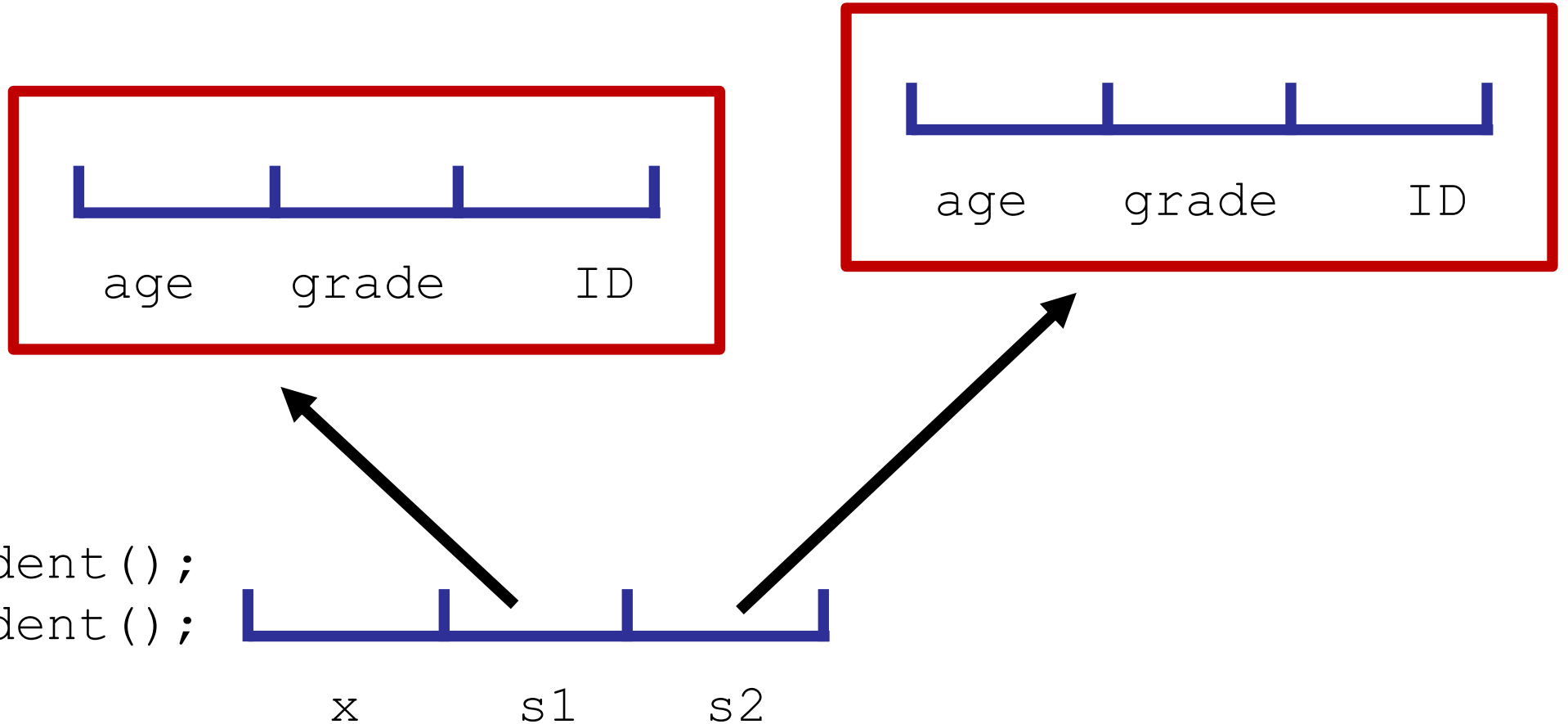# What happens when you store an object

- You really just store the "address" of where the actual object is


```
int x;
Student s1;
Student s2;
s1 = new Student();
s2 = new Student();
```
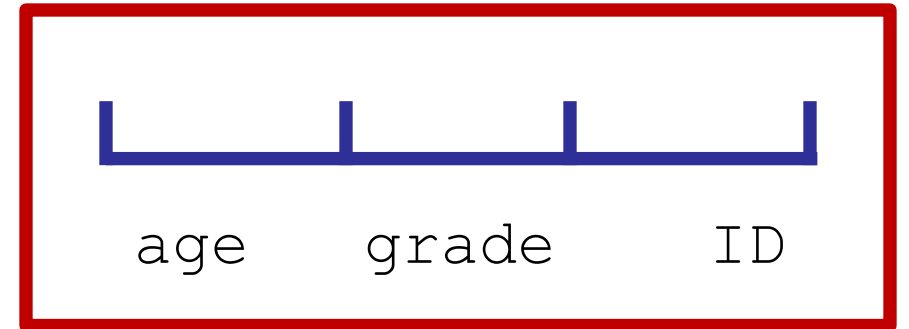
Some implications

age   grade   ID

age   grade   ID

```
int x;
Student s1;
Student s2;
s1 = new Student();
s2 = new Student();
s2 = s1;
```

x   s1   s2

Some implications

age   grade   ID

age   grade   ID

```
int x;
Student s1;
Student s2;
s1 = new Student();
s2 = new Student();
s2 = s1;
```

x   s1   s2
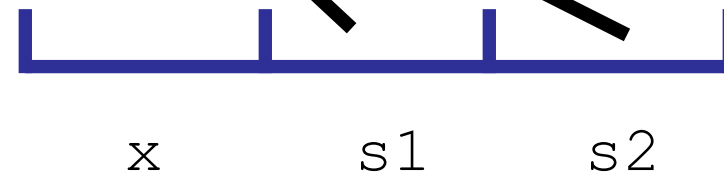
Some implications

- Any changes made to s2 will affect s1 and vice versa
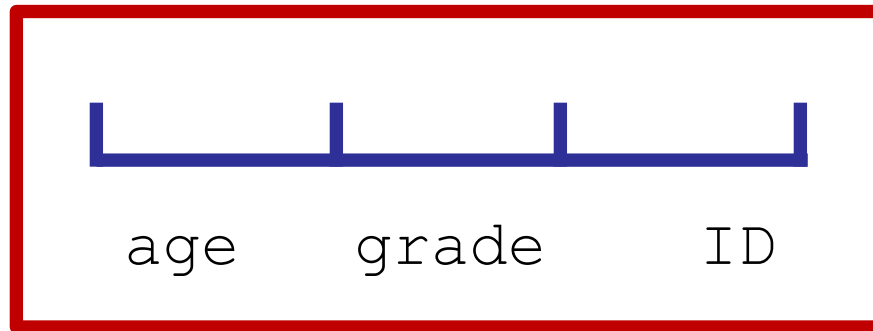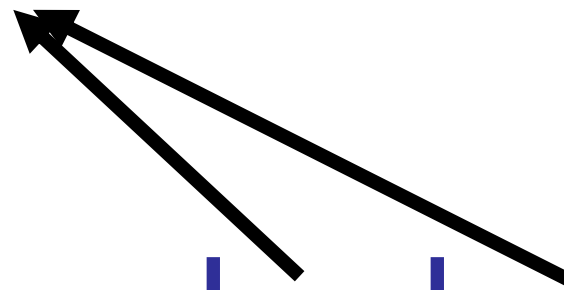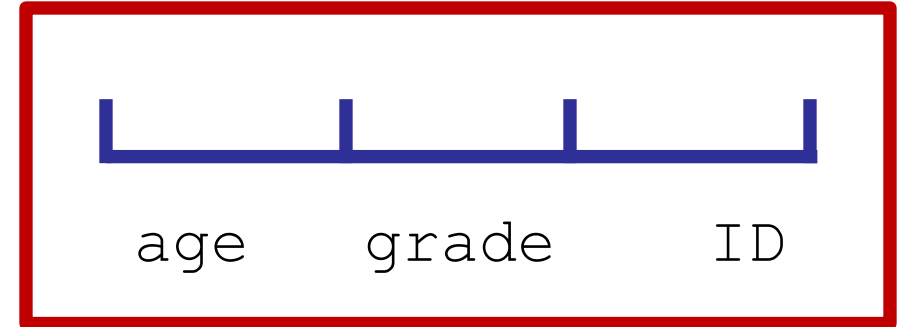- The former s2 will be (eventually) deleted

age   grade   ID

age   grade   ID

```
int x;
Student s1;
Student s2;
s1 = new Student();
s2 = new Student();
s2 = s1;
```

x   s1   s2
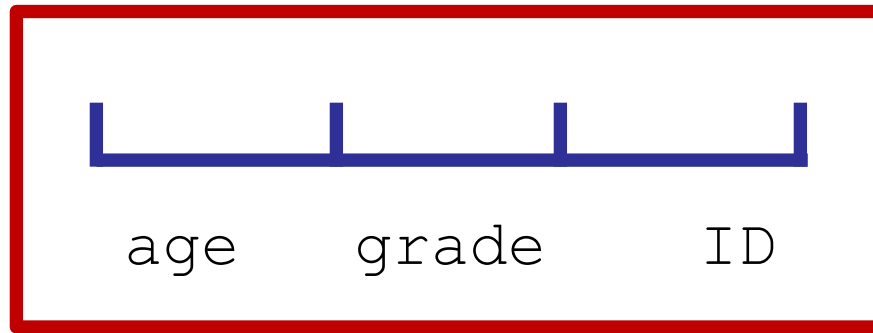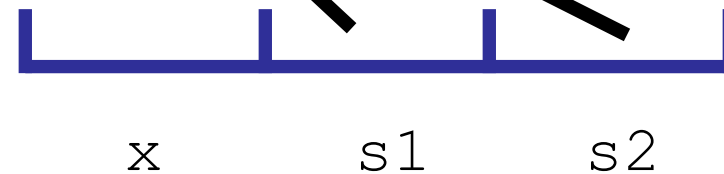
Copy

- Sometimes: want to actually make a new copy of an object
- Need to make a new one (using `new` and calling a constructor)
- Some classes have a "copy constructor," which take an object of the same type as argument and copy it over

Copy: primitive types

- Primitive types always just copy over the value

```
int x = 10;
int y = 20;
y = x;
y++;
```

After all this, `y` stores 11 and `x` stores 10

Function arguments

- All arguments to functions are passed *by value*

- This means that any changes to variables are not reflected in the original function

Arguments with objects

- Objects are passed the same way

- But, it's the *location* that must remain unchanged

- You can change the contents of objects in a function

- But you cannot change which object it is

- Let's see an example

Cleaning up old data

- When are objects deleted?
- Can't use scope
  - Could be "pointed to" from another method

- Answer: Garbage collector
  - Every once in awhile, Java looks at everything you're storing in memory. If you're not pointing to an object anymore, it's deleted

Cleaning up old data

age   grade   ID

age   grade   ID
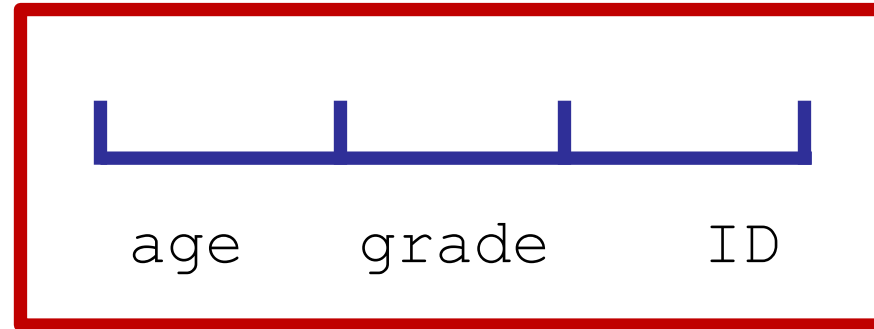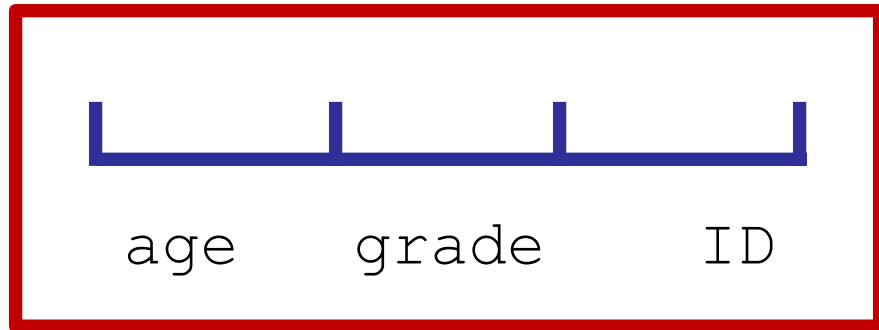
```
int x;
Student s1;
Student s2;
s1 = new Student();
s2 = new Student();
s2 = s1;
```

Cleaning up old data
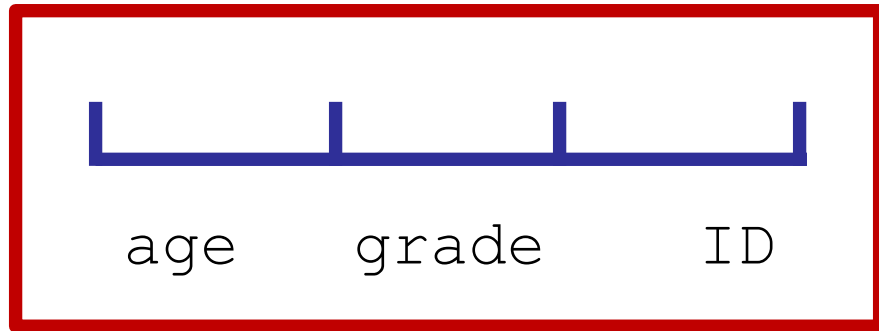

```
int x;
Student s1;
Student s2;
s1 = new Student();
s2 = new Student();
s2 = s1;
s1 = null;
```

Cleaning up old data

- Garbage collection runs automatically
- You don't need to think about it!
  - If you aren't using it, it will be deleted
  - If you are using it, Java won't delete it

- Only comes up with space usage
  - Your program will only clear out space if you stop keeping track of it

Autoboxing

- Sometimes we really want primitive types to be treated as objects

- Otherwise we can't have a Vector of `int`s, or an Association of `int`s (annoying!)

- Java has a tool to help us out with this

Autoboxing

- Java converts int to Integer, char to Character, etc., automatically

- Your vector really does store objects of type Integer. But it's ok to do something like:

```
Vector<Integer> vec = new Vector<Integer>;
vec.add(10);
```

Unboxing

- Can do the opposite too!

```
Vector<Integer> vec = new Vector<Integer>;
vec.add(new Integer(10));
int x = vec.get(0);
```

SCOPE

This Video

- How long do local variables last in Java?
- When can they be accessed?

- Not talking about instance variables/objects-
  --we already went over how long those last

Methods

- Any variable declared in a method only lasts until the end of the method

Loops/if statements/etc.

- Any variable declared in a loop (or an if statement, etc.) only lasts until the end of *that loop*

Slightly more technical outlook

- Local variables only last inside the curly braces in which they were created

- Even if you add in extra braces
  - Unclear why you'd want to?
  - But worth bearing in mind:
  - Variables cannot be accessed after the {} they are in is closed

Takeaways

- Objects are "pointed to" rather than being stored inline
- Take care when copying objects
- But, helpful when passing arguments to functions since changes to instance variables persist
- Autoboxing and garbage collection help us out in the background
- Keep an eye out for scope!