Chapter 3 important notes :

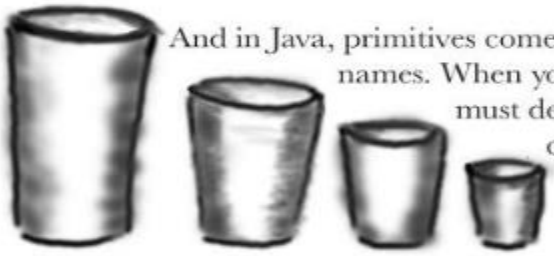
- Variables can be stored into 2 types of things primitive or references and you can find the variables in places like the object states or variables are defines inside the methods as a local ones and then after we will see the arguments and this is the variables that sent to the methods when calling them or you can see them as parameters for methods or as a return types
- You must know that java cares about the type so that means the variables needs name and type and the variables is just a contains which holds something which needs from you to define its size and its type what the type of things will be stored inside this container and be sure the value can fit into the variable so choose the type carefully not be too huge and not too small
- Okay now we know about choosing the type for the variable but what about the names is any name I can choose ? no there is names conventions existed you must follow it :

It must start with a letter, underscore (_), or dollar sign ($). You can't start a name with a number.

After the first character, you can use numbers as well. Just don't start it with a number.

It can be anything you like, subject to those two rules, just so long as it isn't one of Java's reserved words like (new , void , static , int , return)the reserved words is any word the compiler can recognize it .

small short tall grande



And in Java, primitives come in different sizes, and those sizes have names. When you declare any variable in Java, you must declare it with a specific type. The four containers here are for the four integer primitives in Java.

Each cup holds a value, so for Java primitives, rather than saying, "I'd like a tall french roast," you say to the compiler, "I'd like an int variable with the number 90 please." Except for one tiny difference...in Java you also have to give your cup a *name*. So it's actually, "I'd like an int please, with the value of 2486, and name the variable ***height***." Each primitive variable has a fixed number of bits (cup size). The sizes for the six numeric primitives in Java are shown below:

- Now we finish the primitive variables and know how to deal with it and how to assign a value to it now lets start with the non primitive ones(objects) so first thing we know that the objects live in just only one place which is the garbage collectable heap okay in the primitive variables it store inside it the value of this variable otherwise the reference variable is store inside it the way we can get to the object to get its value from there cause the object is in the heap and we need to store each address in the heap and what about we need its value? Go to its place and get its value from there but why this more steps? Why not just store the values of the objects inside the objects and use its value faster rather than the previous steps? That's because the objects size is dynamic not like the primitive types it has fixed size which means we can not store it in the stack that's means the objects needed to be in the heap to can have a space to take its dynamic size okay but different objects types means different sizes so we need something fixed can be used or fitted in all different objects here comes the pointer or the way, address for the object in the heap and the size of the pointer is fixed cause it is address in the memory that's why we can not store value directly inside the objects .
- If we see this example :

Dog d = new Dog();
-The Dod d part here is the first step happened which is the declaration of the object and here there is space allocated here but for the reference variable which will store the pointer address to the object .
-New Dog(); is the second step happened which is created the memory for this object here there is space will be allocated too but for the object itself in the heap.
-= is the third step that makes the assignment the linking happened here

- All references for a given JVM will be the same size regardless of the objects they reference, but each JVM might have a different way of representing references, so references on one JVM may be smaller or larger than references on another JVM and there is an important thing here you can not play with the reference variable increase or decrease it no that's will not happened cause java is not c .
- array element of that type. So in an array of type int (int[]), each element can hold an int. In a Dog array (Dog|[]) each element can hold…a Dog? No, remember that a reference variable just holds a reference (a remote control), not the object itself: So in a Dog array, each element can hold a remote control to a Dog. Of course, we still have to make the Dog objects…the array is an object, even though it's an array of primitives.
- Regardless of what the array holds the array itself is always an object .
- As we know that java cares about the type so when we define an array with specific type you cannot put another different type on the same array for example if you define array of dogs objects you can not add cat with them or if you make an array of int so you can not add float numbers on it too this called spillage but but you can add byte into int array cause the byte is will always fit into the int size and this thing called implicit widening .
- THIS IS THE BULLET POINTS FOR THIS CHAPTER :
  -Variables come in two flavors: primitive and reference.
  -Variables must always be declared with a name and a type.
  -A primitive variable value is the bits representing the value (5, 'a', true, 3.1416, etc.).
  -A reference variable value is the bits representing a way to get to an object on the heap.
  -A reference variable is like a remote control. Using the dot operator (.) on a reference variable is like pressing a button on the remote control to access a method or instance variable.
  -A reference variable has a value of null when it is not referencing any object.

  -An array is always an object, even if the array is declared to hold primitives. There is no such thing as a primitive array, only an array that holds primitives.