

**دانشکده ریاضی و علوم کامپیوتر**

**برنامه‌سازی پیشرفته و کارگاه**

**شیوه‌نامهٔ داکیومنت‌نویسی**

استاد درس

دکتر مهدی قطعی

استاد دوم

بهنام یوسفی مهر

نگارش

سیدآرمان حسینی

بهار ۱۴۰۳

**فهرست**

[**مقدمه 3**](#_Toc190365716)

[**محتوا 4**](#_Toc190365717)

[Interactive بودن داک‌ها 4](#_Toc190365718)

[سریع به کد برسین 4](#_Toc190365719)

[«چه چیزی یاد گرفتیم؟» 5](#_Toc190365720)

[استفاده از کلمات انگلیسی 6](#_Toc190365721)

[**لحن 7**](#_Toc190365722)

[فارسی روان و خودمونی 7](#_Toc190365723)

[استفاده از ChatGPT 7](#_Toc190365724)

[**ظاهر 9**](#_Toc190365725)

[برای headerها از word استفاده کنین 9](#_Toc190365726)

[به template وفادار باشین 10](#_Toc190365727)

[کد‌ها و کامندها 10](#_Toc190365728)

[فاصلهٔ بین پاراگراف‌ها 11](#_Toc190365729)

[متن رو justify کنید 12](#_Toc190365730)

[فهرست 12](#_Toc190365731)

[خیلی زیر عکس‌ها چیزی ننویسید 13](#_Toc190365732)

[از bullet pointها استفادهٔ بی‌جا نکنید 13](#_Toc190365733)

[همهٔ متن راست‌چین شده باشه 14](#_Toc190365734)

[اسم‌گذاری 14](#_Toc190365735)

[**چک و تیک‌های نهایی 15**](#_Toc190365736)

مقدمه

توی این داکیومنت و چند داکیومنت بعدی، قراره با شی‌گرایی آشنا بشیم. شی‌گرایی یا object-oriented programming (به اختصار OOP)، به یک طرز فکر برای تقسیم یک برنامهٔ بزرگ، به واحدهای کوچیک‌تری به اسم class می‌گن. چیزهایی که قبلا بهتون گفته بودیم «این‌ها رو در آینده بهتون توضیح می‌دیم» رو قراره این‌جا بهتون توضیح بدیم!

در طول خوندن این داکیومنت، اگر سوالی داشتین حتما از تدریس‌یارهاتون یا ChatGPT بپرسین، چون که این داکیومنت یکی از مهم‌ترین داکیومنت‌های شماست که توی هفته‌های آینده بر پایهٔ اون به بررسی مفاهیم دیگه‌ای مثل ارث‌بری، کپسوله‌سازی و مباحث مشابه می‌پردازیم.

کلاس‌ها

اول از همه، بیاید یه کلاس کوچولو با هم درست کنیم. با این که شما هنوز نمی‌دونید دقیقا کلاس چیه، ولی این کدها بهتون یه شهود کوچیکی می‌دن راجع بهش (خصوصا اگر هنوز یه سری چیز راجع به structها توی C یادتون مونده باشه). کد زیر رو توی فایل Main.java کپی کنید:

public class Main {  
 public static void main(String[] args){  
 System.*out*.println("Hello World");  
 }  
}  
  
class Car {  
 public int speed;  
 public int yearOfManufacture;  
 public String model;  
}

این کد اولین کدیه که توش جرئت کردیم بیرون از Main هم کد بزنیم! این کار رو کردیم، تا اولین کلاسمون رو تعریف کنیم، اسمش هم گذاشتیم Car. این کلاس، بناست که نمایندهٔ یک ماشین واقعی در کد ما باشه. از یک ماشین برای ما، سرعتش، سال تولیدش و مدلش مهمه و کلاس car هم شامل همچین چیزاییه. در ادامه، ما یک ماشین توی کدمون درست می‌کنیم:

public static void main(String[] args){  
 Car c = new Car();  
  
 c.speed = 20;  
 c.yearOfManufacture = 2018;  
 c.model = "Samand";  
  
 System.*out*.println("Here is out car: ");  
 System.*out*.println("\tSpeed: " + c.speed);  
 System.*out*.println("\tYear of Manufacture: " + c.yearOfManufacture);  
 System.*out*.println("\tModel: " + c.model);  
}

سرعت این ماشین، ۲۰ کیلومتر بر ساعته. سال ۲۰۱۸ درست شده و مدلش هم سمنده. این کد رو اجرا کنید و خروجی‌ش رو ببینید. حالا یه متد کوچیک به Car اضافه می‌کنیم:

class Car {  
 public int speed;  
 public int yearOfManufacture;  
 public String model;  
  
 public void accelerate() {  
 speed += 5;  
 }  
}

این متد، باعث می‌شه که سرعت ماشین ۵ کیلومتر زیادتر بشه. بیاین تا سرعت ماشینی که درست کردیم رو باهاش بیشتر کنیم:

public static void main(String[] args){  
 Car c = new Car();  
  
 c.speed = 20;  
 c.yearOfManufacture = 2018;  
 c.model = "Samand";  
  
 System.*out*.println("Here is out car: ");  
 System.*out*.println("\tSpeed: " + c.speed);  
 System.*out*.println("\tYear of Manufacture: " + c.yearOfManufacture);  
 System.*out*.println("\tModel: " + c.model);  
  
 for (int i = 0; i < 5; i++) {  
 System.*out*.println("Accelerating...");  
 c.accelerate();  
  
 System.*out*.println("Current speed: " + c.speed);  
 }  
}

می‌بینید که سرعت این ماشین، پنج‌تا پنج‌تا زیاد می‌شه تا نهایتا به ۴۵ کیلومتر بر ساعت می‌رسه.

شما اولین کلاستون رو ساختین، و احتمالا کلی سوال براتون پیش اومده که کلاس چیه، چجوری ازش استفاده می‌شه و این که اصلا چه کار کردیم! بیاین بریم تا با هم یه تعریف منسجم‌تر از کلاس ببینیم.

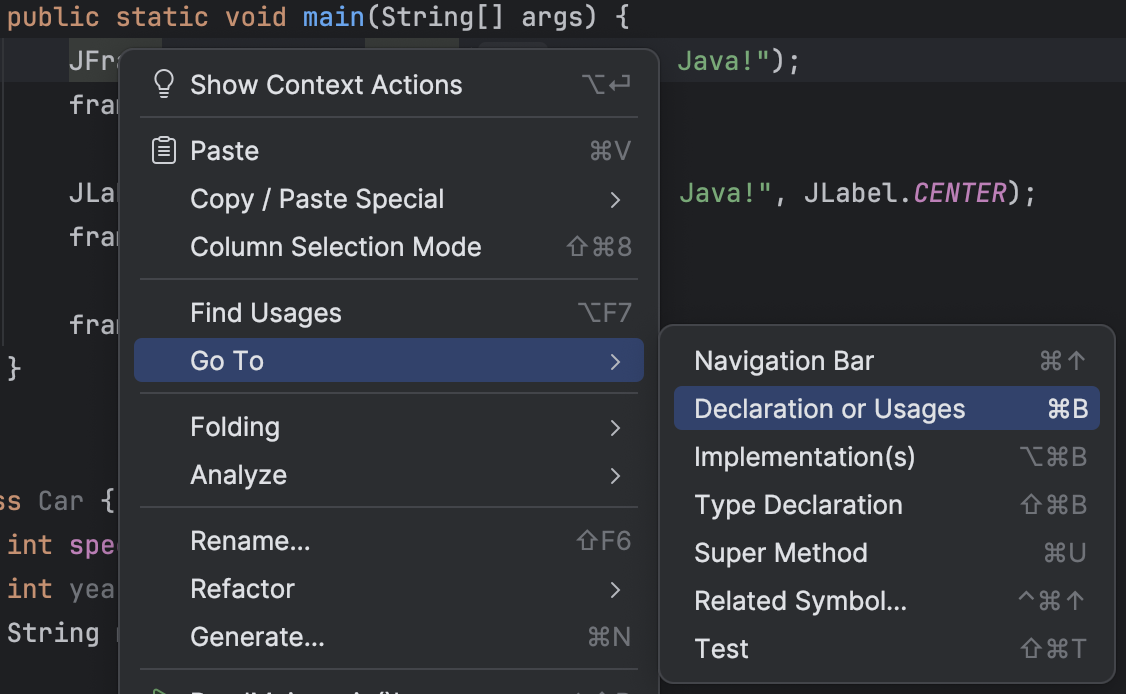
کلاس‌ چیه؟

کلاس‌ها، مجموعه‌ای از متغیرها و متدهای مرتبط با همن که خیلی وقت‌ها، نمایندهٔ یک «چیز» واقعی‌ان. این «چیز» می‌تونه هر چیزی باشه، از یه جسم واقعی مثل ماشین و آدم و کوه گرفته تا یه دکمه روی صفحه.

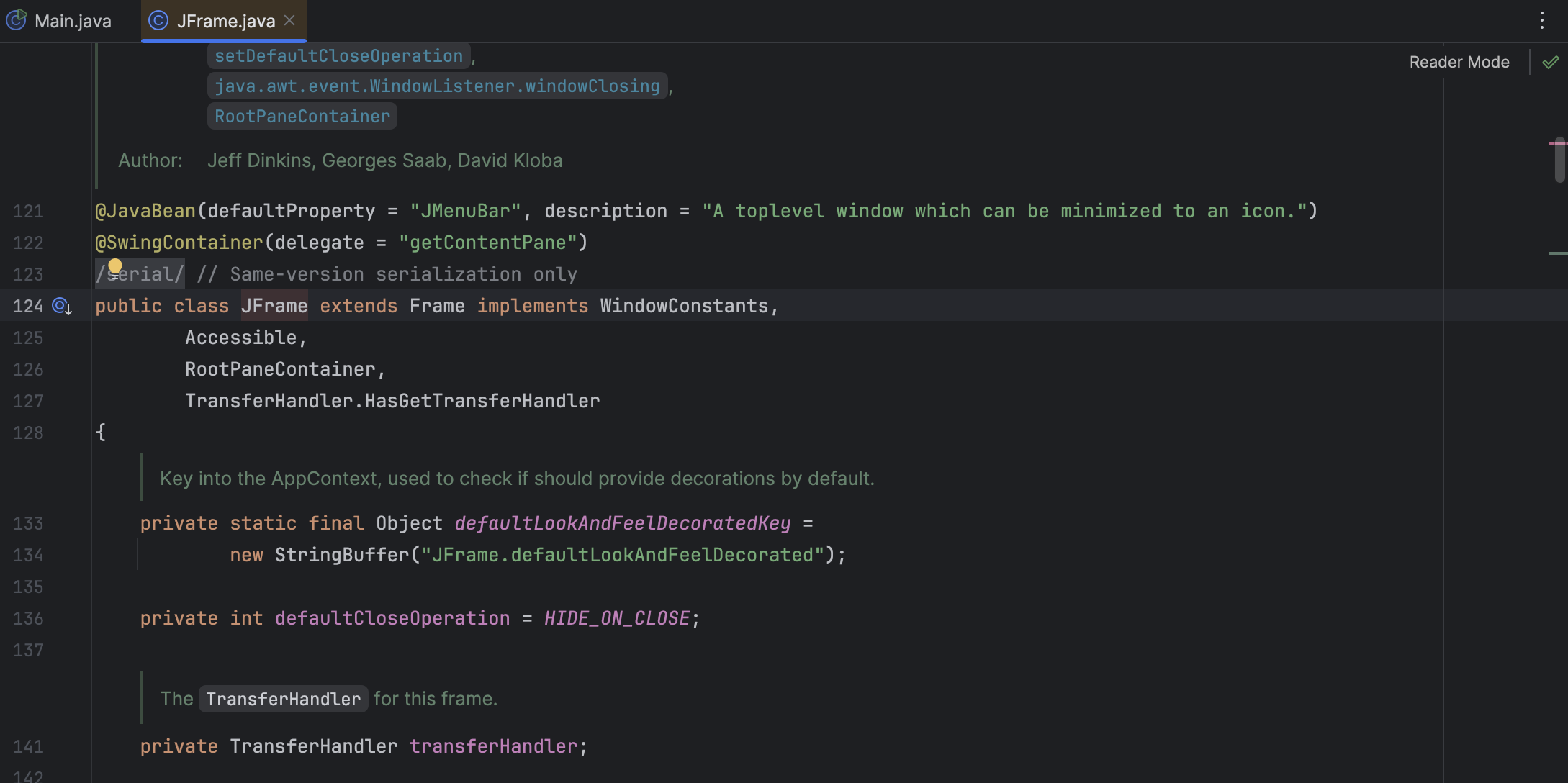
در واقع، شما حتی تا همین الآن هم داشتین از کلاس‌ها استفاده می‌کردین، بدون این که به اون آگاه باشین! کد زیر رو به جای کد Main کپی کنید:

public class Main {  
 public static void main(String[] args) {  
 JFrame frame = new JFrame("Hello, Java!");  
 frame.setSize(300, 150);  
   
 JLabel label = new JLabel("Hello, Java!", JLabel.*CENTER*);  
 frame.add(label);  
   
 frame.setVisible(true);  
 }  
}

حالا روی اون JFrame خط اول main که قبل از frame نوشته شده کلیک راست کنید، به Go To برین و روی Declaration or Usages کلیک کنید:



جلوی خودتون، همچین صفحه‌ای می‌بینین:



این صفحه، شامل پیاده‌سازی JFrame توی جاواست. یه مقدار پیچیده‌ست، ولی می‌خوام توجه‌تون رو به یه تیکه‌ش جلب کنیم:

public class JFrame

می‌بینید؟ JFrame هم یه کلاسه که نمایندهٔ یک Frame در رابط کاربری جاواست و متشکله از متغیرها و متدهای مرتبط با اون. حتی JButton، JLabel، String و خیلی از چیزهایی که تا الآن استفاده می‌کردین، همگی کلاسن. حتی خود Mainای که تا حالا توی کدتون می‌نوشتید هم یک کلاس بوده:

public class Main

پس کلاس‌ها، در عین این که یک خورده غریبه به نظر می‌رسن، براتون آشنا هستن. توی همهٔ کدهایی که تا الآن زدین، ردی از کلاس‌ها دیده می‌شده. کلاس‌ها، به شما اجازه می‌دن تا مسئله‌تون رو به بخش‌های کوچک‌تری تقسیم کنید و منطق اون‌ها رو از هم جدا کنید.

تعریف کلاس

تعریف کلاس، کار راحتیه:

class Human {  
 // Everything a human can do  
}

فقط از کلیدواژهٔ کلاس استفاده کنید و اسم کلاستون رو بنویسید. هر چیزی که بین دو براکت میاد، متعلق به اون کلاسه.

با این کار، شما یک type جدید مثل int، float، String، JFrame و امثال اون‌ها تعریف کردین که می‌تونین توی جاهای مختلف برنامه‌تون، variableهایی از جنس اون درست کنید.

ایجاد Object

با نوشتن کلاس Human، شما یک type جدید به برنامه‌تون اضافه کردین. حالا وقتشه که از اون استفاده کنید:

public class Main {  
 public static void main(String[] args) {  
 Human gholi;  
 Human mamad;  
 Human raees;  
 }  
}

قلی، ممد و رئیس همگی متغیرهایی از جنس Human هستن. به اون‌ها یک instance یا object از کلاس Human می‌گن. دقت کنید با این که تعریف یک آدم توی کلاس Human مشخص شده، ولی تا قبل از اضافه کردن قلی و ممد و رئیس، آدمی توی کد شما وجود نداشته.

شما می‌تونید با استفاده از کلیدواژهٔ new، به این متغیرها مقدار بدین:

Human gholi = new Human();  
Human mamad = new Human();  
Human raees = new Human();

با این کار، contructor کلاس Human صدا زده می‌شه. در ادامه راجع به این که constructor چیه صحبت می‌کنیم.

propertyهای کلاس

propertyها، یا متغیرهای یک کلاس، ویژگی‌های اون کلاس رو نشون می‌دن. مثلا اگر ما بخوایم آدم‌ها توی برنامه‌مون سن و اسم و شماره ملی داشته باشن، می‌تونیم این کار رو با اضافه کردن چندتا متغیر به کلاس Human انجام بدیم:

class Human {  
 public int age;  
 public String name;  
 public String nationalID;  
}

متغیرهای یک کلاس، به همین شکلی که این‌جا می‌بینید تعریف می‌شن. برای الآن، لطفا قبل از همهٔ ‌متغیرهای یک کلاس عبارت public رو بذارید، توی داک‌های بعدی OOP مفهوم اون رو بهتون توضیح می‌دیم و بهتون می‌گیم که در چه مواقعی می‌تونید از شرش راحت بشید.

حالا، به کد قبلی برگردیم و سن و اسم و کد ملی قلی رو تعیین کنیم:

gholi.age = 21;  
gholi.name = "Gholi";  
gholi.nationalID = "1234567890";  
  
System.*out*.println("Gholi:");  
System.*out*.println("\tName: " + gholi.name);  
System.*out*.println("\tAge: " + gholi.age);  
System.*out*.println("\tNationalID: " + gholi.nationalID);

شما به استفاده از یه نقطه، و در ادامه‌ش اسم propertyای که می‌خواین، می‌تونید به اون property دسترسی داشته باشید. اگر این کد رو اجرا کنید، مشخصات قلی رو می‌بینید:

Gholi:  
 Name: Gholi  
 Age: 21  
 NationalID: 1234567890

حالا با کد زیر، بدون این که اسم و مشخصات ممد رو مشخص کنید، اون‌ها رو چاپ کنید:

System.*out*.println("Mamad:");  
System.*out*.println("\tName: " + mamad.name);  
System.*out*.println("\tAge: " + mamad.age);  
System.*out*.println("\tNationalID: " + mamad.nationalID);

خروجی‌تون به این شکله:

Mamad:  
 Name: null  
 Age: 0  
 NationalID: null

می‌بینید با این که شما مقداری به این propertyها ندادین، ولی جاوا براشون یه مقدار دیفالت گذاشته. مقدار متغیرهای عددی توی جاوا ۰ئه، به همین خاطر شما می‌بینید که سن ممد صفر در نظر گرفته شده. مقدار دیفالت objectها توی جاوا nullئه که یه مقدار جلوتر باهاش آشنا می‌شین، و چون name و nationalID از جنس کلاس Stringان، مقدار دیفالت اون‌ها هم null هستش.

شما، می‌تونید این مقادیر دیفالت رو به شکل زیر عوض کنید:

class Human {  
 public int age = 20;  
 public String name = "Unknown";  
 public String nationalID;  
}

حالا اگر دوباره برنامه‌تون رو اجرا کنید، می‌بینید که سن و اسم و کد ملی ممد تغییر کردن:

Mamad:  
 Name: Unknown  
 Age: 20  
 NationalID: 1111111111

Constructorها

کلاس زیر رو به برنامه‌تون اضافه کنید:

class Animal {  
 public String name;  
 public String kind;  
  
 Animal(String animalName, String animalKind) {  
 System.*out*.println("Constructor called.");  
  
 name = animalName;  
 kind = animalKind;  
 }  
}

این کلاس، نمایندهٔ یک Animal توی برنامهٔ شماست. ولی یه چیزی به اون اضافه شده، یه متد عجیب غریب هم اسم کلاس که خروجی‌ای هم نداره. به اون، Constructor می‌گن و وظیفه‌ش اینه که با ورودی گرفتن اسم و نوع یک حیوان، propertyهای name و kind رو مقداردهی کنه.

حالا کد زیر رو به تابع mainتون اضافه کنید:

Animal cat = new Animal("cat", "mammal");  
  
System.*out*.println("Cat:");  
System.*out*.println("\tName: " + cat.name);  
System.*out*.println("\tKind: " + cat.kind);

خروجی زیر رو می‌بینید:

Constructor called.  
Cat:  
 Name: cat  
 Kind: mammal

از روی اون Constructor called توی خط اول خروجی، می‌شه فهمید که تابع constructorای که ما برای Animal نوشته بودیم صدا زده شده و مقادیر name و kind مقداردهی شدن. حالا کانستراکتور زیر و هم به animal اضافه کنید:

Animal(String animalName) {  
 System.*out*.println("Here's another constructor");  
   
 name = animalName;  
 kind = "unknown";  
}

و با استفاده از کد زیر، از این constructor استفاده کنید:

var Panda = new Animal("panda");  
  
System.*out*.println("Panda:");  
System.*out*.println("\tName: " + Panda.name);  
System.*out*.println("\tKind: " + Panda.kind);

می‌بینید که خروجی، همچین چیزیه:

Here's another constructor  
Panda:  
 Name: panda  
 Kind: unknown

همون‌طور که می‌بینید، کانستراکتور دومتون صدا زده شده. یعنی می‌شه برای هر کلاس، چند constructor جدا داشت.

ولی یه چیزی، کلاس Human رو یادتون میاد؟ ما برای اون، Constructorای تعریف نکرده بودیم، ولی با این حال، همچنان می‌تونستیم با عبارت زیر یک object از اون بسازیم:

Human gholi = new Human();

این به خاطر اینه که اگر کلاسی، کانستراکتور نداشته باشه، خود جاوا یه constructor خالی بدون هیچ پارامتری براش درست می‌کنه. شما می‌تونید یه constructor هم برای کلاس Human درست کنید:

class Human {  
 public int age;  
 public String name;  
 public String nationalID;  
   
 Human() {  
 age = 10;  
 name = "arman";  
 nationalID = "unknown";  
 }  
}

به این constructor، که جاوا وقتی خودتون چیزی تعریف نکنید درست می‌کنه، default constructor می‌گن.

متغیرهای static

گفتیم که ویژگی‌های هر کلاس، از طریق propertyهای اون مشخص می‌شن. ولی خب، بعضی ویژگی‌ها هستن که متعلق به هیچ instanceای از اون کلاس نیستن، ولی باز هم متعلق به اون کلاسن.

مثلا، وقتی راجع به آدم‌ها حرف می‌زنیم، هر شخص، سن و اسم و جنسیت و کد ملی خودش رو داره، پس این‌ها ویژگی‌های instanceهای مختلف کلاس آدمن و باید به عنوان property، توی اون کلاس بیان. ولی «جمعیت تمام آدم‌های جهان» چی؟ این‌طور نیست که هر شخص همون طور که برای خودش یک اسم داره، یه «جمعیت تمام آدم‌های جهان» هم داشته باشه، ولی خب، این ويژگی‌ هنوز به کلاس آدم شبیهه.

به این property‌ها، که متعلق به یک عضو مشخص از یک کلاس نیستن و متعلق به خود اون کلاسن، static گفته می‌شه. اون‌ها رو می‌شه به شکل زیر نمایش داد:

class Human {  
 public static int *populationOfAllHumans* = 0;  
   
 public int age;  
 public String name;  
 public String nationalID;  
   
 Human() {  
 age = 10;  
 name = "arman";  
 nationalID = "unknown";  
   
 *populationOfAllHumans*++;  
}

}

توی تکه کد بالا، populationOfAllHumans یک متغیر staticئه و با هر بار صدا زدن constructor این کلاس و ساخته شدن یک آدم جدید، یکی زیاد می‌شه. کد زیر، نشون می‌ده که چطور این مقدار تغییر می‌کنه:

public static void main(String[] args) {  
 System.*out*.println("Human population: " + Human.*populationOfAllHumans*);  
  
 Human gholi = new Human();  
 System.*out*.println("Human population: " + Human.*populationOfAllHumans*);  
  
 Human mamad = new Human();  
 System.*out*.println("Human population: " + Human.*populationOfAllHumans*);  
  
 Human raees = new Human();  
 System.*out*.println("Human population: " + Human.*populationOfAllHumans*);  
}

اگر این کد رو اجرا کنید، خروجی اون به این صورته:

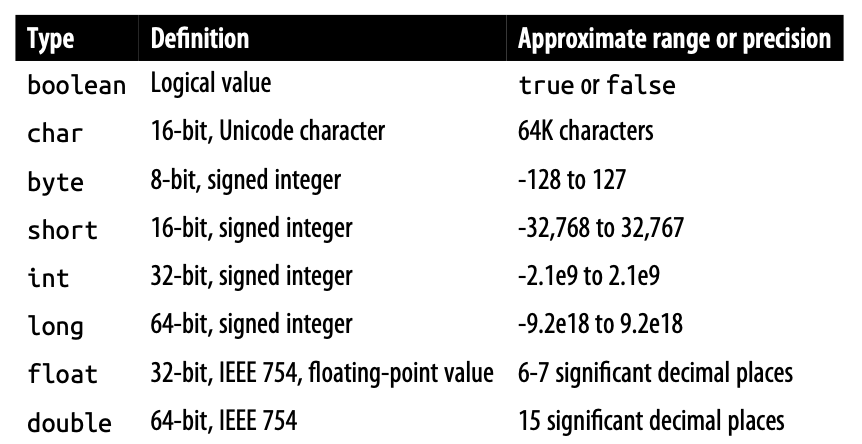
Human population: 0  
Human population: 1  
Human population: 2  
Human population: 3

می‌بینید که با توجه به این که متغیر populationOfAllHumans مربوط به هیچ کدوم از آبجکت‌های کلاس Human نیست، با استفاده از Human.populationOfAllHumans به اون دسترسی پیدا کردیم. هر چند می‌تونستیم از طریق instanceها هم این کار رو انجام بدیم و باز هم همین خروجی رو بگیریم:

public static void main(String[] args) {  
 System.*out*.println("Human population: " + Human.*populationOfAllHumans*);  
  
 Human gholi = new Human();  
 System.*out*.println("Human population: " + gholi.*populationOfAllHumans*);  
  
 Human mamad = new Human();  
 System.*out*.println("Human population: " + mamad.*populationOfAllHumans*);  
  
 Human raees = new Human();  
 System.*out*.println("Human population: " + raees.*populationOfAllHumans*);  
}

Reference typeها

همون‌طور که از جلسهٔ صفر می‌دونید، توی جاوا، typeها به دو دستهٔ primitive type و reference type تقسیم‌بندی می‌شن. primitive typeها، تایپ‌های بسیار ساده‌ای مثل int، char، boolean و امثال اون‌ها هستن. فهرست کامل اون‌ها توی لیست زیر اومده:



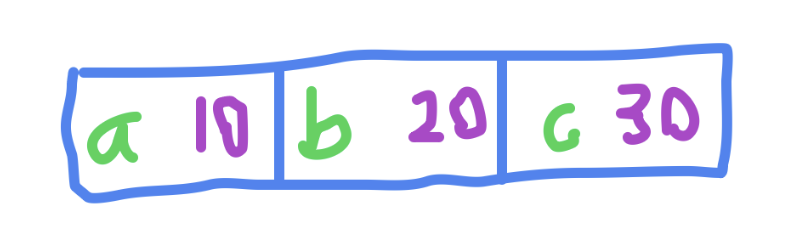
هر تایپ دیگه‌ای توی جاوا، reference typeئه. String، JFrame، ArrayList و حتی تایپ‌هایی مثل Human و Car که تا این‌جای کار تعریف کردیم، همگی reference typeان. هر reference typeای با یک کلاس تعریف شده.

تفاوت reference typeها و primitive typeها

کد زیر رو در نظر بگیرین، توی امتد main سه‌تا متغیر از جنس primitive type تعریف و مقداردهی شدن:

public static void main(String[] args) {  
 int a, b, c;  
 a = 10;  
 b = 20;  
 c = 30;  
}

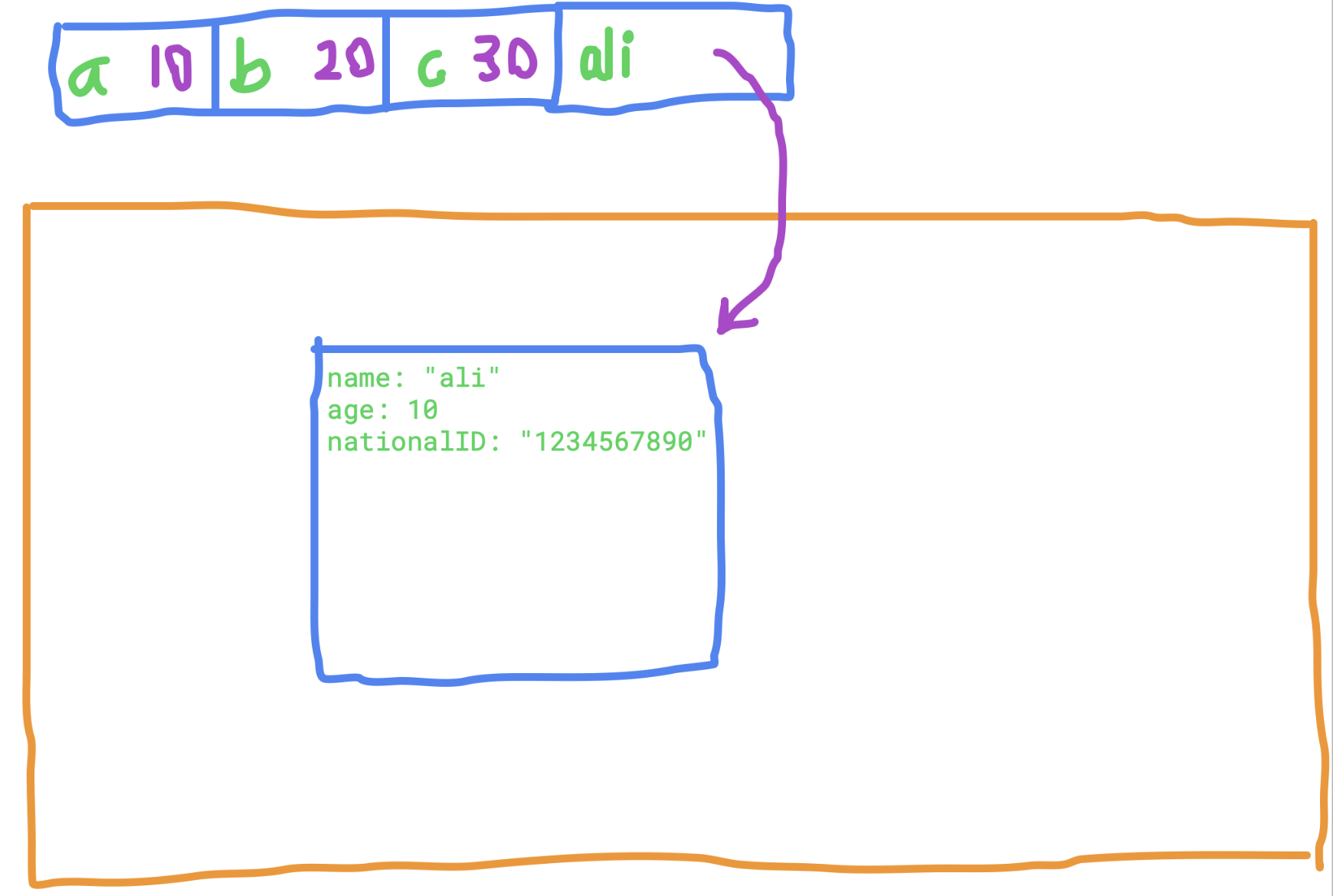
شما می‌تونید متصور باشید که توی مموری‌تون، این متغیرها به شکل زیر نگه‌داری می‌شن:



حالا فرض کنید که توی main، یه متغیر دیگه هم از جنس reference type تعریف کردیم:

Human ali = new Human();

این متغیر، به شکل خاصی ذخیره می‌شه. مقدارید propertyهای علی توی جای دیگه‌ای از مموری ذخیره می‌شه و توی خود متغیر ali، آدرس اون جا ذخیره می‌شه:



اگر الآن ali رو به شکل زیر چاپ کنید می‌تونید این موضوع رو ببینید:

System.*out*.println(ali);

خروجی این کد، به شکل زیره:

Human@6acbcfc0

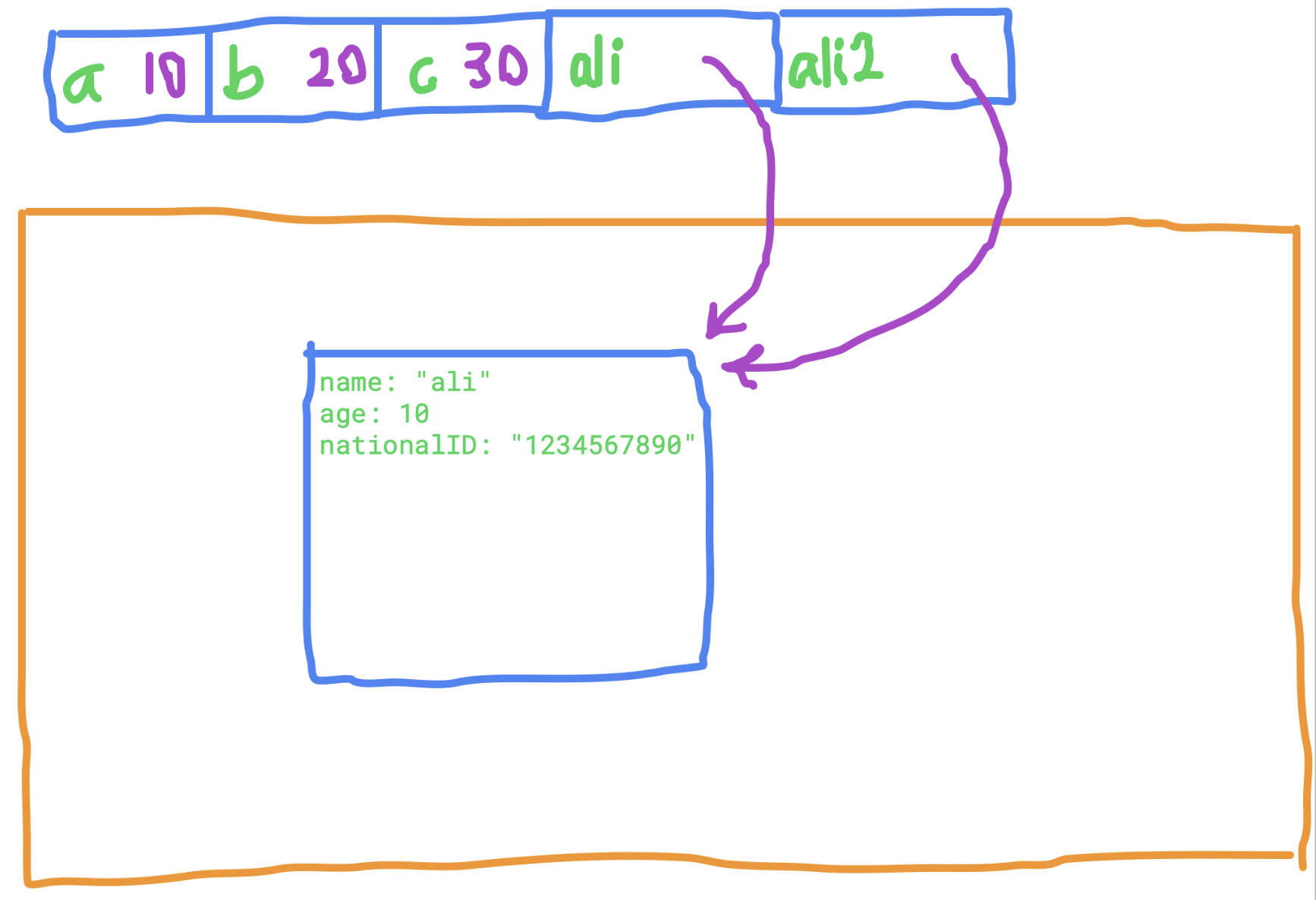
اون 6acbcfc0، آدرسیه که توی اون، propertyهای ali ذخیره شدن. حالا با استفاده از کد زیر، متغیر ali رو داخل یک متغیر دیگه بریزید:

Human ali2 = ali;  
System.*out*.println(ali2);

می‌بینید که ali2 هم دقیقا به همون نقطه از مموری‌تون اشاره می‌کنه:

Human@6acbcfc0

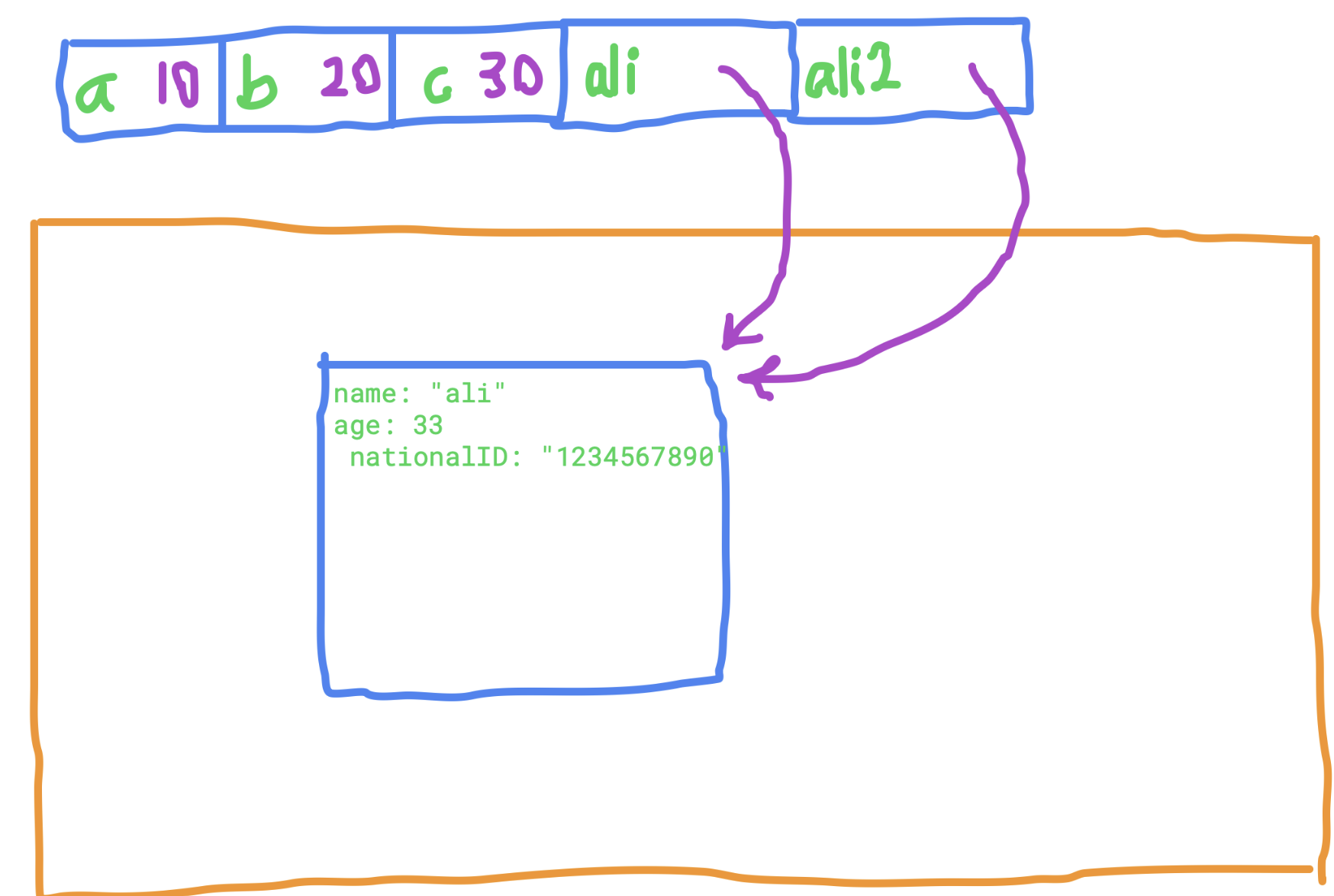
پس الآن حافظه به همچین شکلیه:



این که هر دوی ali و ali2 توی حافظه به یک جا اشاره می‌کنن، یک ساید افکت جالب داره. کد زیر رو اجرا کنید:

ali.age = 33;  
System.*out*.println(ali.age);  
System.*out*.println(ali2.age);

می‌بینید با این که شما فقط سن علی رو به ۳۳ تغییر دادین، ولی سن ali2 هم به ۳۳ تغییر کرده، این اتفاق عجیب به نظر می‌رسه تا این که ببینید که مموری چجوری سن علی رو به ۳۳ تغییر داده:



می‌بینید که چون ali و ali2 هر دو به یک جا اشاره می‌کردن، سن هر دوی اون‌ها واقعا 33 شده.

همین‌ اتفاق، وقتی که reference typeها رو به عنوان ورودی به یک متد می‌دین هم می‌افته:

public class Warrior {

double power;  
 double hp;  
  
 public void hit(double multiplier, Warrior enemy) {  
 if (multiplier >= 100) {  
 multiplier = 100;  
 }  
 double finalPower = power \* multiplier;  
 enemy.hp -= finalPower;  
 }  
  
 public static void main(String[] args) {  
 Warrior warrior1 = new Warrior();  
 warrior1.hp = 100;  
 warrior1.power = 100;  
  
 Warrior warrior2 = new Warrior();  
 warrior2.hp = 200;  
 warrior2.power = 50;  
  
 double m = 1.5;  
 warrior1.hit(m, warrior2);  
 }  
}

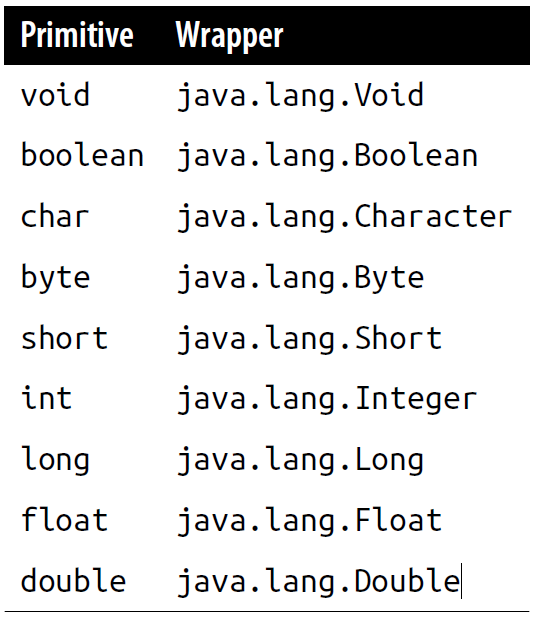
اینجا یک کلاس Warrior ( جنگجو ) داریم که قدرت ( power ) و جون ( hp ) داره. یک متد hit هم تعریف کردیم براش که بتونه به دشمنش حمله کنه! در ادامه، دو تا جنگجو ساختیم و متد hit رو روی warrior1 صدا زدیم و m و warrior2 رو بهش ورودی دادیم. وقتی متد رو صدا زدیم، مقدار m توی یک متغیر محلی توی متد به اسم multiplier کپی میشه و ازش استفاده میشه. متد ما هر تغییری توی multiplier ایجاد کنه، این تغییرات محلی می­مونن و m تغییری نمی­کنه. خودتون امتحان کنید! توی متد اگر multiplier بیشتر از 100 باشه، مقدارش به 100 کاهش پیدا می­کنه ( جنگجو ها نباید زیادی قوی بشن! ). m رو بیشتر از 100 بذارید و ببینید چی میشه.

در مورد ورودی دوم، جاوا یک "ارجاع" ( reference ) به آبجکتی که توی warrior2 قرار داره رو توی متغیر محلی enemy کپی می­کنه و ازش استفاده می­کنه، ولی چون این فقط یک ارجاعه، enemy و warrior2 هر دو در واقع به یک آبجکت اشاره می­کنند و هر تغییری که از طریق enemy یا warrior2 ایجاد بشه، آبجکت اصلی تغییر می­کنه. خودتون امتحان کنید و ببینید! صدا زدن متد باعث میشه جون warrior2 واقعاً کم بشه. ولی، اگر متد hit، توی متغیر enemy ارجاع به یک آبجکت دیگه رو بریزه، warrior2 تغییری نمی­کنه. یعنی مثلاً اگر خط زیر رو اول متد hit بنویسید، warrior2 دیگه هیچ تغییری نمی­کنه.

enemy = new Warrior();

کلاس های wrapper برای primitive type ها

همونطور که تا الان دیدید، توی جاوا به نوعی تفرقه بین جنس متغیر ها وجود داره: بین primitive type ها و آبجکت ها ( reference type ها ). جاوا این تفرقه و دوگانگی رو می­پذیره تا در مقابل، بهینه باشه.وقتی شما با اعداد سر و کار دارید و روی اونا محاسبات انجام میدید، دلتون می­خواد که محاسباتتون سبک باشه و سریع اجرا بشه. اگر قرار بود برای primitive type ها از آبجکت استفاده بشه، بهینه کردن محاسبات کار سختی می­شد. ولی، گاهی اوقات ما نیاز داریم تا اعداد و مقدار های primitive دیگه رو توی آبجکت ها ذخیره کنیم ( مثال هاش رو توی داک آرایه دیده بودید! ). برای این کار، جاوا برای هر primitive type، یک کلاس wrapper استاندارد داره که می­تونید توی این جدول ببینید:



یک آبجکت از جنس یک wrapper class، یک آبجکت غیر قابل تغییره که مثل یک جور جعبه می­مونه که مقداری که بهش می­دیم رو توی خودش نگه می­داره و بهمون اجازه می­ده که ازش استفاده کنیم. ساخت یک آبجکت wrapper، خیلی ساده­ست؛ دقیقا مثل مقدار دهی primitive type هاست:

Double pi = 3.14;  
Boolean someBoolean = false;

همه ی wrapper های عددی، متد های “value” دارن که با استفاده ازشون می­تونید مقداری که توی آبجکت ذخیره کردید رو به هرکدوم از primitive type های عددی که می­خواهید تبدیل کنید. این متد ها doubleValue()، floatValue()، longValue()، intValue()، shortValue() و byteValue() هستند:

Double size = 32.76;  
double d = size.doubleValue(); // 32.76  
float f = size.floatValue(); // 32.76f  
long l = size.longValue(); // 32L  
int i = size.intValue(); // 32

این کد دقیقاً معادل با اینه که یک متغیر primitive از نوع double رو به انواع دیگه cast کنید.

متداول ترین استفاده برای کلاس های wrapper، وقتیه که می­خواهید یک مقدار primitive رو به یک متدی به عنوان ورودی بدید در صورتی که اون متد فقط آبجکت می­پذیره. برای مثال، متد add مربوط به ArrayList که قبلاً باهاش آشنا شدید، فقط با آبجکت ها کار می­کند و اگر بخواهیم از این متد استفاده کنیم، باید مقدار های primitiveی که داریم رو به آبجکت تبدیل کنیم:

// Manually wrapping an integer  
ArrayList myNumbers = new ArrayList();  
Integer thirtyThree = 33;  
myNumbers.add(thirtyThree);

اینجا ما مقدار 33 رو توی یک آبجکت از نوع Integer قرار دادیم تا بتونیم اون رو به متد add بدیم. از اون طرف، وقتی می­خواهیم مقدار 33 رو از لیست دربیاریم، باید اون رو از Integer به int تبدیل کنیم:

// Manually unwrapping an integer  
Integer theNumber = (Integer)myNumbers.get(0);  
int n = theNumber.intValue(); // 33

البته، همونطور که توی داک آرایه هم دیدید، در واقعیت نیازی نیست که ما این کار ها رو انجام بدیم و می­تونیم مستقیماً 33 رو به متد add بدیم و همینطور مستقیماً 33 رو به صورت int از لیست در بیاریم. دلیل این اتفاق اینه که تمام تبدیل های بالا رو جاوا خودش به صورت اتوماتیک برامون انجام میده. جاوا به این کار autoboxing میگه. این قابلیت جاوا باعث میشه که کد ما کوتاه تر و امن تر بشه. استفاده از کلاس های wrapper تا بخش خیلی زیادی از ما توسط کامپایلر پنهان میشه، اما همچنان داخل کامپایلر داره از این کلاس ها استفاده میشه. اینجا یک مثال دیگه داریم که از autoboxing و generics استفاده می­کنه ( با generics توی هفته های آینده بیشتر آشنا می­شید. )

// Using autoboxing and generics  
ArrayList<Integer> myNumbers = new ArrayList<Integer>();  
myNumbers.add(33);  
int n = myNumbers.get(0);

همونطور که می­بینید، اینجا متغیری از نوع Integer نداریم و همه ی تبدیل ها داره به صورت ضمنی انجام میشه. البته توجه کنید که موقع تعریف کردن ArrayList، داریم بهش می­گیم که مقادیرمون قراره از نوع Integer باشند.