انجمن جاواکاپ تقدیم می‌کند

دوره برنامه‌نویسی جاوا

اشیاء در جاوا
JAVA OBJECTS

صادق علی‌اکبری

حقوق مؤلف

- کلیه حقوق این اثر متعلق به انجمن جاواکاپ است
- بازنشر یا تدریس آن‌چه توسط جاواکاپ و به صورت عمومی منتشر شده است، با ذکر مرجع (جاواکاپ) بلامانع است
- اگر این اثر توسط جاواکاپ به صورت عمومی منتشر نشده است و به صورت اختصاصی در اختیار شما یا شرکت شما قرار گرفته، بازنشر آن مجاز نیست
- تغییر محتوای این اثر بدون اطلاع و تأیید انجمن جاواکاپ مجاز نیست

سرفصل مطالب

- ایجاد اشیاء
- وضعیت اشیاء در حافظه
- زباله‌روب (Garbage Collector)
- ارسال پارامتر به متدها (Parameter Passing)
- بخش‌های مختلف حافظه

یکی از اهداف این جلسه

- ارائه یک شهود سطح پایین از اشیاء در حافظه
- نحوه ذخیره‌سازی اشیاء (Objects) و ارجاع‌ها (References)

- اهمیت شهود سطح پایین در کنار شهود سطح بالا
- شهود سطح بالا: بسیار مهم در طراحی
- شهود سطح پایین: بسیار مهم در پیاده‌سازی

اشیاء در برنامه‌های جاوا

یادآوری

تعریف کلاس
Class Declaration

```
public class Dog {
	private String name;
	public void setName(String n) {
		name = n;
	}
	public void bark(){
		System.out.println("Hop! Hop!");
	}
}
```

```
Dog d = new Dog();
```
ساخت شیء جدید، نمونه‌سازی
Object Creation, Instantiation

```
d.setName("Fido");
```
تغییر حالت (state) شیء

```
d.bark();
```
ارسال پیغادم
Message passing

متغیر d چیست؟

یک شیء

ارجاع به یک شیء
(Reference)

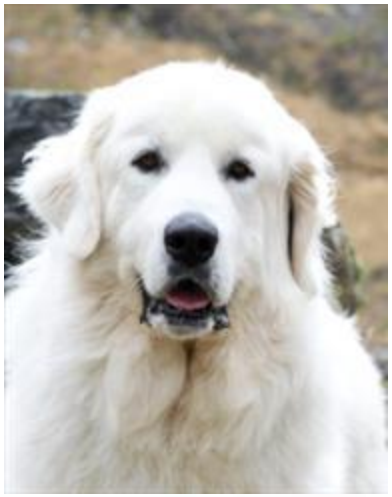
استفاده از یک شیء

object.method(params);

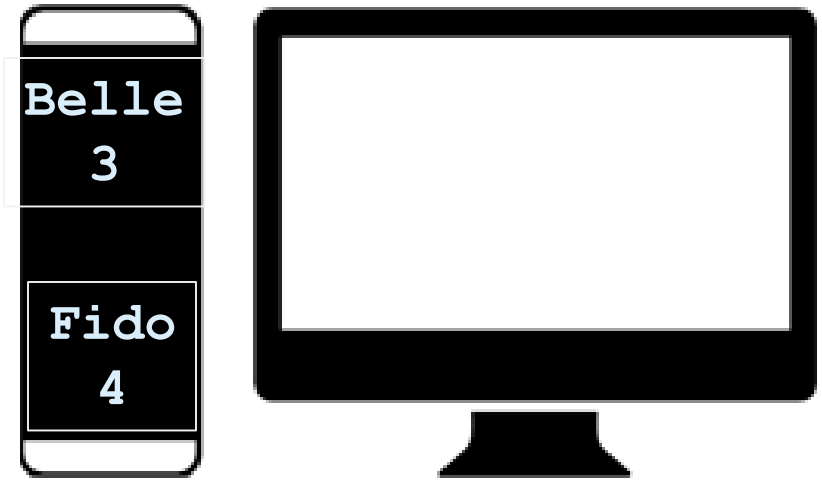
```
Person p = new Person();
p.setName("علی علوی");

Book b1 = new Book();
b1.setTitle("شاهنامه");

p.borrow(b1);

String s = p.name;
p.age = 22;
```

- نقطه (dot):
- قبل از نقطه، اسم شیء می‌آید
- معمولاً بعد از نقطه، یک متد فراخوانی می‌شود
- بعد از نقطه، ممکن است یکی از ویژگی‌های شیء بیاید
  - به شرطی که **در دسترس** باشد
    - مثلاً public باشد

اشیاء در برنامه‌ها



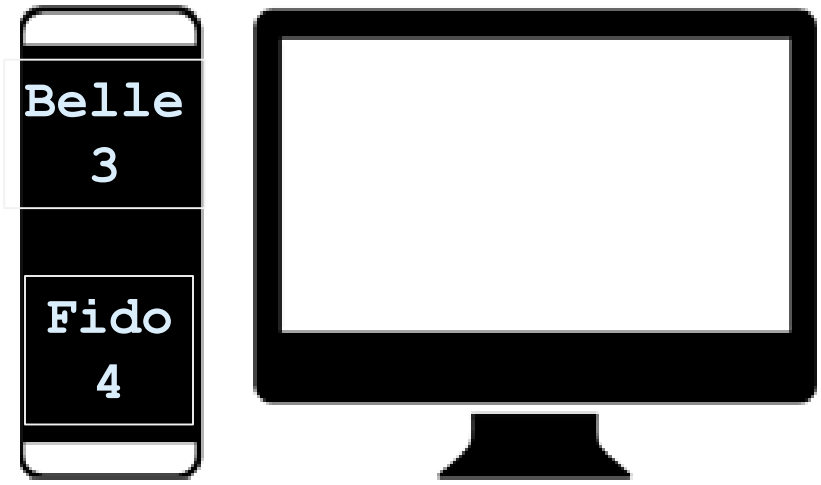
```
Dog d1 = new Dog();
d1.setName("Belle");
d1.setAge(3);
```



```
Dog d2 = new Dog();
d2.setName("Fido");
d2.setAge(4);
```

اشیاء در حافظه

- یادآوری: هر شیء، **حالت**، **رفتار** و **هویت** دارد
- ***state, behavior, identity***



- هر شیء، در حافظه قرار می‌گیرد
- محتوای حافظه = **حالت** شیء
- آدرس شیء در حافظه ≈ **هویت** شیء
- **رفتار** شیء: قبلاً در کلاس شیء تعریف شده

عملگر new

- یک **شیء جدید** از کلاس مشخص شده می‌سازد
- **ارجاع** (**reference**) به شیء ساخته شده را برمی‌گرداند

new String();

new Book();

- `String s = new String();`
- `Dog d = new Dog();`
- `Rectangle r = new Rectangle();`
- `Scanner c = new Scanner(System.in);`
- عملگر new حافظه را در بخشی به نام Heap ایجاد می‌کند

انواع داده‌ها

- در جاوا دو گونه «نوع داده» وجود دارد

۱- **انواع داده اولیه** (Primitive Data Types)

- byte, short, int, long, float, double, boolean, char

```
int number;
char ch;
```

- تعدادی نوع داده محدود و مشخص که در زبان جاوا موجودند
- هر متغیر از این انواع، حاوی یک مقدار است (**نه یک شیء**)

۲- **انواع داده ارجاعی** (Reference Data Types) یک **کلاس**ها

- مثل String ، Scanner ، Dog یا Book

```
String name;
Book book;
```

- برخی (مثل String) در زبان موجود است
- و برخی دیگر (مثل Book) را ما تعریف می‌کنیم
- هر متغیر از این انواع، ارجاعی به یک **شیء** است

تفاوت یک متغیر اولیه (primitive) با یک شیء

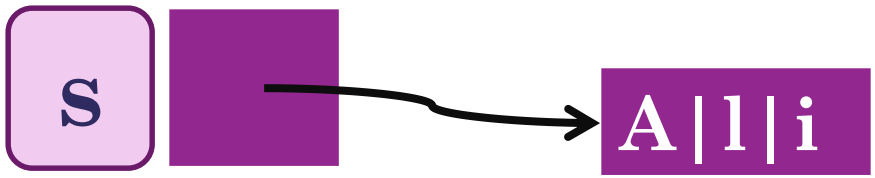
- انواع اولیه (primitive types) امکان استفاده از new را ندارند
  - ارجاعی (Referenced) نیستند

```
new int();
```

  - به جای نگهداری ارجاع به یک شیء، یک مقدار را نگه می‌دارند
- امکان استفاده از dot را ندارند
  - چون شیء نیستند که ویژگی یا رفتار داشته باشند

```
int a = 5;
a.value = ...
a.setValue(...
```

- `String s = new String("Ali");`
- `int a = 5;`



مفهوم ارجاع (Reference)

ارجاع به اشیاء

- ارجاع (Reference) ، مفهومی مانند یک **اشاره‌گر** (Pointer) است
  - تا حدودی مشابه اشاره‌گر در زبان C++
- وقتی یک متغیر تعریف می‌کنید، در واقع یک ارجاع می‌سازید، **نه یک شیء**

String s;

Book b;

- استثنا:
  - متغیرهای انواع داده اولیه
  - این متغیرها ارجاع به یک شیء نیستند، خود مقدار را نگه می‌دارند

ساخت اشیاء

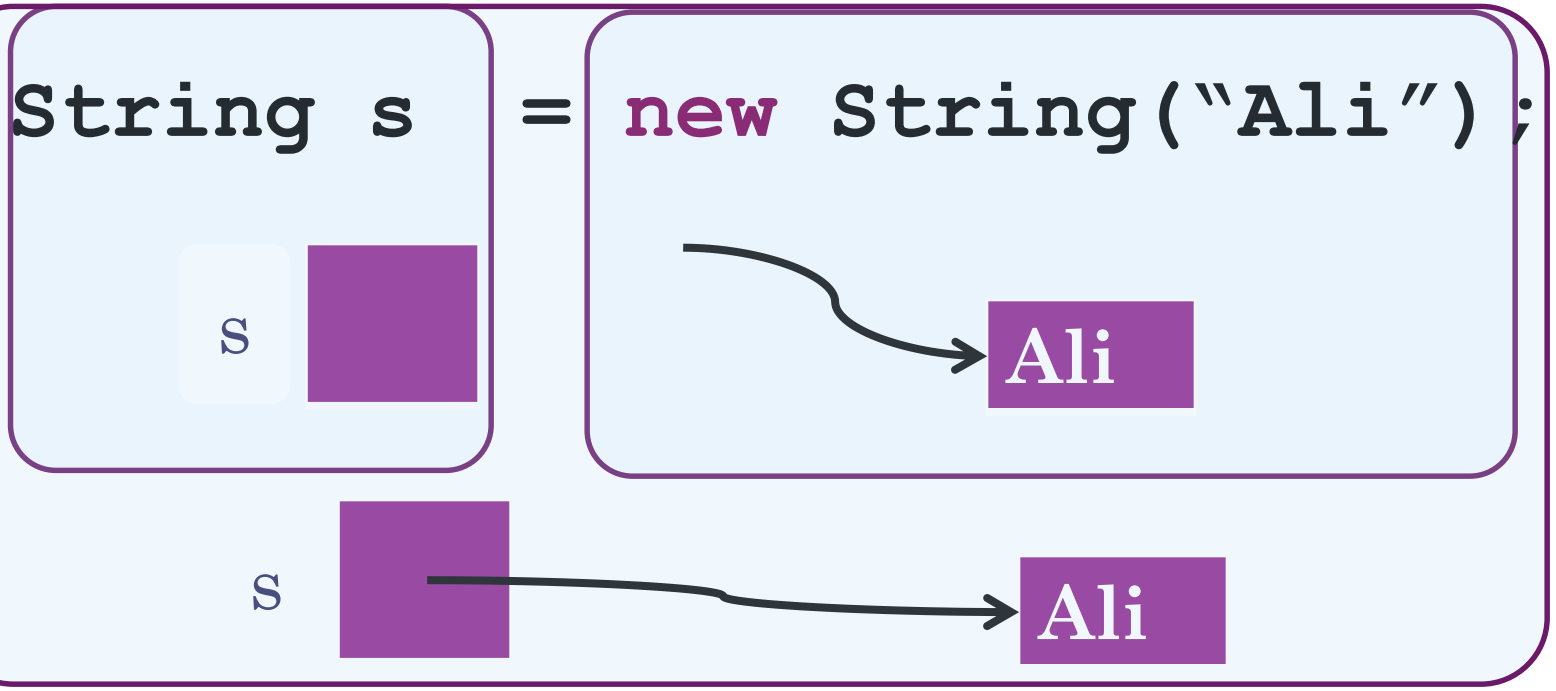
- این کد یک شیء جدید **ایجاد نمی‌کند:** String str;
- فقط یک ارجاع می‌سازد، که هنوز به شیءی ارجاع نمی‌دهد (اشاره نمی‌کند)
- شما فعلاً نمی‌توانید از متغیر str استفاده کنید
- مقدار متغیر str ، خالی (**null**) است
- مقدار **null** : یک ارجاع که به شیءی اشاره نمی‌کند
- هر متغیر (ارجاع)، باید به یک شیء واقعی متصل شود (مگر این که Primitive باشد)
  - به آن اشاره کند، به آن ارجاع دهد
- مثلاً با کمک new به یک شیء جدید اشاره کند

```
str = new String();
str = name;
```

- و یا یک متغیر غیرخالی درون آن ریخته شود (با عملگر =)

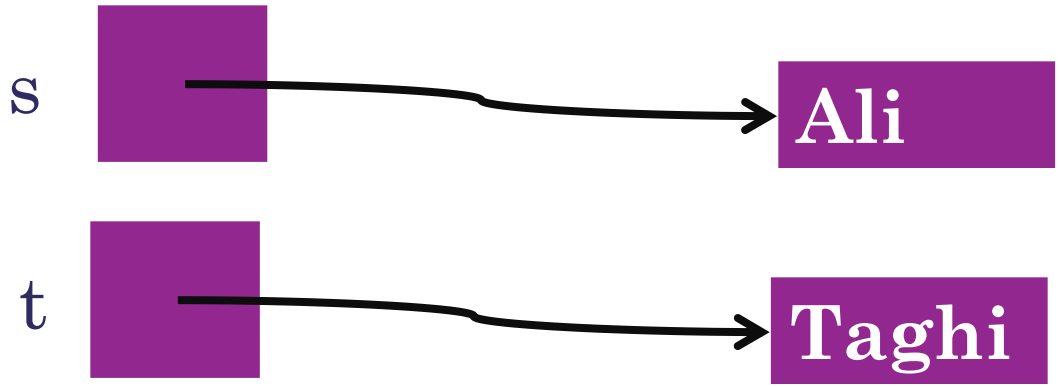
تأثیر عملگر انتساب (مساوی) بر یک ارجاع

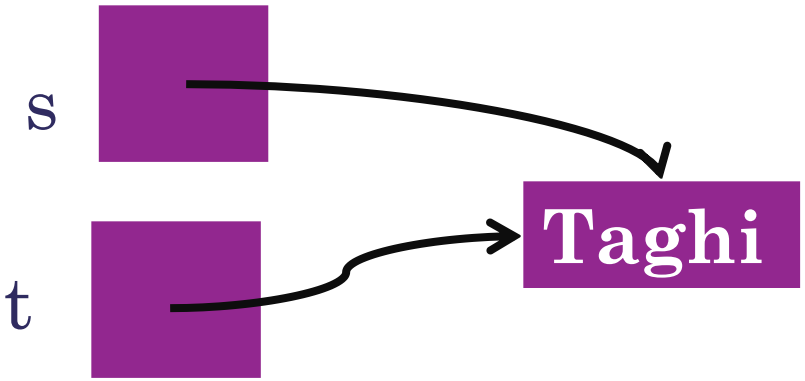
- `String s = new String("Ali");`



- `String t;`
- `t = new String("Taghi");`

تأثیر عملگر مساوی بر یک ارجاع (۲)

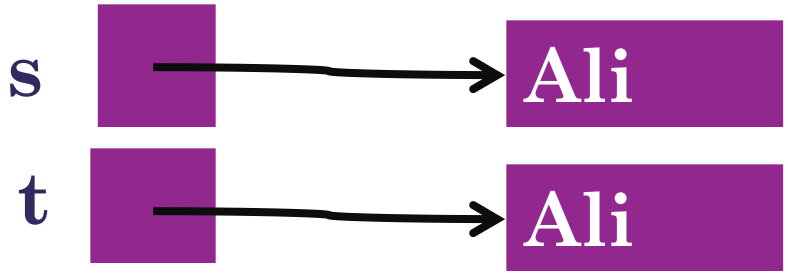




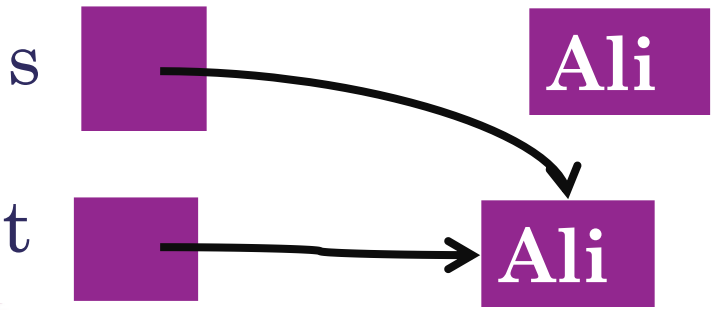
- در این شرایط نتیجه اجرای
  این دستور زیر چیست: `s = t ;`
- دقت کنید: محتوای شیء کپی نمی‌شود
- ارجاع شیء کپی می‌شود (اشاره‌گر کپی می‌شود، آدرس مورد اشاره کپی می‌شود)
- متغیرهای s و t هر دو به یک شیء اشاره می‌کنند

تأثیر عملگر مساوی بر یک ارجاع (۳)

- `String s = new String("Ali");`
- `String t = new String("Ali");`



- متغیرهای s و t به دو شیء متفاوت (متمایز) اشاره می‌کنند
  - هویت این دو شیء متفاوت است (identity)
  - هرچند حالت این دو شیء یکسان است (محتوا، وضعیت یا state)



- با این دستور چه می‌شود: `s=t;`
  - هویت s با هویت t یکی می‌شود

آرایه، به عنوان شیء

آرایه‌ها در جاوا

- هر آرایه در واقع یک شیء است
  - مثل همه اشیاء در بخشی از حافظه به نام Heap ذخیره می‌شود
- می‌توان به آرایه اشاره کرد (ارجاع)

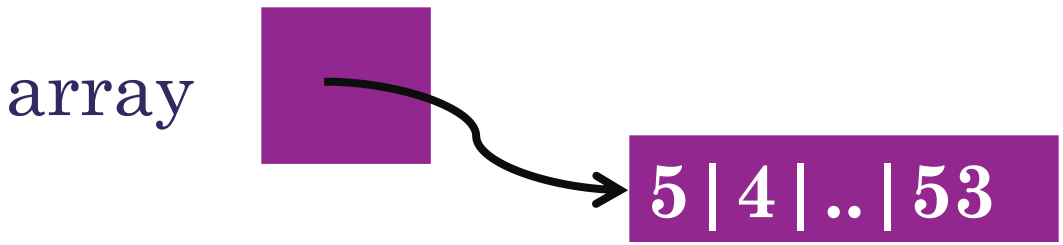
```
String[ ] strs;
Person[ ] people = new Person[5];
int n = …
float[ ] realNumbers = new float[n];
```

- مؤلفه‌های یک آرایه (elements) متغیرهایی از جنس معرفی شده‌اند
  - مثلاً در یک آرایه از جنس String[ ] ، هر مؤلفه **یک ارجاع به یک رشته** است
  - در آرایه‌ای از جنس int[ ] ، هر مؤلفه **یک مقدار int** است

مثال: آرایه‌ای از انواع اولیه

ارجاع به یک شیء از جنس آرایه

```
int[] array = new int[10];
int[][] twoDimArray = new int[5][];
twoDimArray[0] = new int[7];
twoDimArray[1] = new int[12];

int[][] matrix = new int[5][7];
```



مثال از آرایه

```
Scanner scanner = new Scanner(System.in);
int arrayLength = scanner.nextInt();
String[] names = new String[arrayLength];
for (int i = 0; i < names.length; i++) {
    names[i] = scanner.next();
}
System.out.println(names[names.length-1]);
```

- آرایه به عنوان یک شیء، ویژگی length دارد (property)
- که public و قابل استفاده (خواندن) است

آرایه و ارجاعات

```
Scanner scanner = new Scanner(System.in);
String[] array;
array = new String[10];
for (int i = 0; i < array.length; i++) {
  array[i] = scanner.next();
}
```

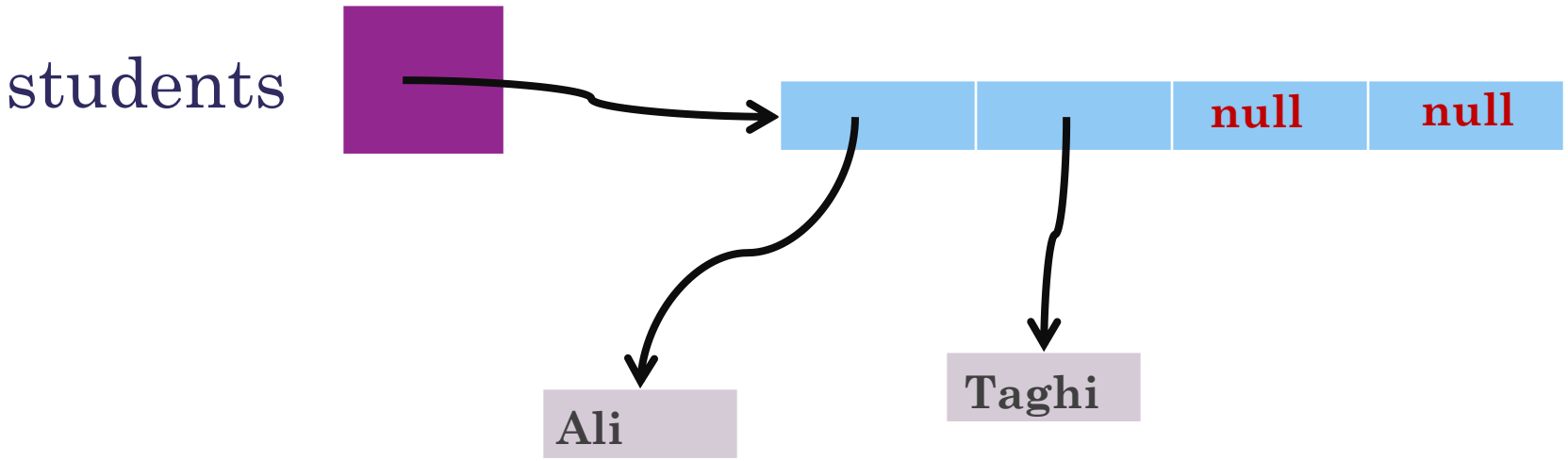
- در این قطعه‌کد، علاوه بر scanner ، چهار نوع مقدار دیده می‌شود:
  - ارجاع آرایه (array)
  - خود شیء آرایه (که با کمک new ایجاد شد)
  - ارجاع مؤلفه‌های درون آرایه (array[i])
    - مقدار اولیه هر مؤلفه: null
  - اشیائی که هر array[i] به یکی از آن‌ها اشاره می‌کند

این کلاس را در نظر بگیرید:

```
public class Student {
  private String name;
  private long id;

  public String getName() {
       return name;
  }
  public void setName(String n) {
       name = n;
  }
  public long getId() {
       return id;
  }
  public void setId(long n) {
       id = n;
  }
}
```

ساخت آرایه از یک نوع دلخواه

```
Student[] students;
students = new Student[4];
students[0] = new Student();
students[0].setName("Ali");
students[1] = new Student();
students[1].setName("Taghi");
```



رشته، به عنوان شیء

رشته در جاوا

- هر رشته (String) در واقع یک شیء است
- هر شیء رشته، متدهای مختلفی دارد (رفتارهای رشته)

```
String input = "Nader and Naser";
char ch = input.charAt(0);
int i = input.indexOf("Naser");
int j = input.lastIndexOf("er");
String newS = input.replace("Nader", "Hamed");
String sth = newS + i ;
System.out.println(sth);
```

یادآوری: متدهای رشته

- charAt
- concat
- startsWith
- endsWith
- indesxOf
- lastIndexOf
- replace
- substring
- length
- equals
- equalsIgnoreCase
- contains

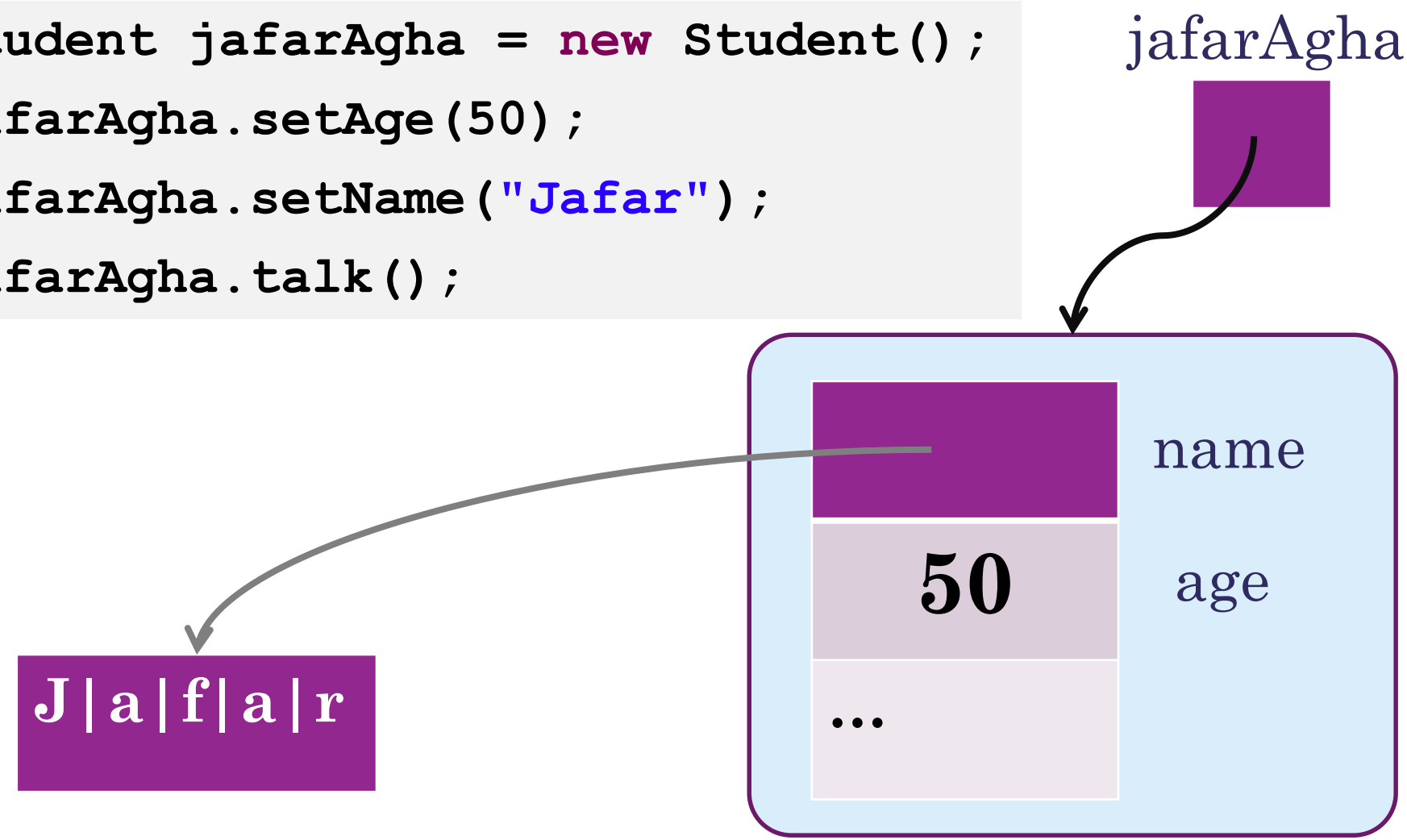
نیم‌نگاهی به کلاس String

- (برخی جزئیات حذف شده‌اند)

```
public class String{
    /** The value is used for character storage. */
  private char value[];
  public int length() {
    return value.length;
  }
  public boolean isEmpty() {
    return value.length == 0;
  }
  public char charAt(int index) {
    return value[index];
  }
  ...
}
```

مثال

```
Student jafarAgha = new Student();
jafarAgha.setAge(50);
jafarAgha.setName("Jafar");
jafarAgha.talk();
```



کوییز

کوییز: نتیجه اجرای برنامه زیر چیست؟

```
public class Student {
    private String name;
    private long ID;

    public String getName() {  return name; }
    public void setName(String name) {this.name = name;}
    public long getID() { return ID; }
    public void setID(long iD) { ID = iD; }

    public static void main(String[] args) {
        Student[] s = new Student[10];
        for(int i=0;i<10;i++){
            s[i].setID(i+10000);
        }
        System.out.println(s[5].getID());
    }
}
```

زباله‌روب (Garbage Collector)

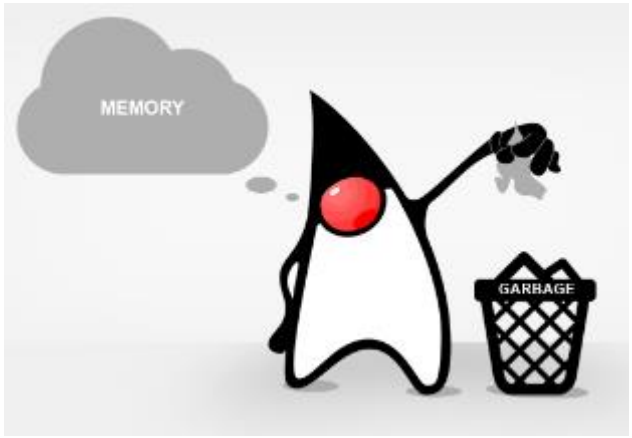
بعد از فراخوانی متد f ، اشیاء ساخته شده چه می‌شوند؟

```
static void f() {
  Scanner scanner = new Scanner(System.in);
  Student[] students = new Student[10];
  for (int i = 0; i < students.length; i++) {
      students[i] = new Student();
      students[i].setId(i);
      students[i].setName(scanner.next() );
  }
}
public static void main(String[] args) {
  f();
  ...
```

آزادسازی حافظه اشیاء

- وقتی که دیگر نیازی به یک شیء نیست، باید حافظه آن آزاد شود
  - یعنی این شیء از حافظه حذف شود
- در برخی زبان‌ها، این کار توسط برنامه‌نویس انجام می‌شود
  - برنامه‌نویس مشخص می‌کند که در چه زمانی، حافظه یک متغیر آزاد شود
  - مثال: عملگر delete در زبان C++
- آزادسازی حافظه توسط برنامه‌نویس فرایندی پرخطا و پیچیده است
  - ممکن است به اشتباه و یا نشت حافظه (Memory Leak) منجر شود

زباله‌روبی (Garbage Collection)



- خبر خوب: فرایند آزادسازی حافظه در جاوا خودکار است!
- نیازی به دخالت برنامه‌نویس نیست
- این کار توسط موجودی به نام زباله‌روب (Garbage Collector) انجام می‌شود
- زباله‌روب بخشی از JVM است

که به صورت خودکار اشیاءی که دیگر در برنامه استفاده نمی‌شوند، آزاد می‌کند

- زباله‌روب به طور متناوب فضای حافظه Heap را بررسی می‌کند
- و **اشیاء مرده** را دور می‌ریزد
- و حافظه را برای نگهداری اشیاء جدید آزاد می‌کند

ارسال پارامترها
Parameter Passing

فراخوانی متدها

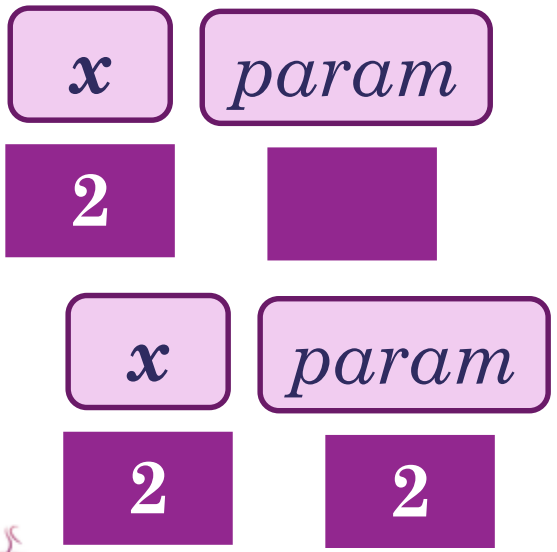
- فراخوانی متد ممکن است با ارسال پارامتر همراه باشد
- نحوه ارسال پارمترها به متدها در جاوا چگونه است؟
  - Call by value
  - Call by reference
  - Call by pointer
- ارسال متغیرهای primitive (مثل int) : call by value
- ارسال اشیاء (مثل String) : شبیه به call by pointer

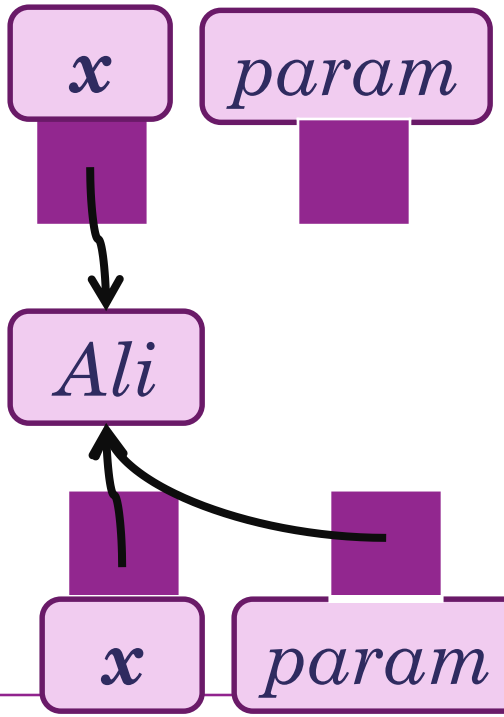
پارامترها چگونه به متدها پاس می‌شوند

```
Type x = ...
f(x);
```

```
void f(Type param){
   ...
}
```

```
void f(){
   Type param = x;
...
}
```





مثال

```
public void javaMethod(
 Person first, Person second, int number){
  first.age = 12;
  number = 5;
  Person newP = new Person();
  second = newP;
}
javaMethod(p1, p2, myInt);
```

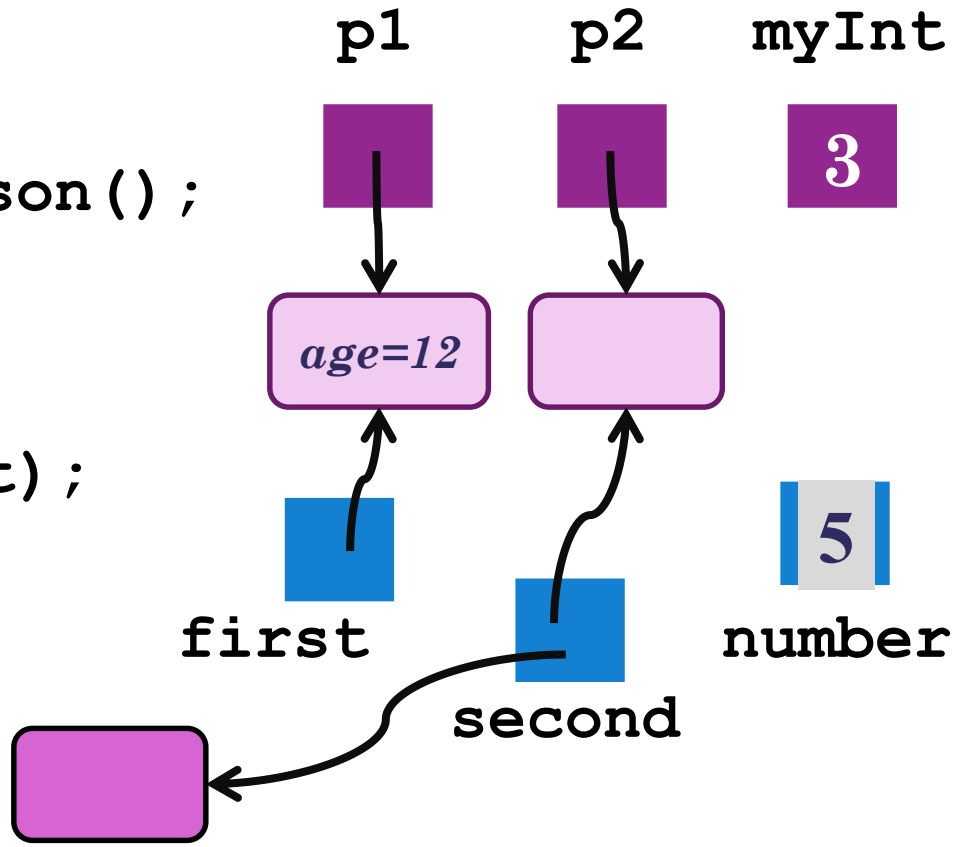
بعد از فراخوانی javaMethod

- آیا p1.age تغییر می‌کند؟
  - بله
- آیا myInt تغییر می‌کند؟
  - خیر
- آیا p2 تغییر می‌کند؟
  - خیر

مثال

```
public void javaMethod(
 Person first, Person second, int number){
  first.age = 12;
  number = 5;
  Person newP = new Person();
  second = newP;
}
javaMethod(p1, p2, myInt);
```

- آیا p1.age تغییر کرد؟ بله
- آیا myInt تغییر کرد؟ خیر
- آیا p2 تغییر کرد؟ خیر



کوییز

بعد از فراخوانی badSwap مقدار a و b چه خواهد بود؟

```
static void badSwap(int var1, int var2){
   int temp = var1;
   var1 = var2;
   var2 = temp;
}

public static void main(String[] args) {
     int a = 5;
     int b = 4;
     badSwap(a, b);
}
```

بعد از فراخوانی badSwap مقدار a و b چه خواهد بود؟

```
static void badSwap(String var1, String var2){
   String temp = var1;
   var1 = var2;
   var2 = temp;
}

public static void main(String[] args) {
     String a = "5";
     String b = "4";
     badSwap(a, b);
}
```

خروجی قطعه‌برنامه زیر چیست؟

```
public static void swapNames(Student s1, Student s2){
  String tmp = s1.name;
  s1.name = s2.name;
  s2.name = tmp;
}
```

```
Student a = new Student();
Student b = new Student();
a.setName("Ali");
b.setName("Taghi");
swapNames(a, b);
System.out.println(a.name);
System.out.println(b.name);
```

خروجی این برنامه چیست؟

```
public class ParameterPassing {
  public static void main(String[] args) {
    int[] array = new int[4];
    for (int i = 0; i < array.length; i++) {
       array[i] = i;
    }
    f(array);
    System.out.println(array[2]);
  }

  private static void f(int[] a) {
    a[2] = 0;
    for (int i = 0; i < a.length; i++) {
      int x = a[i];
      x= 5;
    }
    a = new int[10];
    a[2] = 1;
  }
}
```

```
for (int x : a) {
  x=5;
}
```

اگر حلقه for این گونه (با for each) بازنویسی شود چطور؟

بخش‌های مختلف حافظه

بخش‌های حافظه

- داده‌های یک برنامه (متغیرها) در حافظه نگهداری می‌شوند
  - به صورت عادی در RAM
- یک برنامه جاوا، از بخشی از حافظه استفاده می‌کند
- حافظه‌ی یک برنامه، شامل دو بخش مهم است: Stack و Heap
- اشیاء در Heap قرار می‌گیرند
- متغیرهای محلی هر متد روی Stack قرار می‌گیرند

Stack و Heap

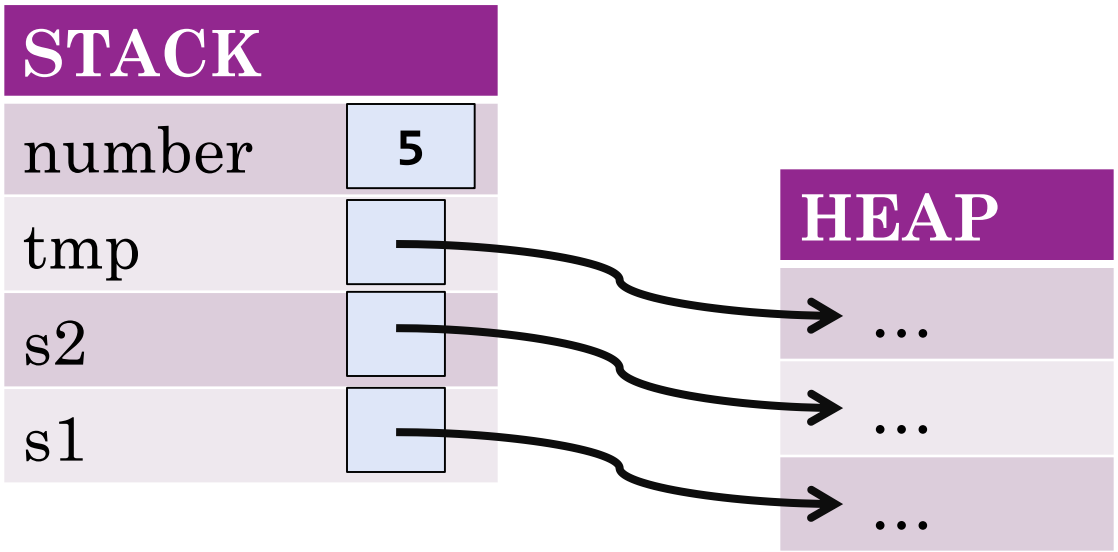
- همه اشیاء در Heap قرار می‌گیرند
  - هر شیء که new می‌شود، روی Heap قرار می‌گیرد
  - اشیاءی که می‌میرند: در زمان لازم توسط زباله‌روب آزاد می‌شوند

- متغیرهای محلی هر متد روی Stack قرار می‌گیرند
  - و در پایان فراخوانی متد، به صورت خودکار از Stack حذف می‌شوند
  - این فرایند نیازی به زباله‌روب ندارد
  - در زبان‌های بدون زباله‌روب (مثل C++) هم انجام می‌شود

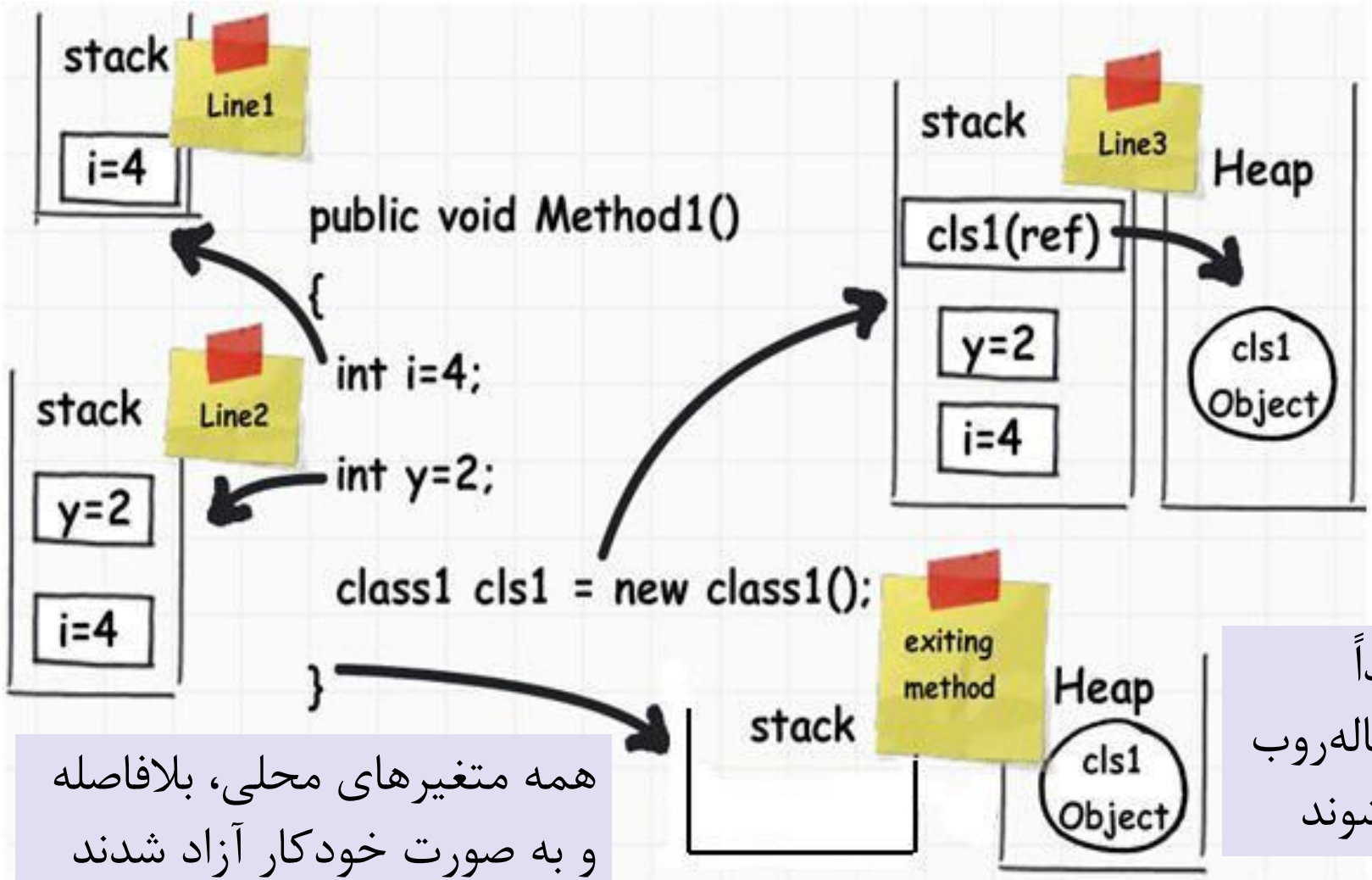
مثال

- در برنامه زیر، کدام متغیرها در Stack و کدام متغیرها در Heap جای دارند؟

```
public static void swapNames(Student s1, Student s2){
   String tmp = s1.name;
   s1.name = s2.name;
   s2.name = tmp;
   int number = 5 ;
}
```



مثال ۲



اشیاء بعداً توسط زباله‌روب آزاد می‌شوند

همه متغیرهای محلی، بلافاصله و به صورت خودکار آزاد شدند

تنظیم اندازه Stack و Heap برای یک برنامه

- هر برنامه جاوا بر روی یک JVM اجرا می‌شود
- JVM مدیریت حافظه هر برنامه را بر عهده دارد
- می‌توانیم JVM را تنظیم کنیم
  - تا میزان حافظه بیشتر یا کمتری در اختیار برنامه قرار دهد
  - با کمک آرگومان‌هایی که برای jvm ارسال می‌کنیم

- java Person
- java -Xms512m -Xmx3750m Person
- java -Xss4m Test
- java -Xmx3750m -Xss4m Main

آرگومان	معنی
-Xms	اندازه اولیه Heap
-Xmx	حداکثر اندازه Heap
-Xss	حداکثر اندازه Stack

کوییز

کوییز

- اگر برنامه‌ای اشیاء فراوانی را new کند، کدام بخش حافظه‌اش پر می‌شود؟

```
OutOfMemoryError: Java heap space
```

- اگر یک متد را به صورت بازگشتی صدا بزنیم، طوری که هیچ شرط پایانی نداشته باشد، Stack سرریز می‌شود یا Heap ؟

مثلاً با فراخوانی این متد، چه می‌شود؟

```
int f(int i){
  return f(i+1);
}
```

```
StackOverflowError
```

کوییز

- برنامه‌ای با نام HeapGames نوشته‌ام
- که در زمان اجرا، دچار این خطا می‌شود:
- `OutOfMemoryError: Java heap space`
- اما مطمئنم برنامه من اشکالی ندارد و کامپیوتر من حافظه کافی برای اجرای آن دارد.
- چه باید بکنم؟

- پاسخ: با کمک **–Xmx** حداکثر ممکن اندازه Heap را افزایش دهید
- مثلاً: java –Xmx4096m HeapGames

تمرین عملی

تمرین عملی

- نوشتن برنامه‌ای که باعث سرریز Stack شود
- افزایش اندازه مجاز برای Stack

- نوشتن برنامه‌ای که باعث سرریز Heap شود
- تغییر اندازه کمینه و بیشینه Heap

جمع‌بندی

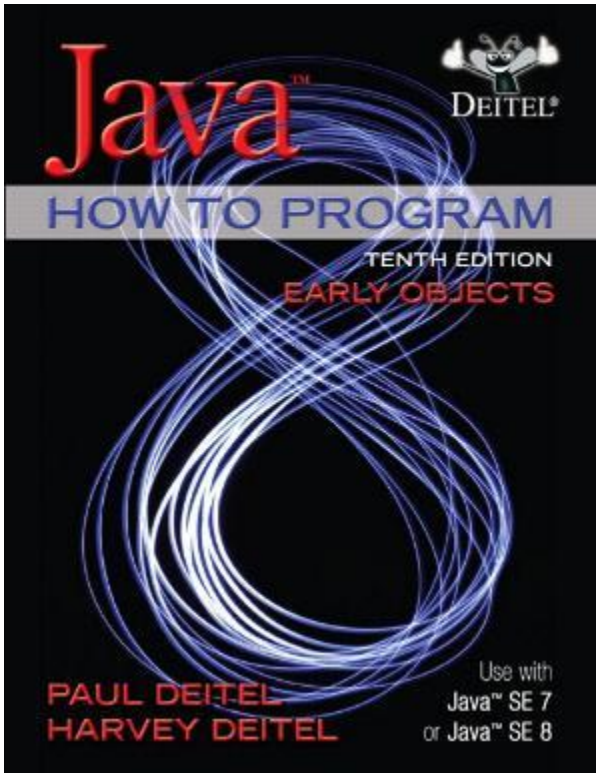
جمع‌بندی

- مفهوم ارجاع به اشیاء
- وضعیت اشیاء در حافظه
- تفاوت primitive types و reference types
  - آرایه‌ها و رشته‌ها هم شیء هستند
- زباله‌روب
- نحوه ارسال پارامتر به متدها در جاوا
- بخش‌های مختلف حافظه: Stack و Heap

مطالعه کنید

- فصل‌های ۳ و ۶ از کتاب دایتل

Java How to Program (Deitel & Deitel)



3- Introduction to Classes, Objects, Methods and Strings

6- Methods: A Deeper Look

- تمرین‌های همین فصل‌ها از کتاب دایتل

تمرین

- سعی کنید برنامه‌هایی بنویسید که Stack را سرریز کنند.
- با ایجاد تعداد زیادی شیء کوچک، برنامه‌ای بنویسید که Heap را سرریز کند.
- با ایجاد تعداد کمی شیء بسیار بزرگ، برنامه‌ای بنویسید که Heap را سرریز کند.
- تنظیمات حافظه JVM را تمرین کنید
  - به صورت command line
  - در محیط توسعه نرم‌افزار (مثلاً Eclipse)

جستجو کنید و بخوانید

- Heap, Stack
- Heap generations
- Garbage Collection Algorithms
- Call by value, Call by pointer, Call by reference

- گونه‌های ارسال پارامتر به متد در C++ :

```
void cppMethod(Person byValue,
  Person* byPointer, Person& byReference){...
```

- گونه‌های ارسال پارامتر به متد در C# :

```
void cSharpMethod(ref int a, ref Person p){...
```

پایان

تاریخچه تغییرات

نسخه	تاریخ	توضیح
۱.۰.۰	۱۳۹۴/۰۲/۲۸	نسخه اولیه ارائه آماده شد