**Academia Java – JAVA**

***Primitivos y referencias***

Tabla de contenido

[CONOCE TUS VARIABLES 3](#_Toc310002842)

[1. Declarando una variable 4](#_Toc310002843)

[1.1. Quisiera un double mocha…ermm, no mejor un int con crema 5](#_Toc310002844)

[1.2. De veras no quieres rebalsar eso… (Asegúrate que el valor se ajuste en tu variable) 7](#_Toc310002845)

[1.2.1. Ejercicio de aplicación 8](#_Toc310002846)

[1.3. Atención a las palabras claves 8](#_Toc310002847)

[1.4. Tabla de palabras reservadas 9](#_Toc310002848)

[2. Controlando a tus objetos 10](#_Toc310002849)

[2.1. 3 pasos para declarar, crear y asignar objetos 11](#_Toc310002850)

[2.2. La vida en el gargabe-collection heap 13](#_Toc310002851)

[2.3. La vida y la muerte en el heap 14](#_Toc310002852)

[2.4. Un arreglo es como…una bandeja de tazas! 15](#_Toc310002853)

[2.5. Los arrays son objetos también 15](#_Toc310002854)

[2.6. Ahora nos toca…un array de Dogs! (¿?) 16](#_Toc310002855)

[2.7. Controlando al Dog (con una variable de referencia) 17](#_Toc310002856)

[2.7.1. Un arreglo de Dogs 17](#_Toc310002857)

[2.8. Sé el compilador 19](#_Toc310002858)

# CONOCE TUS VARIABLES

Las variables son de dos tipos: primitivos y de referencia. Hasta ahora has usado variables en dos lugares - como estado de los objetos (variables de instancia), y como variables locales (variables declaradas dentro de un método). Más adelante, usaremos variables como argumentos (valores enviados a un método por medios de las llamadas), y como los tipos de retorno (valores enviados de retorno a las llamadas de los métodos).

Ya has visto variables declaradas como simples números de tipo enteros primitivos (tipo int). También has visto variables declaradas como algo más complejo, tal como una cadena. Pero seguramente piensas que tiene que haber más en la vida que los números enteros, cadenas y etc.

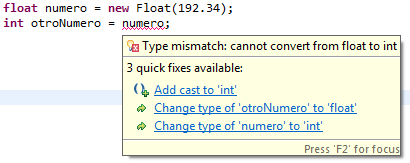
¿Qué pasa si tienes un objeto DueñoDeMascota con una variable de instancia de tipo Perro? O un objeto Carro con una variable Motor? Ahora vamos a develar los misterios de más tipos Java y observarás que puedes declarar como variable, que puedes asignar a una variable, y que puedes hacer con una variable. Y finalmente verás cómo es la vida realmente en el garbage-collectible heap.

# Declarando una variable

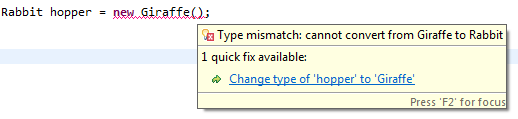
Java se preocupa por el tipo de datos. No te va a permitir hacer algo tan tirado de pelos como instanciar una variable Rabbit con una del tipo Giraffe.

¿Te has preguntado qué pasaría si alguien llama al método hop() de Rabbit?

De esta misma forma no te permitirá asignar un número de punto flotante a una variable entera. A menos que tú conozcas tanto al compilador para lograr hacerlo y tomes en cuenta que perderás precisión (como, todo lo que venga después del punto decimal).



El compilar detectará la mayoría de problemas en tiempo de codificación:



No esperes que compile. Gracias a Dios!

Para que todo esto funcione, debes declarar siempre el tipo de tu variable. Es un entero? Un perro? Un simple carácter? Igual debes hacerlo.

Las variables vienen en dos sabores: primitivos y referencias a objetos. Los primitivos toman valores básicos (piensa en simples patrones de bits) como enteros, booleanos y números de puntos flotantes. Las referencias a objetos toman…bueno…referencias a objetos (tsss, creo que eso no lo clarificó demasiado ☹).

En fin, primero daremos un vistazo a los primitivos y luego veremos que hay con las referencias a objetos (y que significan realmente).

Antes de eso, te recalco que sin importar cuál de los dos uses, debes respetar lo siguiente:

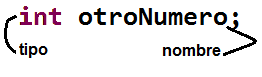
Las variables deben tener un tipo.

Y además del tipo, las variables también necesitan un nombre que debe ser colocado al lado del tipo en el código:

Las variables deben tener un nombre.

Al ver una sentencia como: "un objeto de tipo X", quizás pienses que tipo y clase son sinónimos. Eso lo revisaremos más adelante.

En pocas palabras:



# Quisiera un double mocha…ermm, no mejor un int con crema

Cuando pienses en variables Java, piensa en tazas. Tazas para café, para té, tazas gigantes (¿chops?)...en fin, tazas de todo tipo.

**Una variable es solo una taza. Un contenedor. Contiene algo.**

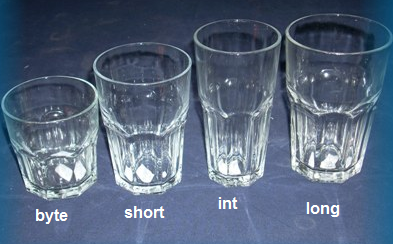
Tiene un tamaño y un tipo. Primero vamos a revisar las variables (tazas) que son primitivos, poquito después veremos las tazas…variables que referencias a objetos.

Esta simple analogía nos ayudará a comprender tanto el lado simple como el lado complejo del asunto de las variables. Lado al cual llegaremos pronto. ☺

Los tipos de datos primitivos son como las tazas del “coffeehouse”. Si has ido a Starbucks, sabes de que estamos hablamos. Ellos sirven las bebidas en tazas de diferentes tamaños, y cada uno tiene un nombre como “normal”, “alto” y “grande”. Incluso puedes ver sus presentaciones a en alguna alacena que exhiben.

|  |  |
| --- | --- |
|  |  |

De la misma forma los tipos Java primitivos vienen en diferentes tamaños y cada uno tiene un nombre. Cuando declares una variable en Java, debes declarar un tipo especifico. Te presento a los cuatro contenedores para los cuatro tipos de números enteros primitivos Java.



Cada vaso toma un valor, entonces para los tipos de datos Java primitivos tendrás que decirle al compilador “Me gustaría una variable de tipo int con valor 90, por favor”.



Una pequeña acotación…en Java además debes darle un nombre a tu taza. Entonces realmente lo que deberías decirle al compilador es “Me gustaría una variable de tipo int con valor 90 y de nombre height, por favor”. Cada variable de tipo primitivos tiene un número fijo de bit (tamaño de la taza). A continuación los tamaños para cada unos de los seis tipos Java primitivos:

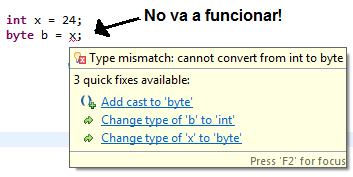
|  |  |
| --- | --- |
|  |  |

# De veras no quieres rebalsar eso… (Asegúrate que el valor se ajuste en tu variable)

Algo que no debes hacer es asignar un valor más grande a una variable con una capacidad menor (no debes servir más café del que la taza puede contener).

Pero eso no significa que no puedas (siempre puedes derramar el café…aunque no deberías). Puedes hacerlo pero vas a perder algunas cosas en el camino…quizás si derramas el café sobre la alfombra favorita de tu mamá podrías perderte el fin de semana pues estarás lavándola.

El compilador intentará prevenirlo, por ejemplo:



Por qué no funciona? Te debes preguntar. Después de todo, el valor de x es 24, y 24 es lo suficientemente pequeño para alcanzar en un byte. Lo sabes y lo sé. Pero en pocas palabras el compilador prefiere prevenir y cuida que en algún momento no vayas asignar un valor mucho más grande en una variable que no pueda contenerlo.

Puedes asignar un valor a una variable de una de varias formas incluyendo:

* Asignar un valor literal después del signo igual (x = 12, isGood = **true**, etc.)
* Asignar el valor de una variable a otra (x = y)
* Usar una expresión combinada de los dos anteriores (y = x + 45)

|  |  |
| --- | --- |
| **int** size = 32; | declara una int variable llamada size y le asigna el valor de 32 |
| **char** initial = 'j'; | declara una variable char llamada initial, se asigna el valor 'j' |
| **double** d = 456.709; | declara una variable double llamada d, se le asigna el valor de 456.709 |
| **boolean** isCrazy; | declara una variable boolean llamada IsCrazy pero no se le asigna nada |
| isCrazy = **true**; | asigna un valor a la variable declarada isCrazy |
| **int** y = x + 456; | declara una variable int llamada y, y le asigna el valor de la suma de x más 456 |

# Ejercicio de aplicación

El compilador no te va a deja poner el contenido de una taza grande en una pequeña. Pero ¿qué pasa con el otro sentido, verter el contenido de una taza pequeña en una grande? Con eso no hay problema. ☺

Con todo lo que sabes hasta ahora de los tipos y tamaños de variables, indica cual de las siguientes sentencias son legales y cuáles no.

|  |  |  |
| --- | --- | --- |
|  | No es legal | Si es legal |
| 1. **int** x = 34.5; |  |  |
| 2. **boolean** boo = x; |  |  |
| 3. **int** g = 17; |  |  |
| 4. **int** y = g; |  |  |
| 5. y = y + 10; |  |  |
| 6. **short** s; |  |  |
| 7. s = y; |  |  |
| 8. **byte** b = 3; |  |  |
| 9. **byte** v = b; |  |  |
| 10. **short** n = 12; |  |  |
| 11. v = n ; |  |  |
| 12. **byte** k = 128; |  |  |

# Atención a las palabras claves

Ya sabes que necesitas un nombre y un tipo para tus variables. Además, también ya conoces los tipos primitivos.

Pero, ¿puedes usar cualquier nombre para llamar a tus variables? Las reglas son bastante simples, menos mal. Puedes usar nombre para clases, métodos o variables de acuerdo a las siguientes reglas (las cuales en su mayoría son flexibles, pero te conviene mucho usarlas):

* Deben empezar con una letra, un guión bajo o un signo de dólar. No puedes comenzar un nombre con un número.
* Después del primer carácter, puedes usar números. Pero no al iniciar.
* A excepción de las dos reglas anteriores y de palabras reservadas, puedes usar cualquier otro patrón.

Ya has usado muchas palabras reservadas cuando has desarrollado tu código anterior:

**public static void**

Las palabras reservadas para los tipos de datos primitivos son:

**boolean char byte short int long float double**

Por ahora solo necesitas conocerlos, no es necesario que sepas que significan. No es necesario aún que los memorices ahora. Más adelante los revisarás más a fondo.

# Tabla de palabras reservadas

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **boolean** | **byte** | **char** | **double** | **float** | **int** | **long** | **short** | **public** | **private** |
| **protected** | **abstract** | **final** | **native** | **static** | **strictfp** | **synchronized** | **transient** | **volatile** | **if** |
| **else** | **do** | **while** | **switch** | **case** | **default** | **for** | **break** | **continue** | **assert** |
| **class** | **extends** | **implements** | **import** | **instanceof** | **interface** | **new** | **package** | **super** | **this** |
| **catch** | **finally** | **try** | **throw** | **throws** | **return** | **void** | **const** | **goto** | **enum** |

# Controlando a tus objetos

Ya sabes cómo declarar variables primitivas y asignarles valores. ¿Pero que hay acerca de las variables no primitivas? ¿En otras palabras, que de los objetos?

* De hecho, no hay tal cosa como una variable objeto.
* Lo que si hay es variables que referencian objetos.
* Una variable de referencia de objeto contiene bits que representan una forma de acceder a un objeto.
* No se tiene el objeto en sí, sino que se asigna algo así como un puntero. O una dirección de memoria. En Java no se sabe muy bien lo que está dentro de una variable de referencia. Lo que si podemos asegurar es que, lo que sea representa a sólo un objeto. Y la JVM sabe cómo usar la referencia para llegar a ese objeto.

|  |  |  |
| --- | --- | --- |
|  |  | Esta sería la analogía correcta:  Dog d = **new** Dog();  d.bark();  Piensa en una variable de referencia como el control remoto de tu mascota (un perro). Se usar para hacer que el objeto haga algo (invocar métodos). |

No se puede asignar un objeto en una variable. A menudo pensamos de esa forma... y decimos cosas como, "Pasé el String al método System.out.println ()." O, "El método me devuelve un Dog", o "Asigné un nuevo objeto Foo en la variable llamada myFoo".

Pero eso no es lo que sucede. No hay tazas tan gigantes que puedan crecer de acuerdo a las necesidades de algún objeto. Los objetos viven en un único lugar: el garbage collectible heap! (aprenderás más sobre esto más adelante).

Ahora, si bien una variable primitiva son un conjunto de bits que representan el valor real de la variable, una variable de referencia a un objeto es un conjunto de bits que representan una forma de llegar al objeto.

Además, usas el operador punto (.) referenciando a una variable para decir, "usa lo que está antes del punto para darme lo que está después del punto." Por ejemplo:

myDog.bark();

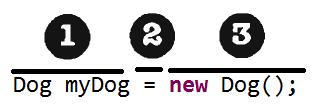
Lo cual significa “usa el objeto referenciado por la variable myDog para invocar al método bark()”.



Una referencia a un objeto es solo otro valor de una variable. Algo que va en una taza, solo que en este caso, el valor es un control remoto.

|  |  |
| --- | --- |
| **Variable primitiva**  **byte** x = 7;  Los bits que representan el número 7 van en la variable. (00000111) | **Variable de referencia**  Dog myDog = **new** Dog();  Los bits representan la ruta para obtener el objeto Dog están en la variable. El objeto Dog por sí mismo no está dentro de la variable! |
|  |  |

# 3 pasos para declarar, crear y asignar objetos



|  |  |
| --- | --- |
|  |  |
| **Dog myDog** = **new** Dog();  Le dice a la JVM que reserve un espacio para una variable de referencia, y la nombra myDog. La variable de referencia será siempre de tipo Dog. En otras palabras, control remoto que tiene botones para controlar a un Dog, no a un Cat o un Button o un Socket. |
|  |  |
| Dog myDog = **new Dog();**  Le dice a la JVM que reserve un espacio para el objeto new Object en el heap. |
|  |  |
| Dog myDog **=** new Dog();  Asigna new Dog a la variable de referencia myDog. |

# La vida en el gargabe-collection heap

|  |  |
| --- | --- |
| Book b = **new** Book();  Book c = **new** Book();  Declara dos variables de referencia Book. Crea dos nuevos objetos Book. Asigna los objetos Book a las variables de referencia.  Los dos objetos Book ahora están viviendo en el heap.  Referencias: 2  Objetos: 2 |  |
| Book d = c;  Declara una nueva variable de referencia Book. En lugar de crear un tercer Book, asigna el valor de la variable c a la variable d. Pero ¿qué significa esto? Es como decir, “Toma los bits de c, has una copia de ellos y cópialos en d”.  **Ambos, c y d referencian al mismo objeto.**  **Las variables toman dos copias diferentes del mismo valor. Dos controles remotos para un mismo TV.**  Referencias: 3  Objetos:2 |  |
| c = b;  Asigna el valor de la variable b a la variable c. Por ahora tú sabes lo que esto significa. Los bits de la variable b son copiados, y la copia es asignada a la variable c.  **Ambos, b y c referencias al mismo objeto.**  Referencias: 3  Objetos: 2 |  |

# La vida y la muerte en el heap

|  |  |  |  |
| --- | --- | --- | --- |
| Book b = **new** Book();  Book c = **new** Book();  Declara dos variables de referencia Book. Crea dos nuevos objetos Book. Asigna los objetos Book a las variables de referencia.  Los dos objetos Book ahora vivin enel heap.  Referencias activas: 2  Objetos accesibles: 2 | | |  |
| b = c;  Asigna el valor de la variable c a la variable b. Los bits de la variable c son copiados, y esta copia es asignada a la variable b. Ambas variables toman valores idénticos.  **Ambos b y c refieren al mismo objeto. Object 1 es abandonado y pasa a ser candidato perfecto para el GC.**  Referencias activas: 2  Objetos accesibles: 1  Objetos abandonados: 1  El primer objeto que referenció b, Object 1, no tiene más referencias. Es inaccesible. | |  | |
| c = **null**;  Asigna el valor null a la variable c. Esto hace que c referencie a null, lo que significa que no referencia a nada. Pero aun es una variable de referencia y otro objeto Bool puede asignársele.  Object 2 aun tiene una referencia activa (b), y por ello no es elegible por el GC.  Referencias activas: 1  Referencias null: 1  Objetos accesibles: 1  Objetos abandonados: 1 |  | | |

# Un arreglo es como…una bandeja de tazas!

|  |  |
| --- | --- |
|  | Declara una variable array de int. Una variable array (arreglo) es un control remoto a un objeto array.  **int**[] nums; |
|  | Crea un nuevo array de int con una longitud de 7, y asígnalo a la variables nums previamente declarada.  nums = **new** **int**[7] ; |
|  | Dale a cada element del array un valor int. Recuerda, los elementos en un array int son solo variables int.  nums[0] = 6;  nums[l] = 19;  nums[2] = 44:  nums[3] = 42:  nums[4] = 10:  nums[5] = 20;  nums[6] = 1; |
|  |  |

# Los arrays son objetos también

La librería standard de java incluye muchas estructuras de datos sofisticadas, incluyendo maps, tres, and sets, pero los arrays son estupendos cuando solo quieres hacer un ordenamiento rápido y eficiente de una lista de cosas. Te dan un acceso inmediato para, a través de un índice, acceder a cualquier elemento del array.

Cualquier elemento en un array es solo una variable. En otras palabras, uno de los ocho tipos primitivos o una variable de referencia. Cualquier cosa que puedas pones en una variable de esos tipos puede ser asignado a un elemento del array de ese mismo tipo. Entonces en un array de de tipo int (int[ ]), cada elemento puede tomar un int. En un array de Dog (Dog[ ]) cada elemento puede tomar…un Dog? No, recuerda que una varaible de referencia solo toma una referencia (un control remoto), no el objeto en sí. Entonces en un array de Dog, cada elemento puede tomar el control remoto de un Dog. Por supuesto, aún podemos crear objetos Dog…y lo verás enseguida!

Asegúrate de ver bien algo clave en la figura de arriba – el array es un objeto, a pesar que se trata de un serie de primitivos.

Los arrays siempre son objetos, como sea que sean declarados, para tomar primitivos o referencias a objetos. Pero puedes tener un objeto array que sea declarado tomando valores primitivos. En otras palabras, el objeto array puede tomar elementos que son de tipo primitivos pero el array en si nunca será un primitivo.

En pocas palabras, no importa lo que el array tenga…siempre va a ser un objeto!

# Ahora nos toca…un array de Dogs! (¿?)

|  |  |  |  |
| --- | --- | --- | --- |
|  | Declara una variable array de Dog.  Dog[] pets; | |  |
|  | Crea un nuevo array de Dog con una longitud de 7 y asígnalo a la variable pets.  pets = **new** Dog[7];  Qué faltaría? Dogs! Tenemos un array de referencias a Dog, pero no tenemos objetos Dog! |  | |
|  | Crea nuevos objetos Dog y asignalos a un array de elementos.  Recuerda, los elementos en el array de Dog son solo referencias a variables. Aún necesitamos Dogs!  pets[0] = **new** Dog();  pets[l] = **new** Dog(); |  | |

# Controlando al Dog (con una variable de referencia)

|  |  |
| --- | --- |
|  |  |
| Dog fido = **new** Dog();  fido.name = "Fido";  Crea un objeto Dog y usa el operador punto en la variable de referencia fido para acceder a las variables de la variable fido.\*  Puedes usar a fido como referencia para invocar al método bark() o eat() o chaseCat():  fido.bark()  fido.chaseCat() |  |

# Un arreglo de Dogs

Sabes que puedes acceder a las variables de una instancia de Dog y a sus métodos usando el operador punto (.), pero sobre qué?

Cuando un Dog está en un array, no tenemos el nombre real de la variable (como pasa con fido). En su lugar usamos la notación de los arreglos y apretamos el botón del control remoto (operador punto) sobre un objeto de un índice (posición) en particular del array:

|  |  |
| --- | --- |
| Dog[] myDogs = **new** Dog[3];  myDogs[O] = **new** Dog() ;  myDogs[O].name = "Fido";  myDogs[O] .bark(); | Si, si, ya lo sé, no uso encapsulación aún. Lo veremos más adelante. |

La clase completa seria:

**public** **class** Dog {

String name;

**public** **static** **void** main(String[] args) {

Dog dog1 = **new** Dog();

dog1.bark();

dog1.name = "Bart";

Dog[] myDogs = **new** Dog[3];

myDogs[0] = **new** Dog();

myDogs[1] = **new** Dog();

myDogs[2] = dog1;

myDogs[0].name = "Fred";

myDogs[1].name = "Marge";

System.*out*.print("last don't name is ");

System.*out*.println(myDogs[2].name);

**int** x = 0;

**while** (x < myDogs.length) {

myDogs[x].bark();

x = x + 1;

}

}

**public** **void** bark() {

System.*out*.println(name + " says Ruff!");

}

**public** **void** eat() {

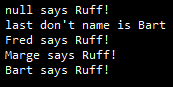
}

**public** **void** chaseCat() {

}

}

Y obtendríamos como salida:



|  |  |
| --- | --- |
| * Las variables vienen en dos tipos: primitivos y de referencia. * Las variables siempre deben declararse con un nombre y un tipo. * El valor de una variable primitiva está representado en los bits que esta tiene (5, 'a', true, 3.1416, etc.). * El valor de una variable de referencia está representado en los bits que indican la forma de llegar a un objeto en el heap. * Una variable de referencia es como un mando a distancia. Usar el operador punto (.) en una variable de referencia es como apretar un botón del control remoto para acceder a un método o una variable de la instancia. * Una variable de referencia tiene un valor nulo cuando no se está referenciando a ningún objeto * Una matriz es siempre un objeto, incluso si la matriz contiene primitivos. No existe un array primitivo, sólo una matriz que tiene valores primitivos. | C:\Users\ingas\Desktop\Atencion2.png |

# Sé el compilador

Para seguir ejercitándonos haremos unos pequeños ejercicios donde nos convertiremos (momentáneamente) en el compilador de Java:

|  |  |
| --- | --- |
| **class** Books {  String title;  String author;  }  **class** BooksTestDrive {    **public** **static** **void** main(String [] args){    Books [] myBooks = **new** Books[3];  **int** x = 0;  myBooks[0].title = "The Grapes of Java";  myBooks[1].title = "The Java Gatsby";  myBooks[2].title = "The Java Cookbook";  myBooks[0].author = "bob";  myBooks[1].author = "sue";  myBooks[2].author = "ian";    **while** (x < 3) {  System.*out*.print(myBooks[x].title);  System.*out*.print(" by ");  System.*out*.println(myBooks[x].author) ;  x = x + 1;  }  }  } |  |
| **class** Hobbits {  String name;  **public** **static** **void** main(String[] args) {  Hobbits[] h = **new** Hobbits[3];  **int** z = 0;  **while** (z < 4) {  z = z + 1;  h[z] = **new** Hobbits();  h[z].name = "bilbo";  **if** (z == 1) {  h[z].name = "frodo";  }  **if** (z == 2) {  h[z].name = "Usam";  System.*out*.print(h[z].name + " is a H");  System.*out*.println("Ugood Hobbit name");  }  }  }  } |  |