Questions

Can we spend more than 10 minutes on these?

- yes!

How should we complete these code-writing questions? Typing in the text box is a pain.

- however you want
  - writing the code in Eclipse and pasting it in is fine, but a complete working program is not an obligation

Questions

How do I proceed if I don't understand what to do?

- identify as specifically as possible what you are stuck on
  - don't understand the question or task? don't know how to get started? unfamiliar terminology? don't know the syntax? …
- for class prep exercises, that's enough
- in general, take steps to address that specific thing
  - about the problem or task → office hours or email
  - how to get started → office hours or TFs
  - something talked about in class → posted slides/examples and/or textbook, then office hours or TFs
  - …

Questions

I think I may be writing too much. This isn't as efficient as it could be.

- getting something written down first is valuable
  - easier to analyze / think about something concrete on the page
- what do you think is unnecessary or could be improved?
  - can you leave it out? what would happen?
  - can you think of possible alternatives? would one of those work instead?
- discuss in office hours / TFs

Using Objects

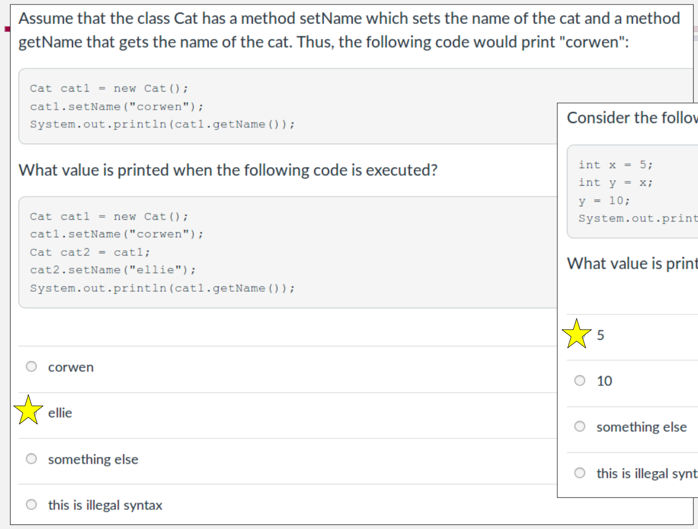
Three steps –

- declare a variable to hold a reference to the object
- create the object itself (using a *constructor*)
- use the object by invoking methods on it

Find out how to use the constructor and what methods are available using the API documentation for the class (or looking at the public comments and method headers in the class itself).

Variables vs Objects

- variables hold references to objects but there is not necessarily a one-to-one correspondence between them
- you can create a new variable without creating a new object
  - do this when the object that the variable should refer to already exists
- you can have several variables referring to the same object
- you can have variables that don't refer to an object
  - value is `null` (not the same as an uninitialized variable)
  - calling a method on a `null`-valued variable is illegal – results in a `NullPointerException` (runtime error)



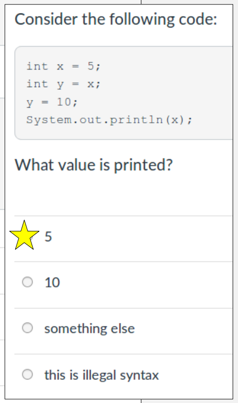
Assume that the class Cat has a method setName which sets the name of the cat and a method getName that gets the name of the cat. Thus, the following code would print "corwen":

```
Cat cat1 = new Cat();
cat1.setName("corwen");
System.out.println(cat1.getName());
```

What value is printed when the following code is executed?

```
Cat cat1 = new Cat();
cat1.setName("corwen");
Cat cat2 = cat1;
cat2.setName("ellie");
System.out.println(cat1.getName());
```

- ○ corwen
- [✓ ellie]
- ○ something else
- ○ this is illegal syntax



Consider the following code:

```
int x = 5;
int y = x;
y = 10;
System.out.println(x);
```

What value is printed?

- [✓ 5]
- ○ 10
- ○ something else
- ○ this is illegal syntax

== vs `equals(obj)`

- *a* `==` *b* compares the value of expression *a* and the value of expression *b*
  - for variables, the value is what is in the box
  - for variables that aren't primitive types (i.e. arrays or objects), the address of the thing rather than the thing itself is in the box
    - == tells you whether the addresses are the same i.e. whether a and b refer to the *same object*
- sometimes you want a notion of *equivalence* instead
  - implemented with the `equals(obj)`
  - see the API for the object's class to determine what “equivalence” means for that class
    - if listed in “Method Summary”, the description should say
    - if listed in “Methods inherited from class” something other than `java.lang.Object`, follow that link and the description should say
    - if only listed in “Methods inherited from class `java.lang.Object`”, it is “same object” (the same as ==)

Printing Objects

```
System.out.println(“the object: “+obj);
System.out.println(obj);
```

- when an object is used in a context where a `String` is expected, the system automatically treats this as if the object's `toString()` method is being called

```
System.out.println(“the object: “+obj.toString());
System.out.println(obj.toString());
```

- see the API for the object's class to determine what `toString()` does
  - if listed in “Method Summary”, the description should say
  - if listed in “Methods inherited from class” something other than `java.lang.Object`, follow that link and the description should say
  - if only listed in “Methods inherited from class `java.lang.Object`”, will get a default of the form type@address

Arrays of Objects

- the steps are the same as for arrays of other types
  - declare a variable for the array
  - create the compartments
  - initialize each compartment

Garbage Collection

We create new objects (and arrays) with new but never throw them away.

- some languages require the programmer to explicitly *deallocate* objects when they are no longer used
- Java has a *garbage collector* which automatically detects and deallocates objects that are no longer in use