RE-BOOTCAMP

Classes and Methods

CLASSES & OBJECTS

- Classes act as a blueprint of objects, which are real world representations of objects, in programming
- Each object is referred to as an instance of the class it is created from
- Everything in java is written in a **.java** file which will be compiled to a **.class** file
  - Note: The java file and class file should have the same name
- Objects are created by using **new** keyword in the following format:
  - ClassName referenceName = new ClassName()

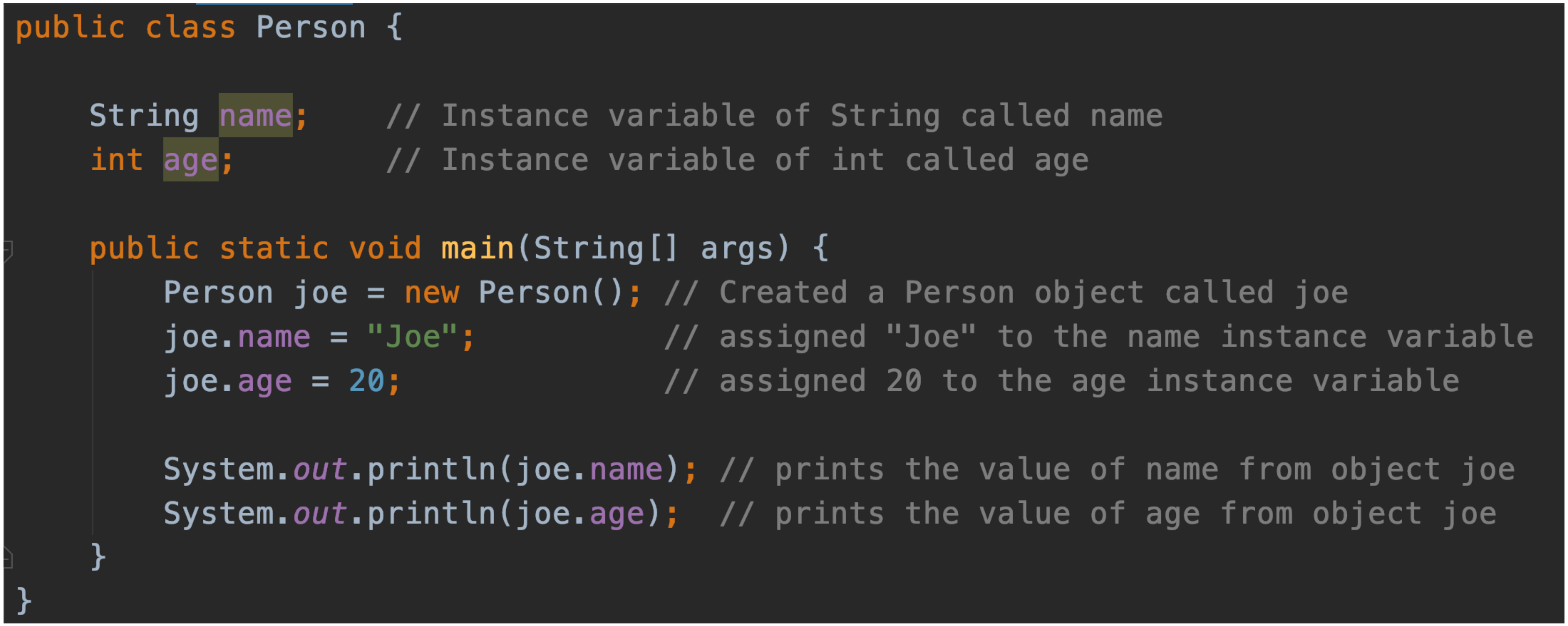
THIS

- The **this** keyword is used in a class to refer to the receiving object
- It can be used to refer to the instance variables of an object or the objects itself
- this() can be used to refer to the objects' constructors
    - If this() is called in an overloaded constructor, it must be the first line

CLASS MEMBERS - INSTANCE VARIABLES

- Instance variables are the data items inside of classes
  - These variables have datatypes as any other variable
- Every instance, which refers to each object, created from a class will have those instance variables
- These variables have default values
  - Numbers default to 0, Objects default to null, Booleans default to false
- Objects can refer to their instance variables with the dot " . " operator

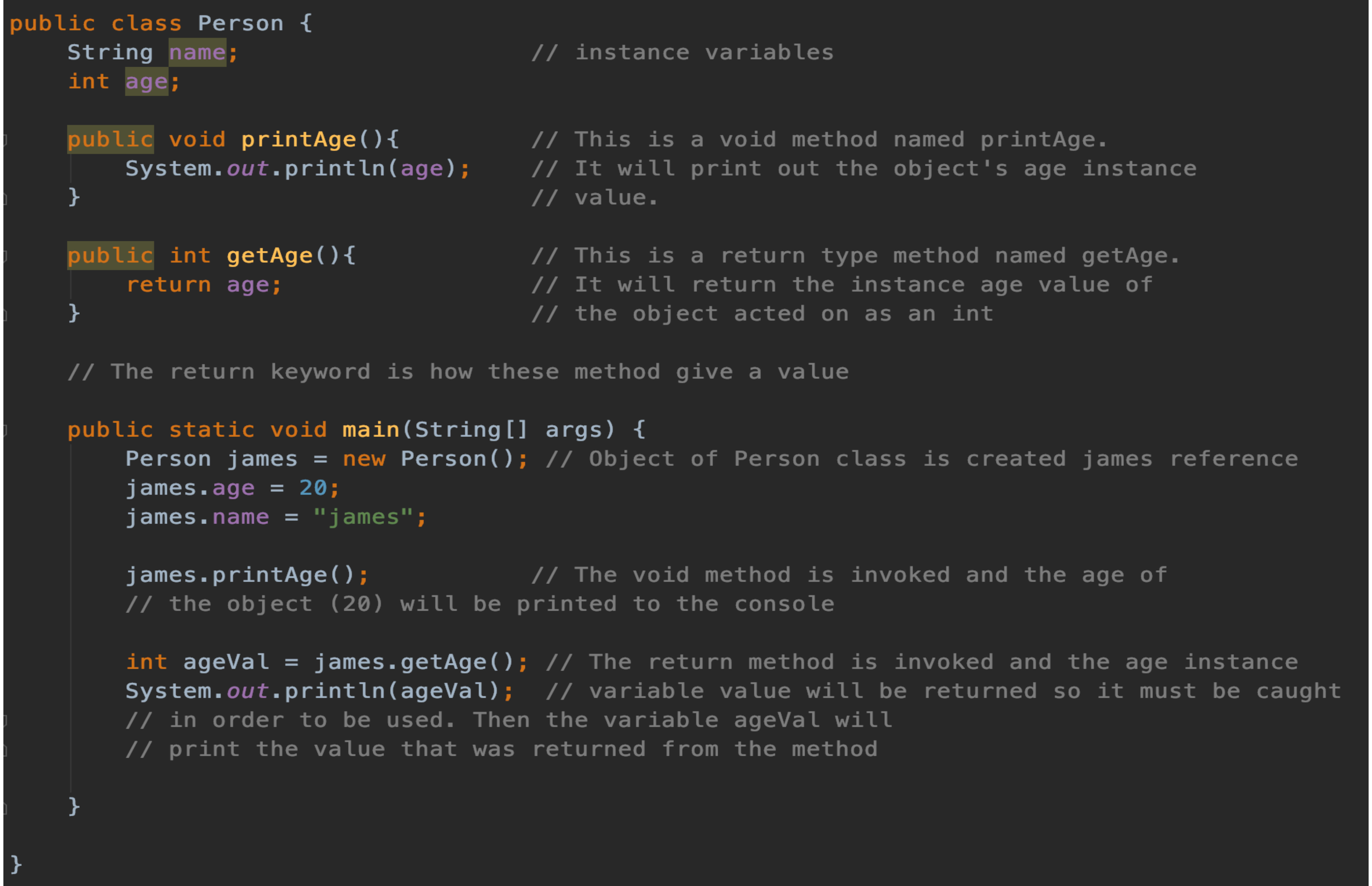
CLASS MEMBERS - INSTANCE VARIABLES



```
public class Person {

    String name;     // Instance variable of String called name
    int age;         // Instance variable of int called age

    public static void main(String[] args) {
        Person joe = new Person(); // Created a Person object called joe
        joe.name = "Joe";          // assigned "Joe" to the name instance variable
        joe.age = 20;              // assigned 20 to the age instance variable

        System.out.println(joe.name); // prints the value of name from object joe
        System.out.println(joe.age);  // prints the value of age from object joe
    }
}
```

CLASS MEMBERS - METHODS

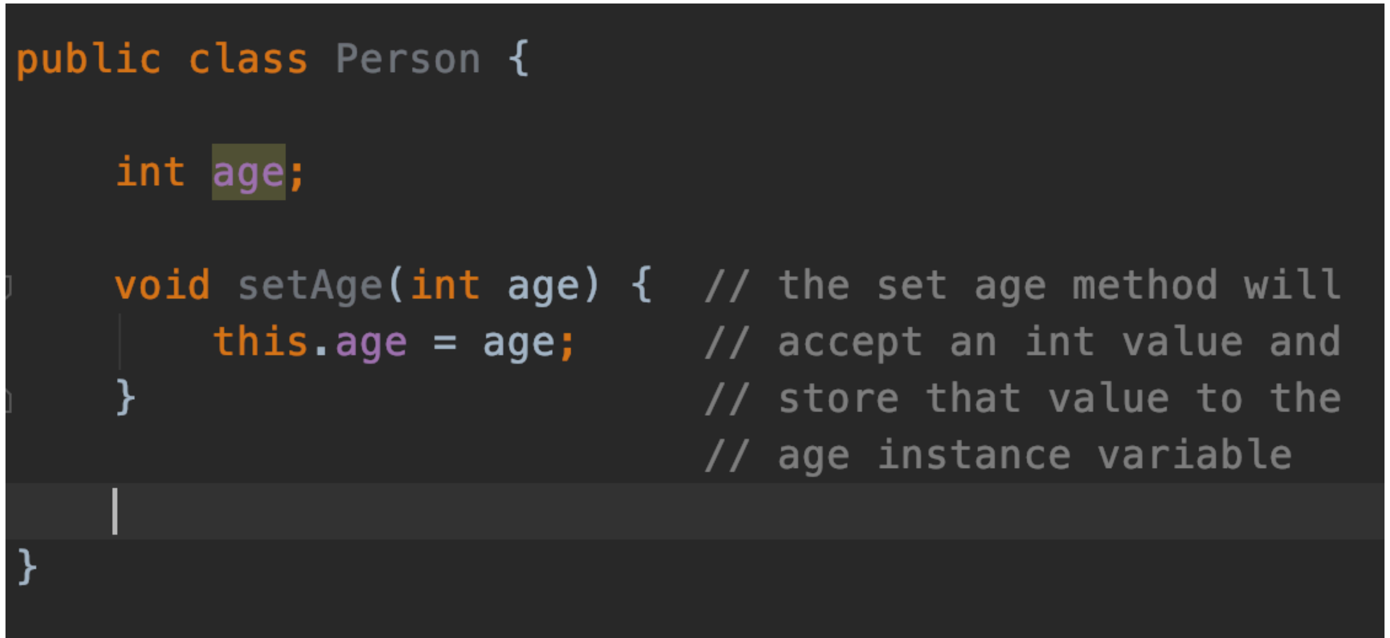
- Methods are functions which perform certain actions
- There is two types of methods: void and return
    - void: Methods that perform some action without returning any value
    - return: Methods that will perform an action and return a single value or object back
- Methods can be invoked by using an object following by the dot operator and method name with parentheses at the end



```java
public class Person {
    String name;                        // instance variables
    int age;

    public void printAge(){             // This is a void method named printAge.
        System.out.println(age);        // It will print out the object's age instance
    }                                   // value.

    public int getAge(){                // This is a return type method named getAge.
        return age;                     // It will return the instance age value of
    }                                   // the object acted on as an int

    // The return keyword is how these method give a value

    public static void main(String[] args) {
        Person james = new Person(); // Object of Person class is created james reference
        james.age = 20;
        james.name = "james";

        james.printAge();               // The void method is invoked and the age of
        // the object (20) will be printed to the console

        int ageVal = james.getAge(); // The return method is invoked and the age instance
        System.out.println(ageVal);  // variable value will be returned so it must be caught
        // in order to be used. Then the variable ageVal will
        // print the value that was returned from the method

    }

}
```

CLASS MEMBERS - METHODS

- Methods can have parameters which act as required information to execute a method
- Any parameters declared in a method act as local variables for that method
- Multiple parameters can be given by separating each variable with a comma



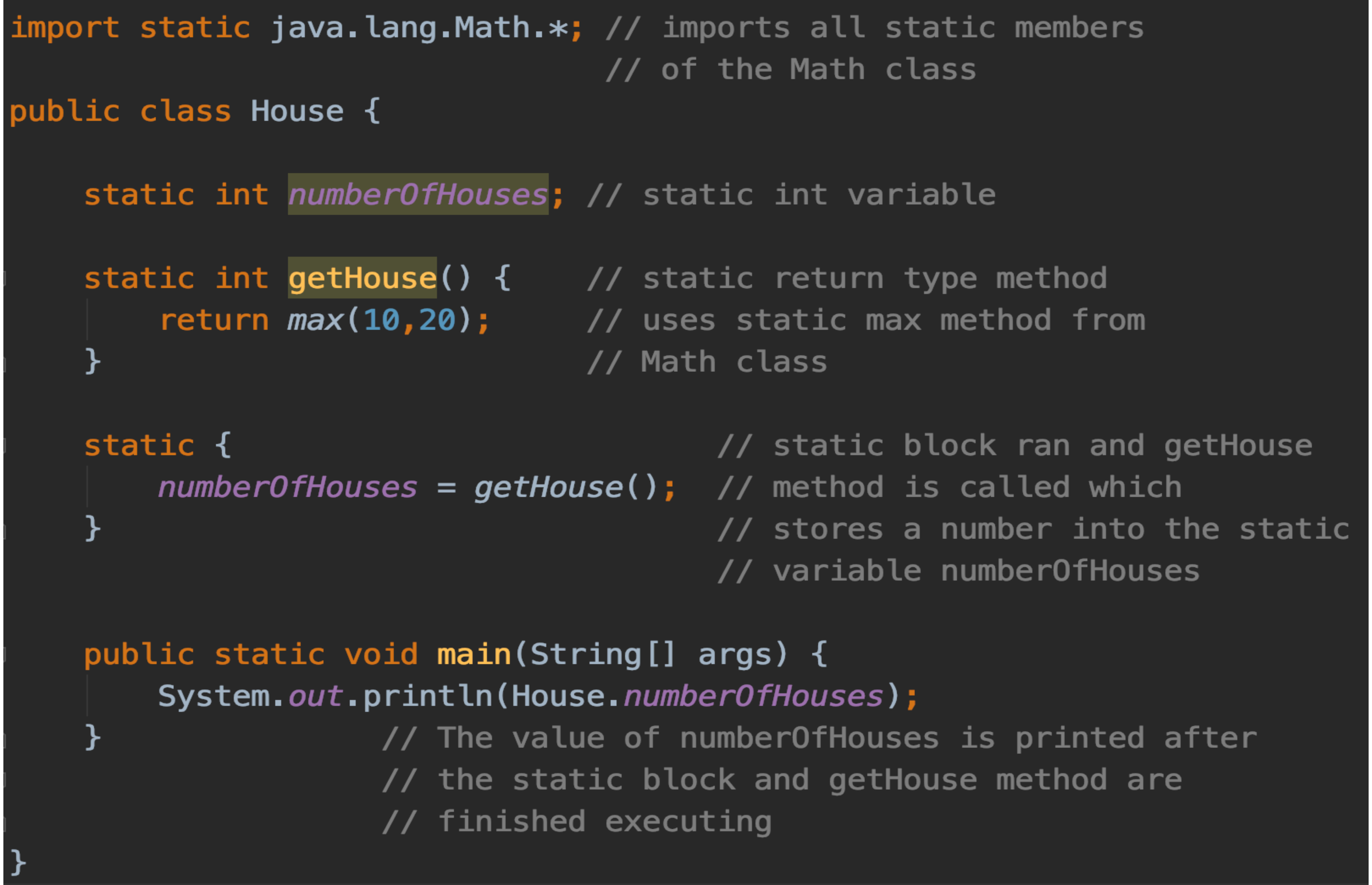
```java
public class Person {

    int age;

    void setAge(int age) {  // the set age method will
        this.age = age;     // accept an int value and
    }                       // store that value to the
                            // age instance variable

}
```

METHOD OVERLOADING

- Method overloading is having multiple methods with the same name within the same class
- Method overloading is achieved by having method with the same name that have a different number of parameters or have a different number of parameters. In short, the method signature must be different to overload a method
- If a method is successfully overloaded, it is possible to change the return type of the method if needed
- When a method is overloaded the arguments will match to the best method that matches those parameter types and execute that method. Methods cannot be ambiguous with these parameters

STATIC MEMBERS

- There can be static variables, methods, blocks, and imports
- Static variables belong to the class and as a result are shared with all objects of the class
- Static methods are methods that belong to the class, so it is possible to call them by giving the class name followed by the method. It is also possible to call these method using any object created from that class
    - Only static variables can be in a static method
- Static blocks allow static variable to be initialized. This code block will run once and before anything else. Only static members are allowed in a static block



```java
import static java.lang.Math.*; // imports all static members
                                // of the Math class
public class House {

    static int numberOfHouses; // static int variable

    static int getHouse() {    // static return type method
        return max(10,20);     // uses static max method from
    }                          // Math class

    static {                              // static block ran and getHouse
        numberOfHouses = getHouse();      // method is called which
    }                                     // stores a number into the static
                                          // variable numberOfHouses

    public static void main(String[] args) {
        System.out.println(House.numberOfHouses);
    }               // The value of numberOfHouses is printed after
                    // the static block and getHouse method are
                    // finished executing
}
```

CONSTRUCTOR

- Constructors are special methods, which are called whenever the new keyword is used to create an object of a class
- By default every class always has a default constructor with no parameters. This default constructor is no longer there if a constructor is created manually
- Constructors usually act to initialize instance variables or perform actions that need to be taken whenever an object of a class is created

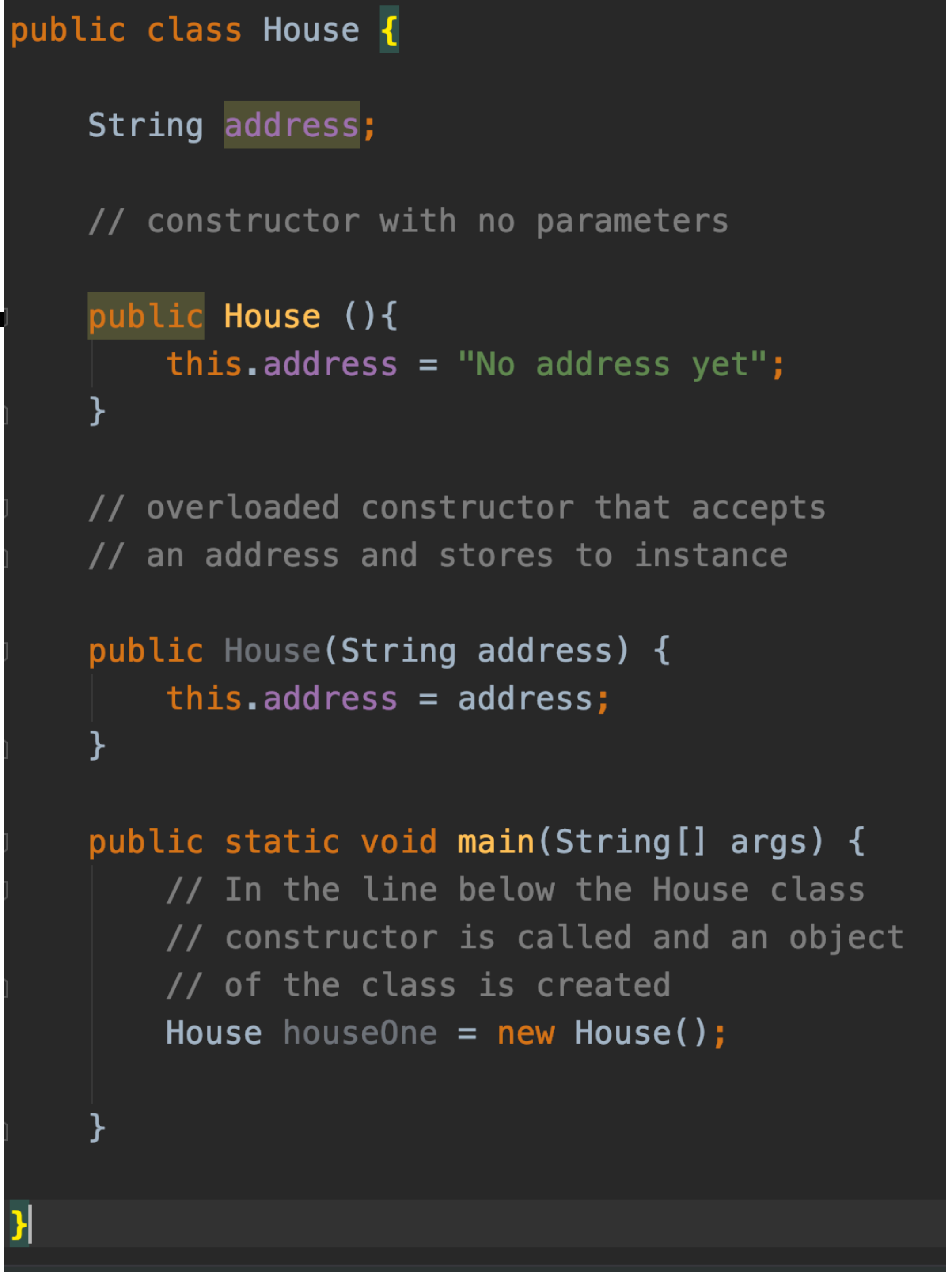
CONSTRUCTOR

In this example the first constructor will be invoked because the constructor call does not have parameters, but as Shown it is possible to also overload constructors

House -> Class name

houseOne -> reference name of object

new House() -> used to invoke constructor

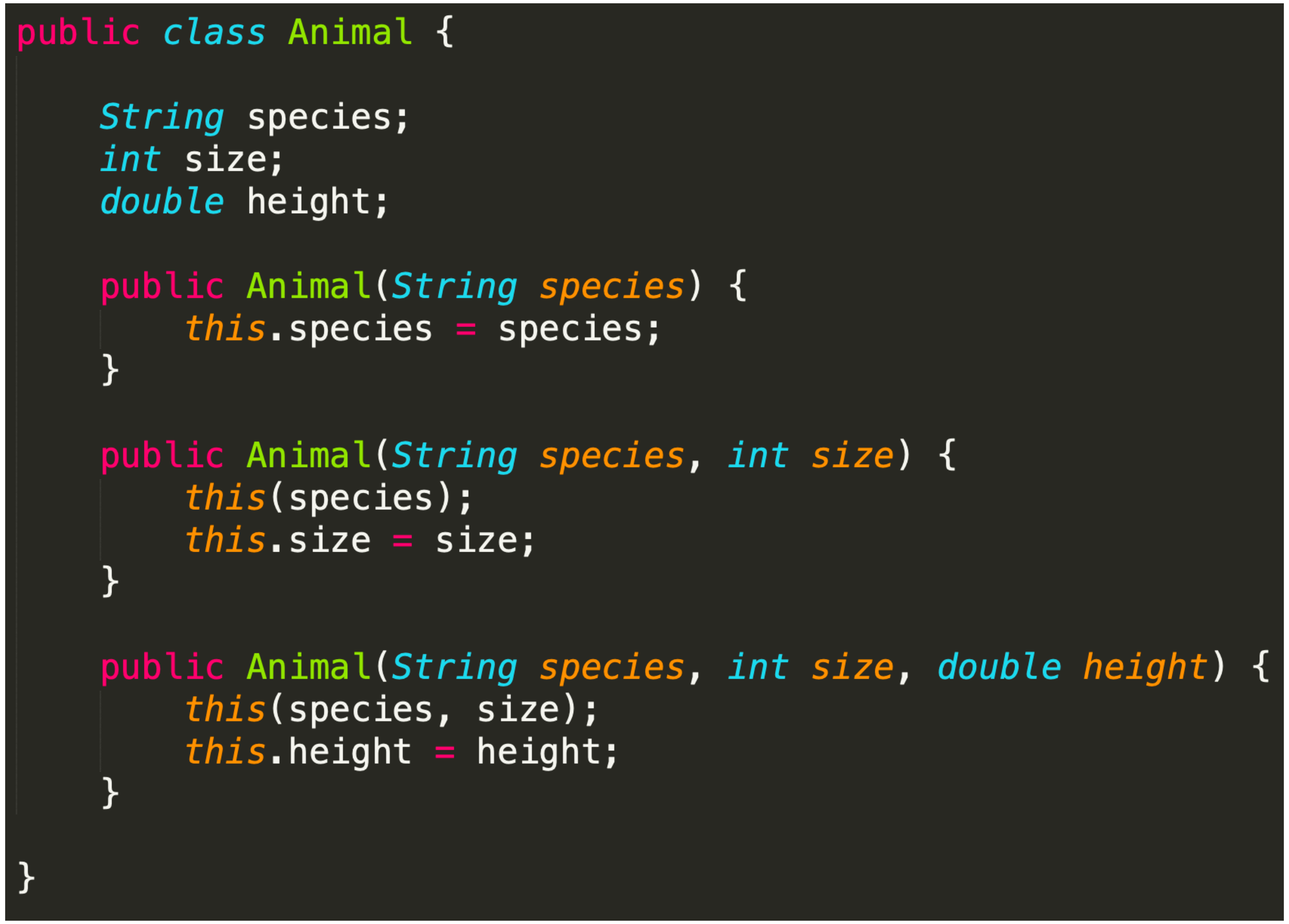


```java
public class House {

    String address;

    // constructor with no parameters

    public House (){
        this.address = "No address yet";
    }

    // overloaded constructor that accepts
    // an address and stores to instance

    public House(String address) {
        this.address = address;
    }

    public static void main(String[] args) {
        // In the line below the House class
        // constructor is called and an object
        // of the class is created
        House houseOne = new House();

    }

}
```

CONSTRUCTOR - OVERLOAD

- Like methods, constructors can also be overloaded. The same rules apply. This is done by providing a constructor with with a different parameters

- this() can be used to call the overloaded constructors with other parameters



```java
public class Animal {

    String species;
    int size;
    double height;

    public Animal(String species) {
        this.species = species;
    }

    public Animal(String species, int size) {
        this(species);
        this.size = size;
    }

    public Animal(String species, int size, double height) {
        this(species, size);
        this.height = height;
    }

}
```

RULES FOR CONSTRUCTOR CHAINING

1. Any constructor call must be call from another constructor

2. In order to call an overloaded constructor, MUST use this() keyword

    this(): can only be used in a constructor calling another constructor (DO NOT USE NAME OF CONSTRUCTOR)

3. Constructor call MUST be at FIRST line of a constructor. If we call at later step ==> Compiler error

4. A Constructor can call ONLY ONE Constructor. If we call twice ==> Compiler error

5. A Constructor can NOT call ITSELF. If it calls itself ==> Compiler error

6. A Constructor can NOT contain itself ==> Meaning if a constructor called another, the latter cannot call the first (because of rule 5)

MUTABILITY

- Mutable: Ability to change the original object
  - Ex: StringBuilder
- Immutable: Inability to change the original object
  - Ex: String

QUESTION - WHAT IS METHOD OVERLOADING

Overloading mean having the same method name but different parameters in the same class. This allows us to have more than one method with same name.

Example: sort method of Arrays class

Arrays.sort(int[] arr)

Arrays.sort(String[] arr)

Method overloading improves the reusability and readability. and it's easy to remember (one method name instead of remembering multiple method names)

QUESTIONS

- Can you overload main method?

  Yes, if we pass different parameters

- Should the return type of an overloaded method be the same or not?

  Return type can be same or different in method overloading

TASK - 1

Create a method that will accept a number and print all the prime number from 2 to that given number

TASK - 2

Create a method that will accept a number and check if the number is an Armstrong number. If the number is an Armstrong number return true otherwise return false.

TASK - 3

- Create a method that will accept a number (long) and determine if the number of palindrome or not.

TASK - 4

- Create a method that will accept a that will accept a number (int) and print that many Fibonacci numbers

TASK - 5

- Create a method that will take any String of letters and numbers and sort each substring of numbers and letters

Ex:

Input: "DC501GCCCA098911"

OutPut: "CD015ACCCG011899"

OPTIONAL PRACTICE FOR CLASSES

TASK - 6

- Create a class Email that has instance variables: recipient (string), subject (string), body (string)
- Create a constructor that require the user to enter the recipient and initialize the value
- Overload the constructor to accept all three variables: recipient, subject, body and initialize the values

TASK - 7 - GOOGLE

- Create a class for a google user with the following instance variables: username, password, name, age, inbox (Array of Emails)
- Create a constructor which will create a google user by only taking the username and password. If the password contains the username then it is not a valid password and should not be saved. In this case print "Password contained username. Default password created: 'password'" and set the password for the user to the default value.
- Overload the constructor to accept username, password, and name. If the name entered has any characters that are not letters from the alphabet, then the name is invalid. In this case print "invalid name" and assign the name value to be 'no name'

TASK - 7 - GOOGLE

- Create a method send that will accept an Email object and return a boolean. If the google user has a valid name (not 'no name'), then the method will return true otherwise the method should return false