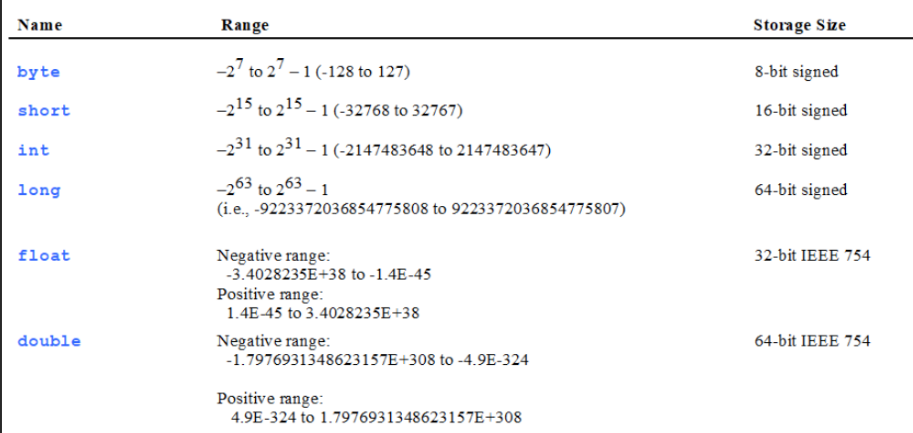
**Data Type**

* Tells the compiler what type of data is stored in a variable
* Memory is being allocated on the stack when we declare a data type
* Once the data type is declared it will only behave that way

Below are primitive data types:



**Type Casting**

Cannot put larger variable into smaller variable unless you do casting

Casting: Converts one data type to another. Can widen or narrow.

**Widening(Implicit casting)**

* Put a value with a smaller range into a larger one
* Done automatically by Java
* Ex: double num = 2;

**Narrowing(Explicit casting)**

* Puts a value with a bigger range into a smaller one
* Must do explicitly because it can result in losing data
* Ex: int num = (int)9.78; the 78 is discarded

**Classes and Objects:**

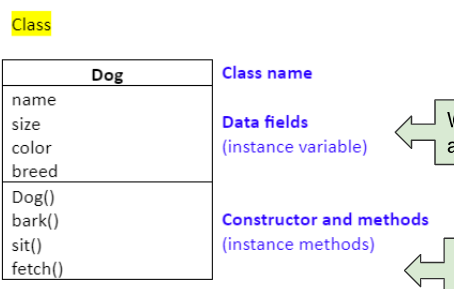
**The difference between a class and an object:**

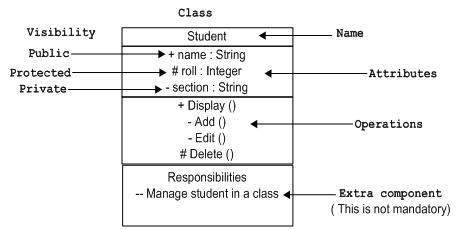
* Classes create the objects properties and behavior
* It's like a blueprint and objects are created from the class.
* Memory is not set aside

**State:** is the properties such as data fields that define an object(what the object knows)

**Behavior:** is the methods that operate in an object(what the object does)

**UML Class Diagram:**





**How to create an object:**

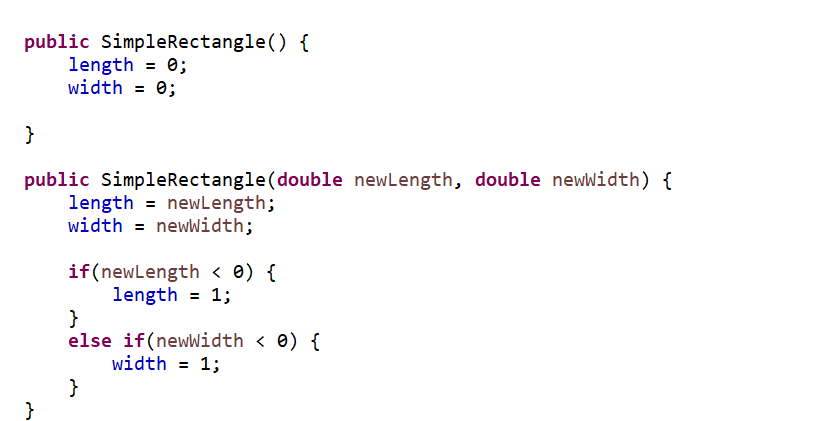
Creating an object is also called instantiation

An object is an instance of a class

Object obj = new Object();

Class name object name = new call constructor

**Constructors:**

****

**what a constructor is and what a constructor does?**

The constructor creates the object and it’s the instance of a class

**In what ways is a constructor different from a regular method?**

The constructor has the same name as the class. It also initializes instance variables of an object.

**The default constructor and an overloaded constructor:**

An overloaded constructor has the same name but different parameters

**what happens when constructors are invoked using new and what is happening in memory on the stack and on the heap?**

The memory is allocated to the heap

**Primitive Types Vs Reference Types**

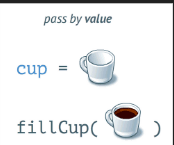
Primitive data types: Memory is on the stack when the variable is declared(int num)

Reference variable: Memory is on the stack when declared(Rectangle rectangle1)

When an object is constructed with new it’s allocated on the heap

**Pass By Reference and By Value**

**Pass by value:** a copy of the value is passed(primitive data types)



**Pass by Reference:** a memory address is passed that refers to where the values are stored on the heap. when you pass a variable by reference to a function, any changes made to that variable inside the function will directly affect the original variable outside the function.

**How are objects passed:**

Objects are passed by value.

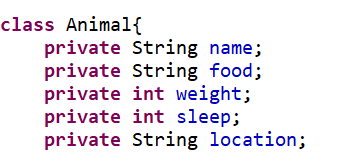
**What are reference variables?**

Variables that store the reference of an instance of a class. Contains the address of the object.

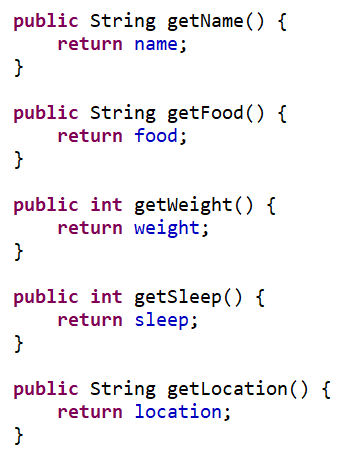
**Instance and Static variables and methods**

**What are instance methods and variables and when should they be used?**

* An instance variable is a variable that is part of a instance of a class(state of an object).
* You use instance variables when the variables belong to an independent instance of a class.

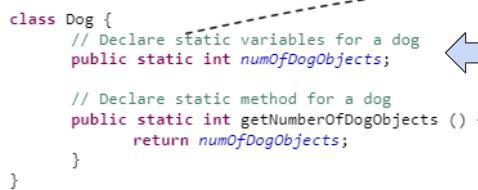


* An instance method is part of an instance of class and can only be executed through a reference of a class(behavior of an object).
* This is used when you want to call a method from instance of a class. Such as accessing/ or modifying(getters and setters) instance variables.



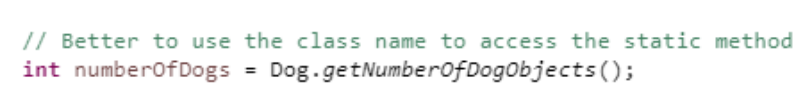
**What static methods and variables and when should they be used?**

* Static variables are variables that are shared by all objects of the class, only one copy exists.
* Use this when you need all objects to share the same value.



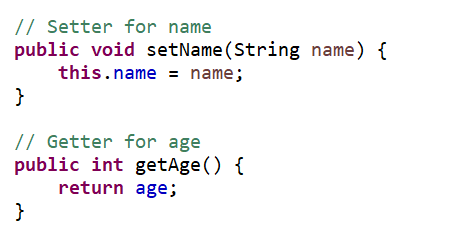
It’s not stored on the heap with the instance variables of an object

* Static methods are methods that are shared by all objects of a class. Dont need ot reference a particular instance.
* You use this when you need a general task done such as calculations

.

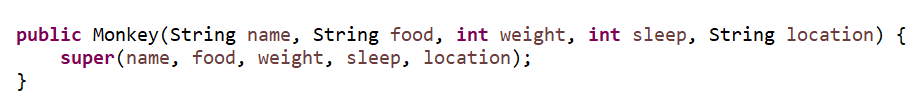
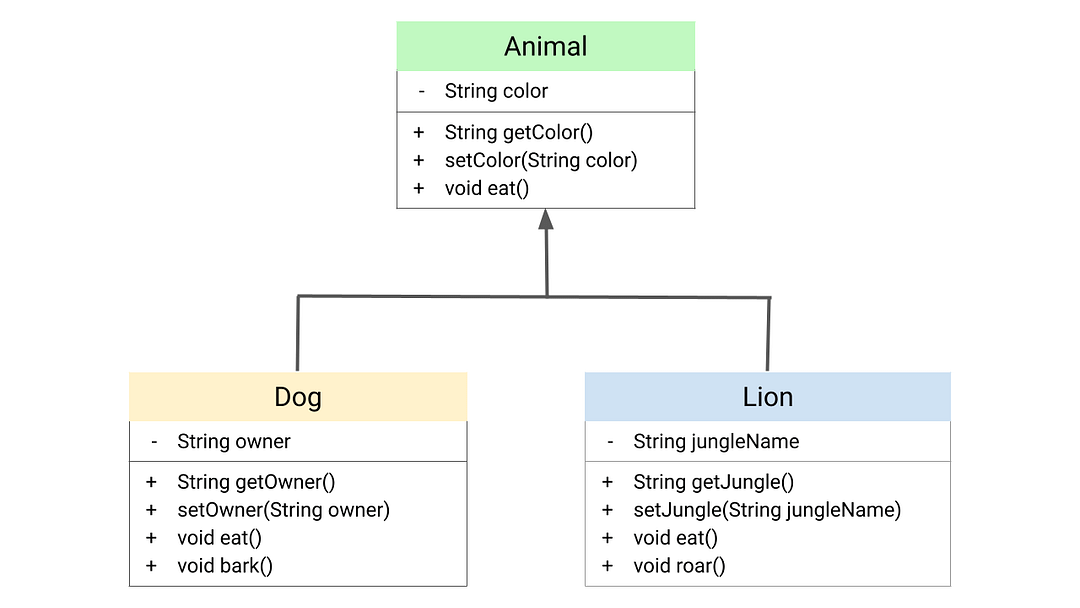
**Encapsulation**

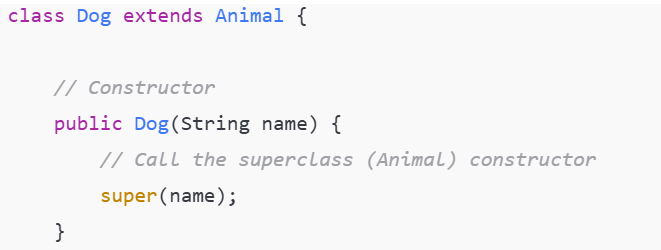
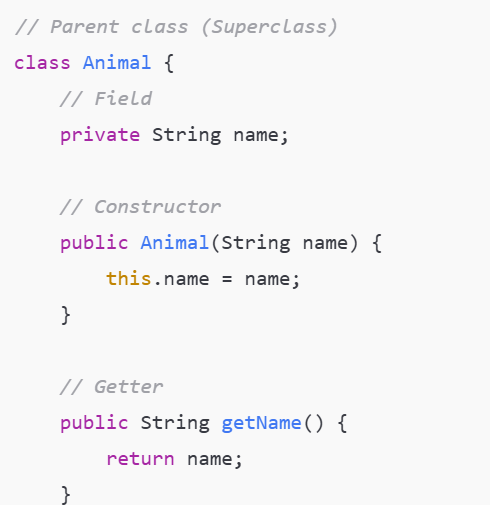
* Structure and implementation is hidden
* Hide data within our classes by using private
* Use getter and setters to change or access private instance variables



**Inheritance**

* When a class is based on an existing class
* Classes have both state and behavior placed
* Class can become a super class and other classes can extend this class
* Classes that extend the super class become subclasses
* Constructor is not inherited by subclass(cannot directly call the superclass constructor).
* Super class constructor can be accessed explicitly or implicitly
* Accessed explicitly use the keyword super
* To access implicitly done use the keyword and the superclasses default constructor will automatically be invoked





**Polymorphism**

* Defines the same method with different function(methods with different behaviors)
* Does this by method overloading and overriding

**Overriding:**

Method overloading is the same method name but different parameters

Method overriding is the same method name and parameters just different function

