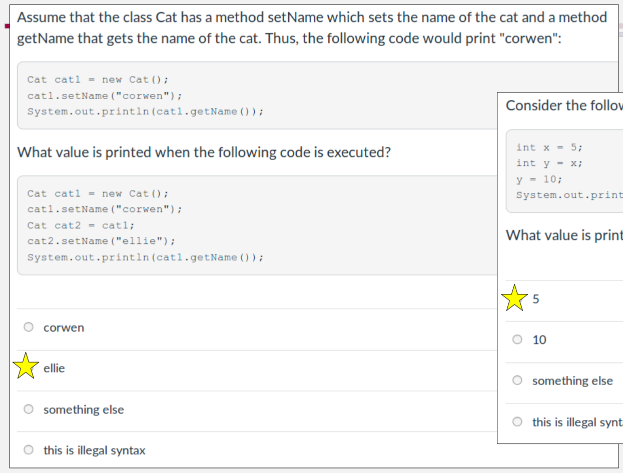
Variables vs Objects

- 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)



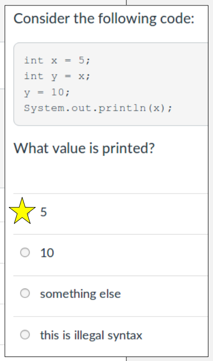
Assume that the class Cat has a method setName which sets the name of the cat and a method getName that gets the name of the cat. Thus, the following code would print "corwen":

```
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