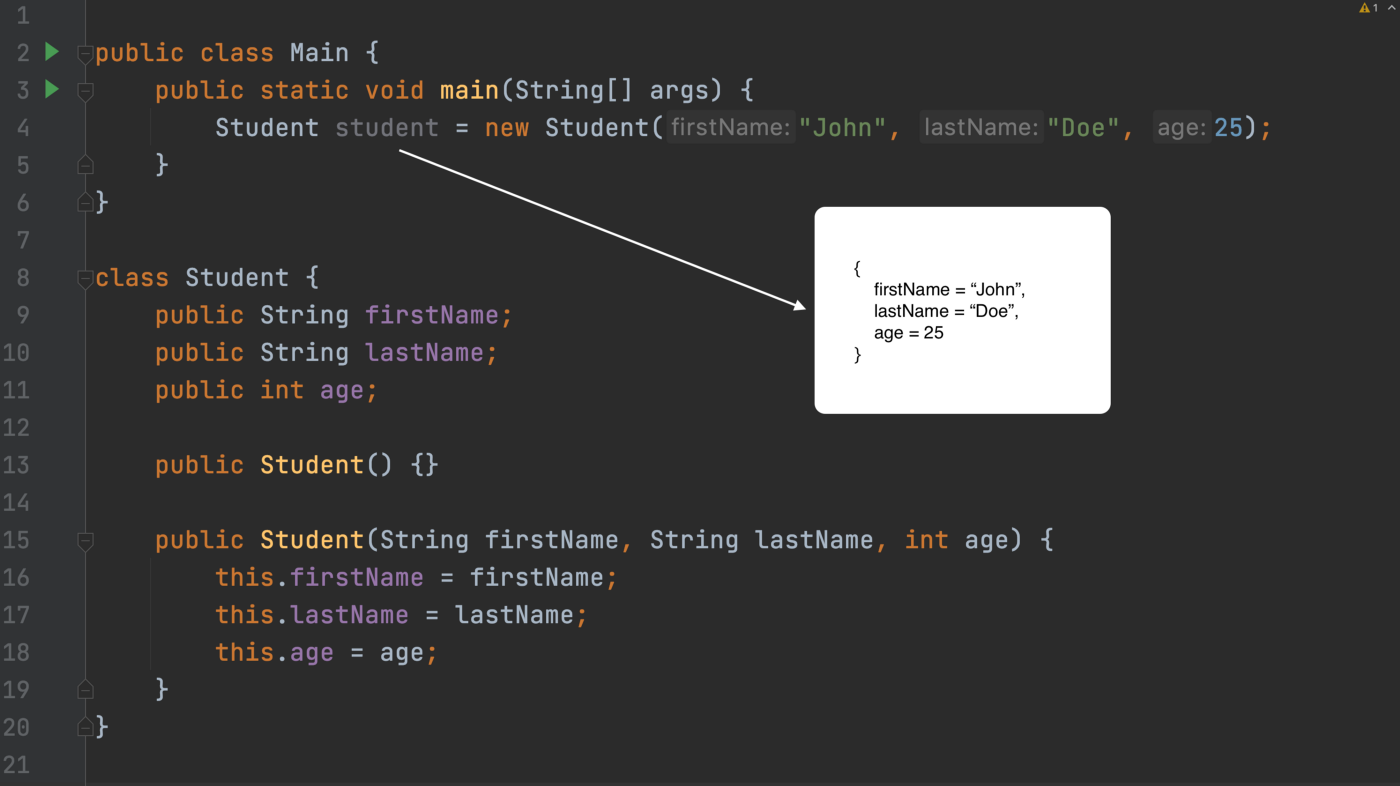
**Class and Object in Java**

Objects are the main part of object-oriented programming. It’s crucial to know how they work.



When building software we move and manipulate with a lot of data. Data comes in different forms so we need a way to represent our data in our code. Java provides 8 core primitive data types.

byte for whole numbers  
short for whole numbers  
int for whole numbers  
long for whole numbers  
float for floating numbers  
double for floating numbers  
char for single character(ACII table)  
boolean logical - can be true or false

Using these primitives we can represent whole numbers, floating numbers, single characters, and logical boolean type.

These primitives are not enough to handle all data types we would need to write our applications. That’s why we have objects. For example, our application works with student data. It would be great to have some data type where we could represent student as an entity. We can do it by creating our own Student class with its properties and methods.

**Class and Objects in Java**

The object in Java is an implementation of a class. The class gives us a template or we can say a structure for our object. In the class, we can declare the object's properties and methods.

*Class is a blue print for objects. Object is an actual implementation of its class.*

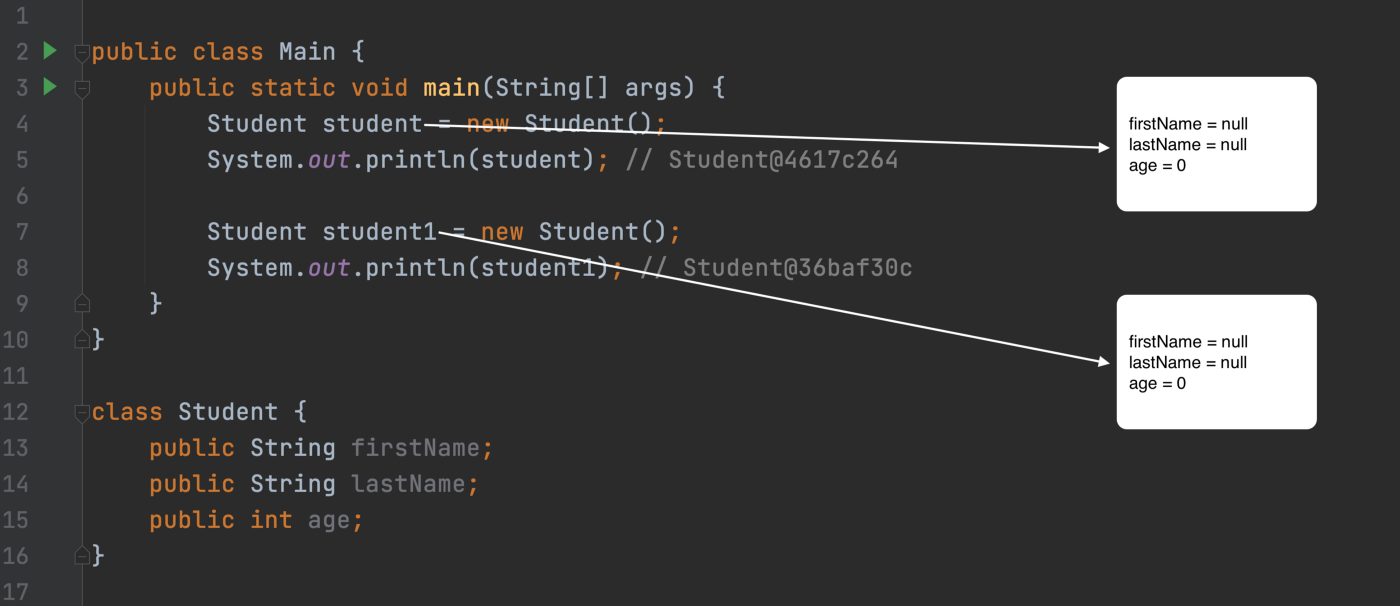
public class Student {  
 public String firstName;  
 public String lastName;  
 public int age;  
}

We have a class above with 3 instance variables. Even name — instance says that they belong to the object. So these three variables are properties of objects that will be created from this class.

public static void main(String[] args) {  
 Student student = new Student();  
 System.out.println(student); // Student@4617c264  
}

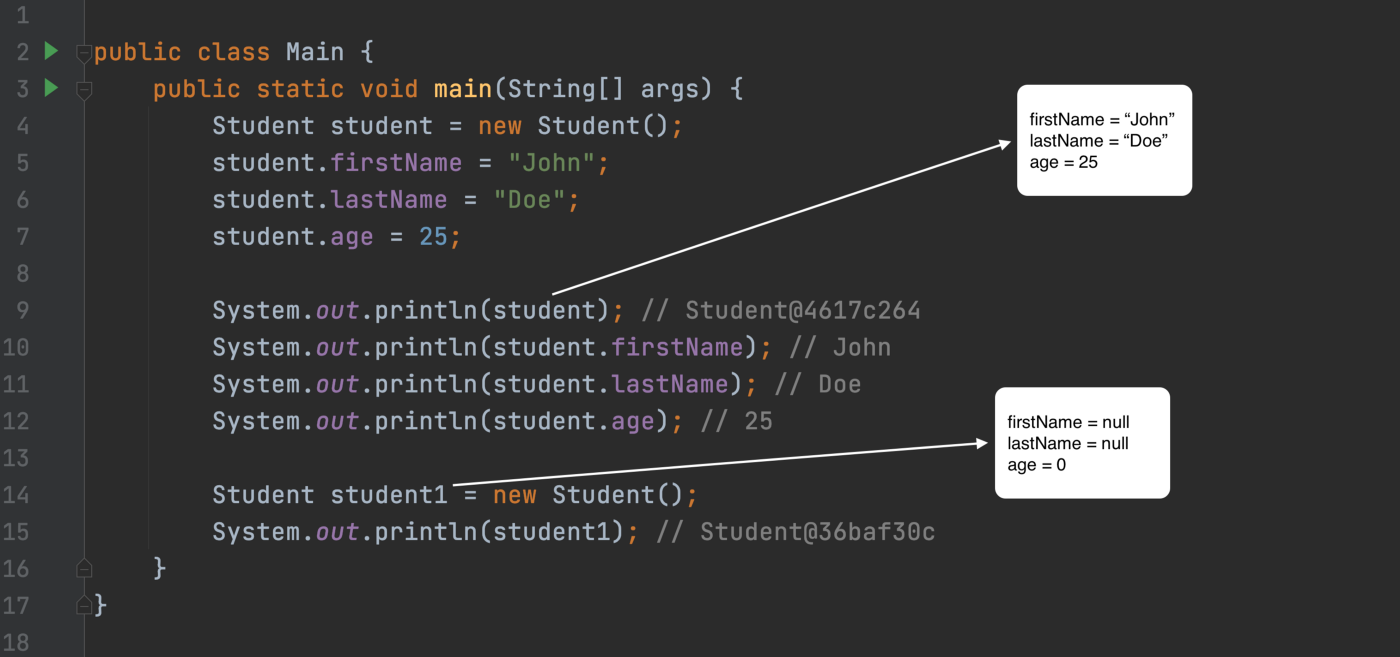
That’s how we create an object from a class. new keyword is always used to create objects in Java.

Student student this part is called reference type and student is an actual reference. We called it a reference because Java stores object in the memory and points to them using the reference. Even when we try to print it, we will get something like this Student@4617c264 . This is the address where the actual object is located. By using this reference we can access object properties and methods.



Here, we created two objects. In the memory, we will have different spaces for these objects and each reference will point to its object.

Another thing to note, the properties of the object will have default values if nothing assigned. Object’s default value is null, the number default value is 0, and the boolean default value is false.



We access object’s properties by using reference name and . . In the above example, we assigned values for our first object and printing them out after.

Different objects are not connected with each other. So when we change firstName for student object it will not affect student1 the object at all. Well, if the variable is static, it belongs to a class, not a specific object. That time it will have an effect.

Let’s talk about non-static methods in the class. Very often methods of the object will manipulate with object’s properties.

The last thing I want to mention is static properties and methods in the class. The static properties belong to the class itself and not to a specific object/instance(even though a specific object has access to static properties and methods). The correct way of using the static members is by class name. The classes which consist of static methods are usually helper classes and they do not serve as data type and we do not create an object from them (for example java.util.Arrays and java.util.Math). There are many examples of hybrid classes that have static and instance members together.

**Constructor in Java**

Constructor is a special method to create an object.

public class Student {  
 public String firstName;  
 public String lastName;  
 public int age;

public Student(String firstName, String lastName, int age) {  
 this.firstName = firstName;  
 this.lastName = lastName;  
 this.age = age;  
 }}

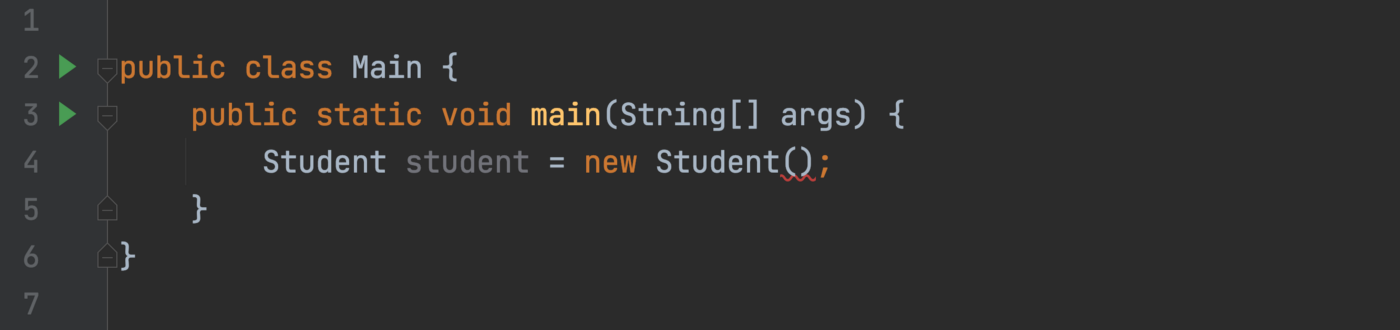
In the above example, we have class Student with one constructor with three arguments.

* The constructor’s name is the same as the class name.
* the constructor can have arguments similarly as methods do.
* the class can have multiple constructors.

Now let's see how we can use our constructor.



* new Student("John", "Doe", 25); this part is using our constructor. It’s the same as with methods if arguments are matching constructor will get called.
* The properties of our values are assigned by our constructor so we don’t need to assign one by one.

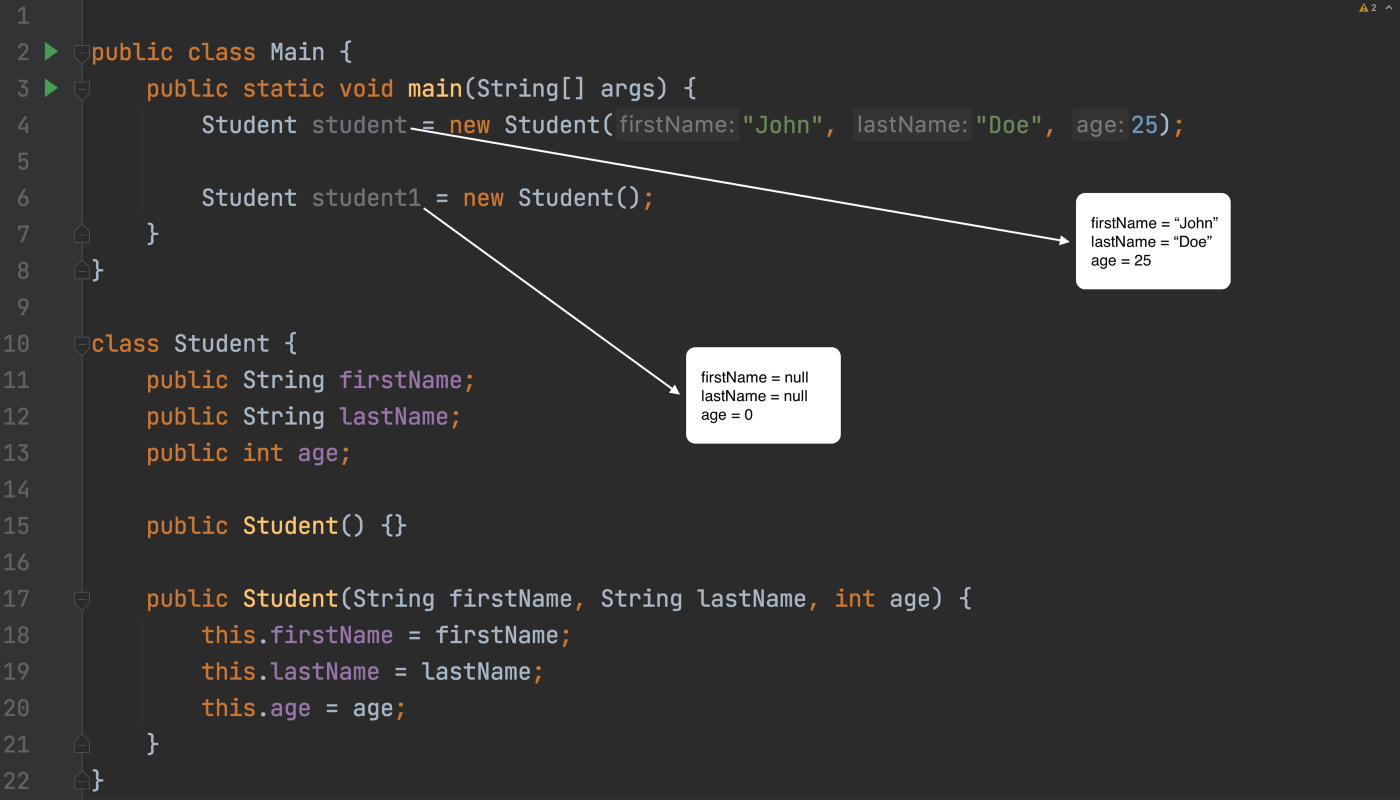


From the same class, we are creating object but didn’t provide any arguments and it’s not compiling. The reason it doesn’t compile because there is no constructor that will match empty arguments.

Initially, when we create a class the default constructor will be provided by java. The default constructor does have any arguments and it’s empty.

When creating our one constructor in our class java will not provide the default one.

We can just add an empty constructor to our class to fix this issue.



* student uses constructor with three arguments.
* student1 uses an empty constructor with no arguments.