Constructors and Instantiation (OCA Objectives 6.5 and 7.5)

- A constructor is always invoked when a new object is created.
- Each superclass in an object's inheritance tree will have a constructor called.
- Every class, even an abstract class, has at least one constructor.
- Constructors must have the same name as the class.
- Constructors don't have a return type. If you see code with a return type, it's a method with the same name as the class; it's not a constructor.
- Typical constructor execution occurs as follows:
  - The constructor calls its superclass constructor, which calls its superclass constructor, and so on all the way up to the `Object` constructor.
  - The `Object` constructor executes and then returns to the calling constructor, which runs to completion and then returns to its calling constructor, and so on back down to the completion of the constructor of the actual instance being created.
- Constructors can use any access modifier (even `private`!).
- The compiler will create a default constructor if you don't create any constructors in your class.
- The default constructor is a no-arg constructor with a no-arg call to `super()`.
- The first statement of every constructor must be a call either to `this()` (an overloaded constructor) or to `super()`.
- The compiler will add a call to `super()` unless you have already put in a call to `this()` or `super()`.
- Instance members are accessible only after the `super` constructor runs.
- `Abstract` classes have constructors that are called when a concrete subclass is instantiated.
- Interfaces do not have constructors.
- If your superclass does not have a no-arg constructor, you must create a constructor and insert a call to `super()` with arguments matching those of the superclass constructor.
- Constructors are never inherited; thus they cannot be overridden.
- A constructor can be directly invoked only by another constructor (using a call to `super()` or `this()`).
- Regarding issues with calls to `this()`:
  - They may appear only as the first statement in a constructor.
  - The argument list determines which overloaded constructor is called.
  - Constructors can call constructors, and so on, but sooner or later one of them better call `super()` or the stack will explode.
  - Calls to `this()` and `super()` cannot be in the same constructor. You can have one or the other, but never both.

Initialization Blocks (OCA Objective 6.5-ish)

- Use `static init` blocks—`static { /* code here */ }`—for code you want to have run once, when the class is first loaded. Multiple blocks run from the top down.
- Use normal `init` blocks—`{ /* code here }`—for code you want to have run for every new instance, right after all the super constructors have run. Again, multiple blocks run from the top of the class down.

Statics (OCA Objective 6.2)

- Use `static` methods to implement behaviors that are not affected by the state of any instances.
- Use `static` variables to hold data that is class specific as opposed to instance specific—there will be only one copy of a `static` variable.
- All `static` members belong to the class, not to any instance.
- A `static` method can't access an instance variable directly.
- Use the dot operator to access `static` members, but remember that using a reference variable with the dot operator is really a syntax trick, and the compiler will substitute the class name for the reference variable; for instance:

```
d.doStuff();
```

becomes

```
Dog.doStuff();
```

- `static` methods can't be overridden, but they can be redefined.