CS61B DISCUSSION 2

TA: SHERDIL NIYAZ

Updates:

- I’m back for lab this week! My office hours are also back on 5-6 PM in Josh Hug’s office.
- You have a project due this Friday! We’re here to help!
- Extra Problem Set 2 isn’t out yet, but should be by the end of the week…sorry for the hold up.
- Extra reading from Professor Shewchuck’s CS61B that will be helpful in learning about static and classes: https://www.cs.berkeley.edu/~jrs/61b/lec/03

A note on bits:

- Every variable in java is stored as bits under the hood.
- Example: when you make a variable x that stores an int, Java stores bits that represent the number.

  int x = 7

  Java sees:

  x —-> 111
- In the case of **primitive** types, the bits **literally store what the variable is** (i.e for byte, short, int, long, float, double, boolean, char).
- What about when you aren't dealing with a primitive type?

Non primitive types

- When you make an object that is not primitive (literally any type **except** the 8 on the previous slide), you can't literally store what the object is. The object is created in a **far off** place, and then the bits in the variable act as an **address** to let the java access that object later. **This is called pass by reference.** Address is a synonym for reference!

  Walrus x = new Walrus();

  Java sees:

  x —> 010101011111111

  where the bits represent the address of the Walrus object.

- Pictorially, you can show this as an arrow pointing to the Walrus object in a **box and pointer** diagram.

- Example

COULD YOU READ THAT LAST SLIDE?

Static Variables

- When a variable in a class is static, there is **one** (and only one!) for that class, and **every instance of that class!** If you have a class Dog and an instance of that class (sparky), Dog might have a static variable num_dogs.

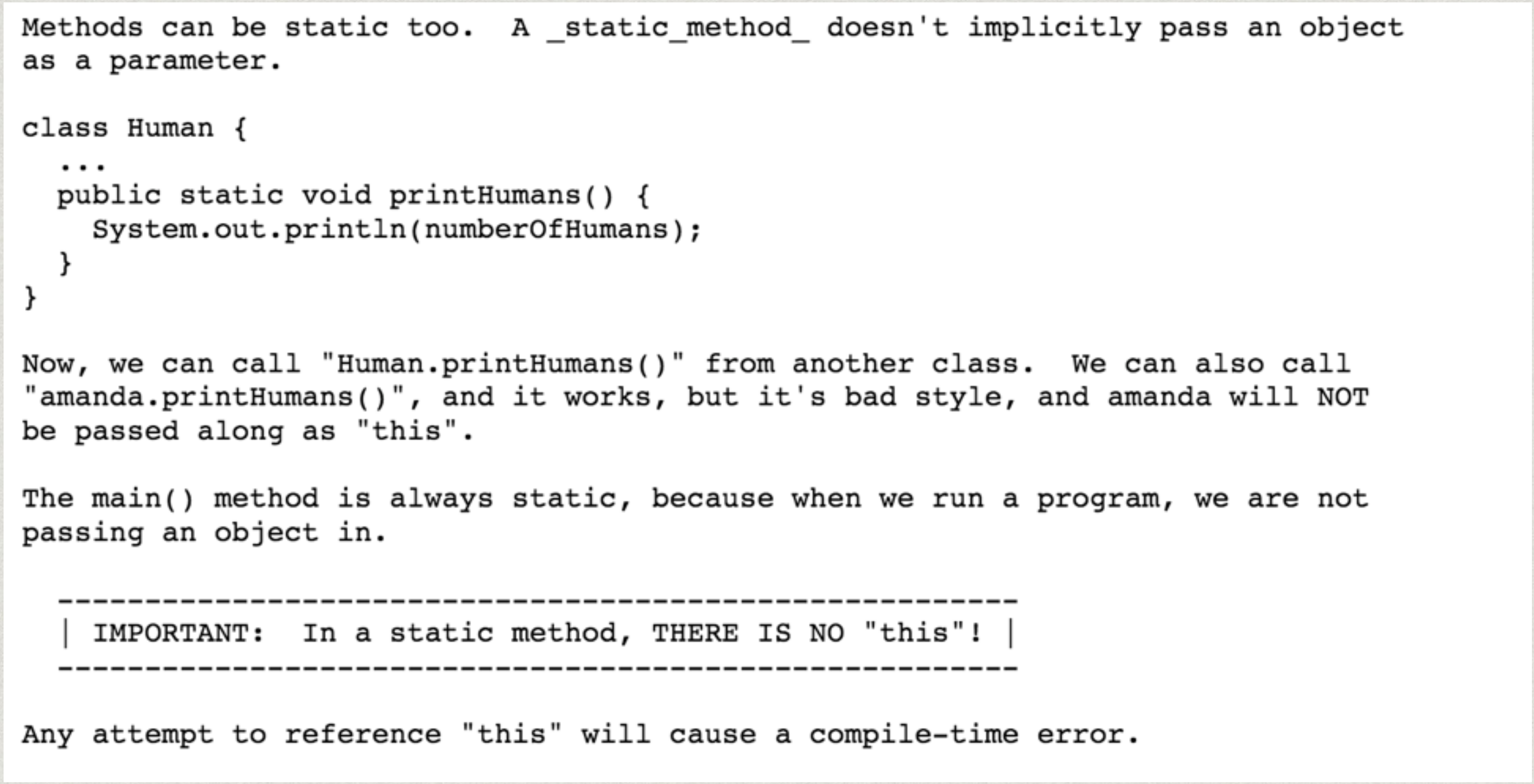
  Dog Sparky = new Dog();
  int x = Dog.num_dogs// Groovy
  x = Sparky.num_dogs // Refers to exact same thing.

  The previous line is bad style though- num_dogs isn’t a variable specific to Sparky

THIS IS DIFFERENT FROM 61A! BE WARNED!

STATIC METHODS != STATIC VARIABLES

Static Methods (Stolen from JRS)



Methods can be static too. A _static_method_ doesn't implicitly pass an object as a parameter.

```
class Human {
  ...
  public static void printHumans() {
    System.out.println(numberOfHumans);
  }
}
```

Now, we can call "Human.printHumans()" from another class. We can also call "amanda.printHumans()", and it works, but it's bad style, and amanda will NOT be passed along as "this".

The main() method is always static, because when we run a program, we are not passing an object in.

```
  ------------------------------------------------------
  | IMPORTANT:  In a static method, THERE IS NO "this"! |
  ------------------------------------------------------
```

Any attempt to reference "this" will cause a compile-time error.

Analogy: def from Python (done outside a class)

QUESTIONS?