** **this(...)** calls constructor of the same class based on the matching arguments

-> Since it should be the first line it can be used only once
-> no recursive constructor calls allowed

this.variableName
this.method()

this.___ is used to access the properties of the current instance of a class, mainly to give priorities to local properties with the same name over global properties.
property: variable/methods

> **process of instantiating and initializing an object**

1. First, the invocation of **NEW** causes the allocation of memory for the entire object, including all the parent elements of which it is made. That memory is also zeroed in this phase (means, memory is allocated for all of its instance variables, and those variables are initialized to default values).

int, float, double, etc., are initialized to 0.
boolean -> false.
Object references (e.g., String, ArrayList, custom objects) -> null.

2. Next, assuming no errors (such as running out of memory) occur in the first step, control is transferred to the constructor with an argument type sequence that matches, or is compatible with, the actual parameters of the invocation

In all contemporary(recent) releases of Java (including Java 17 and later) all constructors start with one of three code elements.
1. First, there is an implicit call to super() with no arguments.
2. followed by either an explicit call to super(...), 3. or a call to this(...), both of which may take arguments.

code:
```
01:  abstract class SupA {
02:    SupA() { this(null); }
03:    SupA(SubA s) {this.init();}
04:    abstract void init();
05:  }
06:  class SubA extends SupA {
07:    void init() {System.out.print("SubA");}
08:  }
```

here:
First, the invocation new SubA() begins by allocating and zeroing memory for the entire object, including the storage necessary for the SubA, SupA, and Object parts. There are no instance fields in the code you see in this question, but the principle is the same.

Next, the newly allocated object is passed as the implicit this argument into the implicit constructor for SubA. That constructor immediately delegates—using super()—to the zero-argument constructor for SupA. That constructor in turn immediately delegates to the one-argument constructor for SupA on line 03 using the explicit call this(null).

this.init();, which invokes the implementation on line 07 and prints the message SubA